MongoDB

Escalabilidad en almacenes de datos

Índice

- Factores que afectan al rendimiento y escalabilidad
- Rendimiento con índices
- Disponibilidad con replicación
- Escalabilidad mediante clustering
- Herramientas de administración y operación
- Seguridad
- Tips & tricks

Factores de impacto en el rendimiento y escalabilidad

Factores que afectan al rendimiento

- CPU: Nº de procesadores, cores/procesador, velocidad de reloj… Potencia!
- Disco: HDD vs SSD
- Memoria RAM
- Índices!!!!!
- Volumetría de los datos a almacenar

Técnicas disponibles en MongoDB

- Índices: para incrementar rendimiento de consulta
- Replicación (Replica Sets):
  - Para incrementar la disponibilidad
  - Para incrementar el throughput de consultas
- Particionamiento y Clústering (sharding):
  - Para incrementar disponibilidad
  - Para incrementar la escalabilidad en datos y throughput

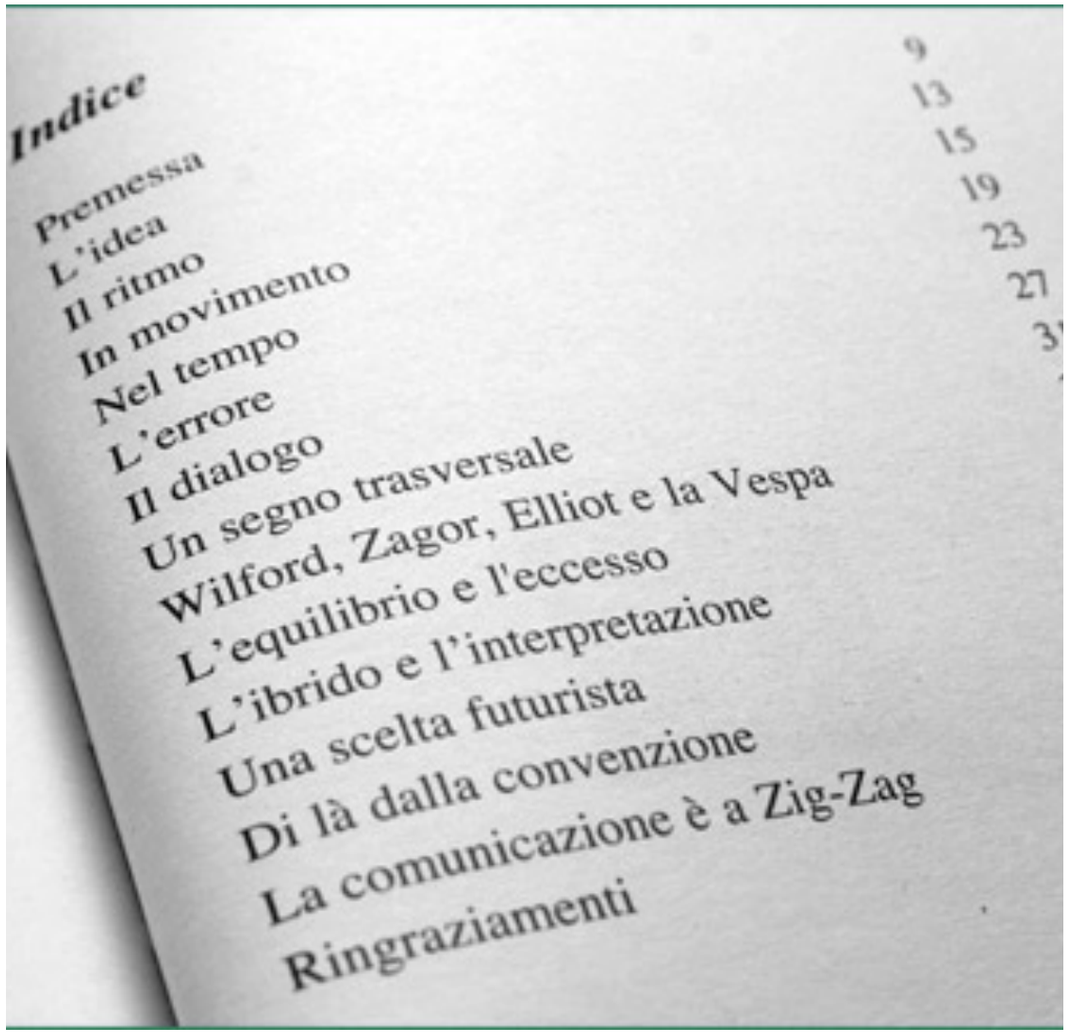
Rendimiento con índices

¿Qué es un índice?

- Ejercicio mental 1: averiguar el precio de una ensalada en el menú del VIPs

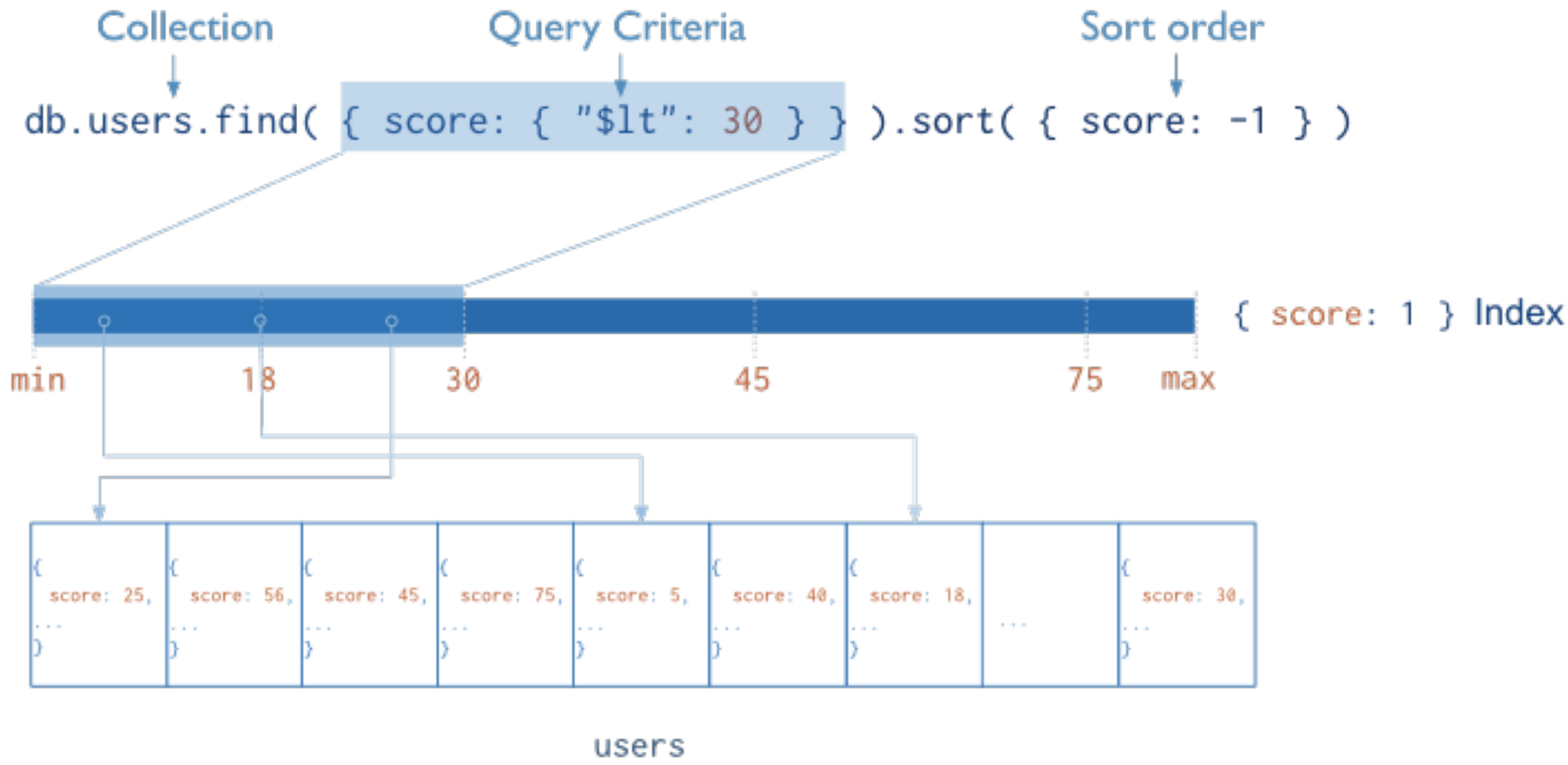
- Ejercicio mental 2: consultar en la Odisea de Homero de qué estaba disfrazada Atenea cuando Odiseo llega a Itaca

- Ejercicio mental 3: averiguar la fecha de nacimiento de Tolkien en la Enciclopedia Británica



Índices en MongoDB

La colección de usuarios está indexada por puntuación:



Índices en MongoDB

- El índice es una estructura de datos adicional a la colección de documentos.

- El índice almacena una porción muy reducida de datos de una colección: un campo o un conjunto de campos de cada documento de la colección.

- El índice almacena esos datos de manera ordenada. Creciente o decreciente.

- MongoDB se apoya en índices (si existen) para obtener los documentos ordenados.

- En ausencia de índices, MongoDB deberá recorrer TODOS los documentos (*full scan*) de la colección para obtener resultados de consulta.

    - Debemos evitar consultas *full scan* a toda costa.

Índices en MongoDB

- MongoDB siempre indexa las colecciones por el identificador de cada documento:
  - El campo `_id`
  - Recordatorio: `_id` es único; si no lo creamos, MongoDB generará un valor de tipo `ObjectId` en el momento de la inserción.
  - No pueden existir dos documentos con el mismo `_id`

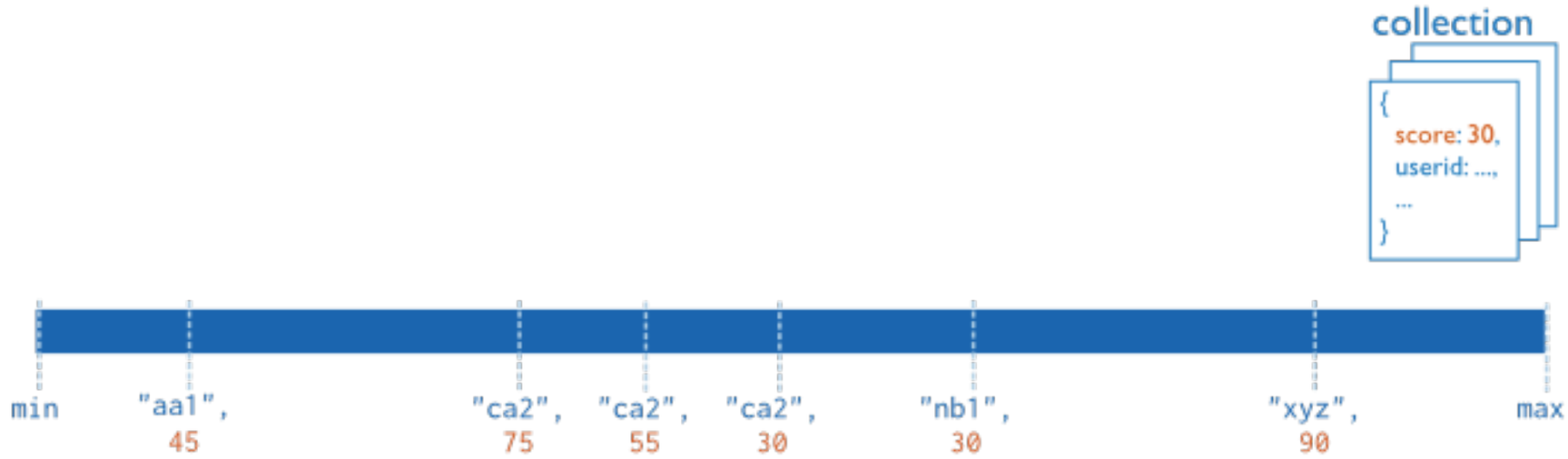
Tipos de índices: Simple

- Indexa un único campo del documento

- `db.users.createIndex( { “score”: 1 } )`
- `1` para indexar de forma creciente y `-1` para decreciente
- MongoDB puede recorrer el índice en ambas direcciones

Tipos de índices: Compuesto

- Indexa por varios campos del documento



- `createIndex({“userid” : 1, “score”: -1})`
- Indexará primero por `userid` creciente y después por `score` decreciente.

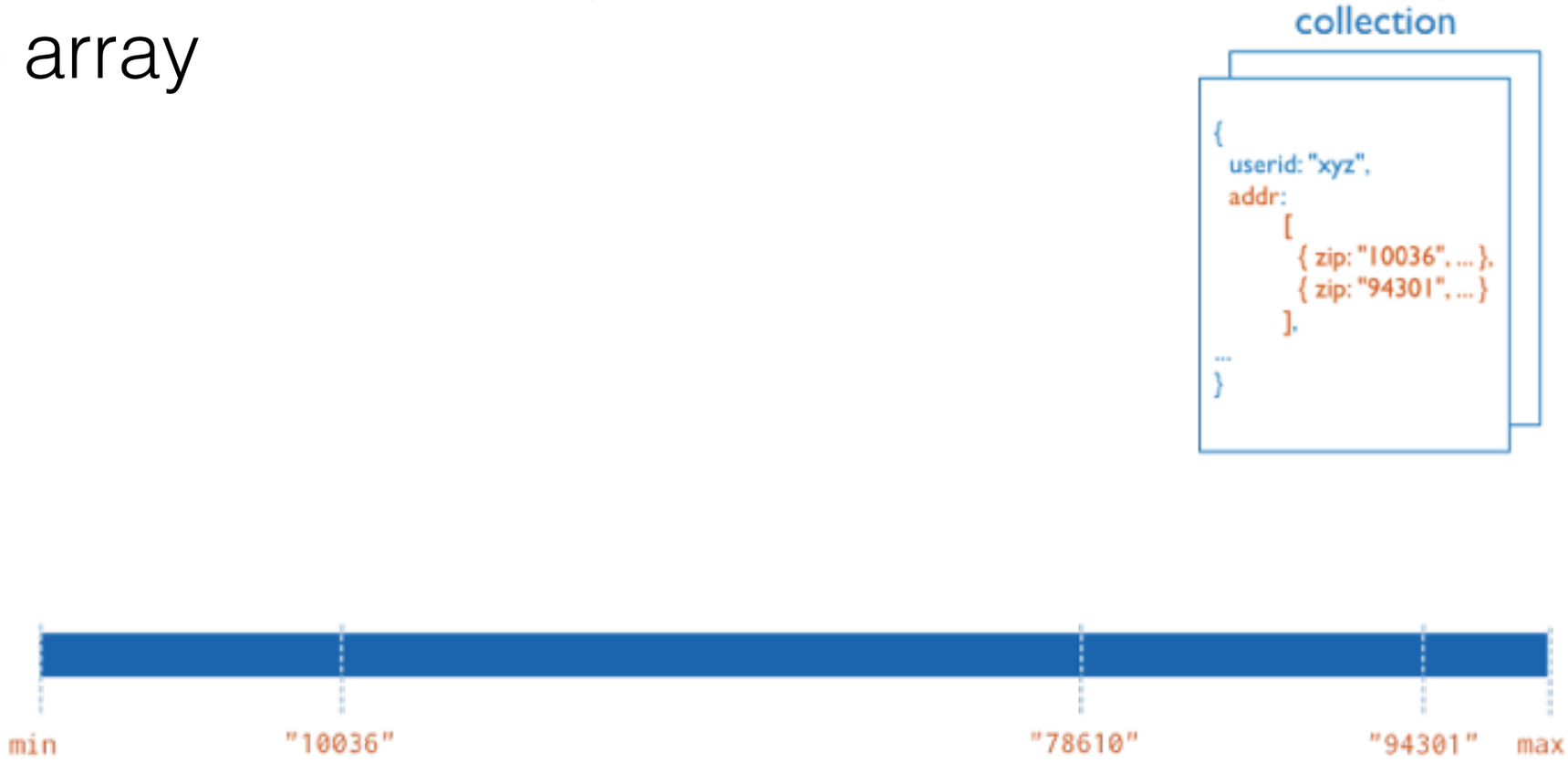
Tipos de índices: Compuesto

- El índice compuesto puede soportar o no consultas ordenadas.

- El índice `{"userid":1, "score":-1}` soporta
  - `db.users.find().sort({userid:1, score:-1})`
  - `db.users.find().sort({userid:-1, score:1})`
- El índice `{"userid":1, "score":-1}` NO soporta
  - `db.users.find().sort({userid:1, score:1})`

Tipos de índices: Multiclave

- Indexa documentos por el contenido de campos de tipo array



- `db.users.createIndex( { “addr.zip”: 1 } )`

Tipos de índices: geoespacial

- Indexa campos espaciales en geometría plana (índice `2d`) o esférica (índice `2dsphere`)

- `db.users.createIndex({"direccion.coord": “2d"})`

- Los usa el comando `geoNear` para buscar documentos cerca de una localización dada:

```
db.runCommand(
   {
     geoNear: "restaurantes",
     near: { type: "Point", coordinates: [ -73.9667, 40.78 ] },
     spherical: true,
     query: { tipo : "italiano" },
     minDistance: 3000,
     maxDistance: 7000
   }
)
```

Tipos de índices: textuales

- Indexa campos que contienen texto.
- El índice almacena solo raíces de palabras. No almacena *stop words*.

```
• db.posts.createIndex({comments:"text"})
• db.posts.createIndex({"$**":"text"})
• db.posts.find({$text:{$search:"coffee -cake"}})
• db.posts.find({$text:{$search:"\"coffee cake\""}})
```

Tipos de índices: hashed

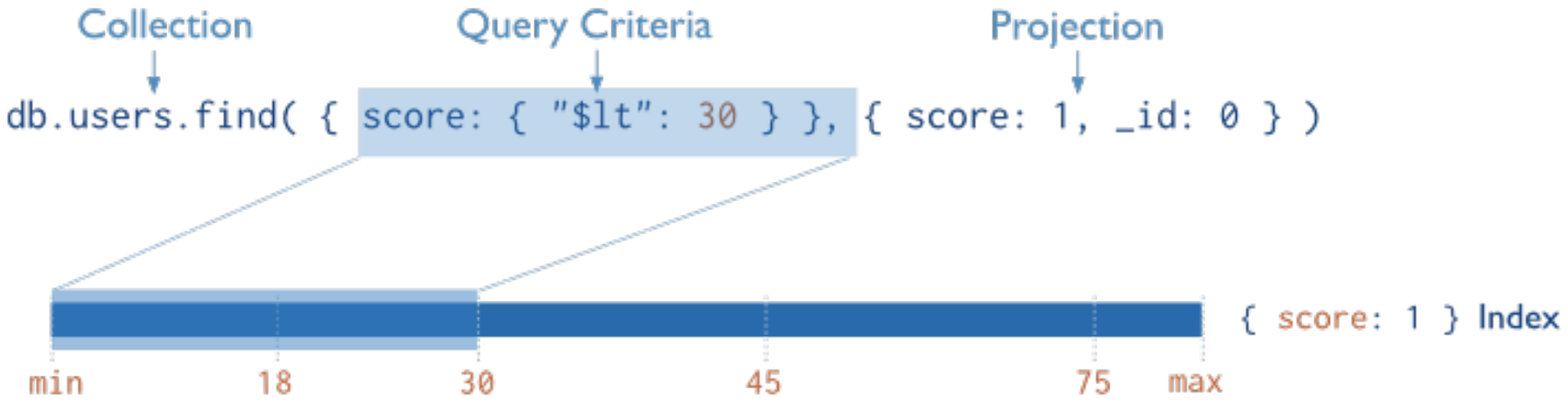
- Indexa el valor hash de un campo del documento.
- `db.coll.createIndex({a:"hashed"})`
- No soporta consultas de rangos del dato indexado.
- Útil para particionar colecciones (sharding) muy grandes en un cluster de MongoDB.

Propiedades de los índices

- Índices únicos: para restringir que el campo indexado no contenga duplicados:
  - `db.members.createIndex({"dni":1},{unique:true})`
- Índices *sparse*: el índice solo contiene aquellos documentos que tienen algún valor en el campo a indexar. Los índices geoespaciales y textuales son *sparse* por defecto.
- Índices TTL (*time to live*): para marcar documentos que MongoDB eliminará automáticamente transcurrido el tiempo especificado:
  - `db.applog.createIndex({"lastModified":1},`
    `{expireAfterSeconds:3600})`

“Covered queries”

- El índice “cubre” la consulta cuando:
  - Todos los campos de la consulta forman parte del índice, y
  - todos los campos del resultado de la consulta están incluidos en el índice.



- Las consultas “cubiertas” son las más eficientes: se resuelven accediendo solo al índice, sin necesidad de llegar a la colección.

Limitaciones con índices

- Una colección no puede tener más de 64 índices.
- Una entrada en el índice no debe sobrepasar los 1024 bytes.
- El nombre de un índice no debe sobrepasar los 128 caracteres.
  Téngase en cuenta que el nombre de un índice concatenará el nombre de los campos incluidos y el tipo del índice.
- Un índice compuesto no podrá incluir más de 31 campos.
- No se pueden usar índices geoespaciales y textuales en la misma consulta.
- Un índice multi-clave no soporta *covered queries*.

Comandos para índices

- db.*collection*.getIndexes()
- db.*collection*.createIndex({“*field*”:1})
- db.*collection*.dropIndex(*index-name*)

Ejercicio 1

1. Arrancar la VM suministrada
2. Arrancar la shell de MongoDB: `mongo`
3. Usar la base de datos `test`: `use test`
4. Averiguar las colecciones suministradas y visualizar el formato de los documentos
5. Crear un índice en el campo `quantity` de la colección `inventory`. ¿Cuántos índices tiene ahora la colección? ¿Por qué? Pista: consultar los índices de la colección.
6. Ejecutar el comando `db.inventory.find({"quantity":{$gt:100}}, {"quantity":1,"_id":0}).explain("executionStats")`
7. Revisaremos la función `explain` a continuación.

Optimización de queries

- MongoDB usa internamente un optimizador de queries para evaluar cuál es la forma óptima de ejecutarla (el "plan") en función de los índices disponibles.

- El optimizador re-evalúa periódicamente los planes de ejecución según el contenido de la colección cambia.

- La función `explain()` permite acceder a la información de planes de ejecución para poder desarrollar estrategias de indexación eficientes.

Optimización de queries

- Formas de ejecutar `explain()`:
  - **`db.collection.explain().update(), db.collection.explain().aggregate(), db.collection.explain().remove(), db.collection.explain().count(), …`**
  - **`db.collection.explain().find(), db.collection.find().explain()`**
  - **`cursor.explain()`**
- Argumentos de `explain(`*`verbosity`*`)`:
  - verbosity = "**`queryPlanner`**" (por defecto): se ejecuta el optimizador, se determina el plan ganador y se devuelve información de éste.
  - verbosity = "**`executionStats`**": devuelve información y estadísticas solo del plan ganador.
  - verbosity = "**`allPlansExecution`**": devuelve información completa del plan ganador y de los planes descartados.

Optimización de consultas

```
> db.inventory.find({“quantity”:{$gt:100}},{"quantity":1, "_id":
0}).explain("executionStats")

        {
            "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "test.inventory",
                "indexFilterSet" : false,
                "parsedQuery" : {
                    "quantity" : {
                        "$gt" : 100
                    }
                },
                "winningPlan" …
```

Optimización de consultas

```
"winningPlan" : {
    "stage" : "PROJECTION",
    "transformBy" : {
        "quantity" : 1,
        "_id" : 0
    },
    "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
            "quantity" : 1
        },
        "indexName" : "quantity_1",
        "isMultiKey" : false,
        "direction" : "forward",
        "indexBounds" : {
            "quantity" : [
                "(100.0, inf.0]"
            ]
        }
    }
},
```

Un plan es un árbol de *stages*. Cada stage pasa sus resultados al nodo padre. Los nodos hojas son *stages* que acceden a colecciones o índices. El nodo raíz es aquel del que se deriva el resultado de la query.

Tipos de stages:

- COLLSCAN: recorrido de colección
- IXSCAN: recorrido de clave de índice
- FETCH: recuperación de documento
- SHARD_MERGE: mezcla de resultados de varios shards

Optimización de consultas

```
"executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 7,
    "executionTimeMillis" : 0,
    "totalKeysExamined" : 7,
    "totalDocsExamined" : 0,
    "executionStages" : {
        "stage" : "PROJECTION",
        "nReturned" : 7,
        "executionTimeMillisEstimate" : 0,
        "works" : 8,
        "advanced" : 7,
        "needTime" : 0,
        "needFetch" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "invalidates" : 0,
        "transformBy" : {
            "quantity" : 1,
            "_id" : 0
        },
        "inputStage" : {: 0,
```

Número de documentos que cumplen la condición de la consulta.

Número de entradas del índice recorridas.

Número de documentos recorridos.

Optimización de consultas


```
"inputStage" : {
    "stage" : "IXSCAN",
    "nReturned" : 7,
    "executionTimeMillisEstimate" : 0,
    "works" : 8,
    "advanced" : 7,
    "needTime" : 0,
    "needFetch" : 0,
    "saveState" : 0,
    "restoreState" : 0,
    "isEOF" : 1,
    "invalidates" : 0,
    "keyPattern" : {
        "quantity" : 1
    },
    "indexName" : "quantity_1",
    "isMultiKey" : false,
    "direction" : "forward",
    "indexBounds" : {
        "quantity" : [
            "(100.0, inf.0]"
        ]
    },
```

Optimización de consultas

```
            "keysExamined" : 7,
            "dupsTested" : 0,
            "dupsDropped" : 0,
            "seenInvalidated" : 0,
            "matchTested" : 0
        }
    }
        },
        "serverInfo" : {
            "host" : "ubuntu",
            "port" : 27017,
            "version" : "3.0.6",
            "gitVersion" : "1ef45a23a4c5e3480ac919b28afcba3c615488f2"
        },
        "ok" : 1
    }
```

https://docs.mongodb.org/manual/reference/explain-results/

Optimización de consultas

- Objetivos informales:
  - Evitar full scans a toda costa usando índices (`executionStats.totalDocsExamined` igual o cercano al número de documentos de la colección).
  - Minimizar en lo posible `executionStats.totalDocsExamined`.
  - Indexar de modo que `executionStats.nReturned` sea lo más cercano posible a `executionStats.totalKeysExamined`.

Ejercicio 2

1. Obtener el plan de ejecución de la consulta
   `db.inventory.find({"quantity":{$gt:100}})`

2. ¿Cuántos documentos se obtienen como resultado?

3. ¿Se hace *full scan* de la colección?

4. ¿Usa un índice esa consulta? ¿Cuál?

5. ¿Se puede optimizar todavía más?

6. ¿Por qué la consulta del ejercicio 1 no examinaba documentos y esta sí?

7. Borrar el índice usado y explicar de nuevo la consulta. ¿Cuántos documentos se recorren?

Ejercicio 3

Usar para este ejercicio la colección de restaurantes.

1. Optimizar la consulta que obtiene los nombres de restaurantes para un código postal dado.

2. Optimizar la consulta que obtiene los nombres de restaurantes de un código postal dado con buena puntuación.

3. ¿Es redundante el índice creado para 1 con el generado para 2? ¿Por qué?

4. Para nota: obtener de manera óptima los nombres de los restaurantes cerca de las coordenadas -73.9791458, 40.744328.

Cuestiones avanzadas

- Ordenación de resultados e índices.
- Intersección de índices:
  - Internamente MongoDB puede decidir usar la intersección de varios índices o un índice compuesto para resolver el criterio de consulta.
    - `db.orders.createIndex({“qty":1})`
    - `db.orders.createIndex({“item":1})`
    - `db.orders.find({item:”abc123", qty:{$gt:15}})`
- La intersección de índices puede tener mejor rendimiento que usar índices compuestos.

Tips y buenas prácticas con índices

- Monitorizar uso de índices. https://github.com/jwilder/mongodb-tools
- Eliminar índices que no se usan. Su mantenimiento implica consumo de RAM y escritura en disco en consultas de escritura.
- Eliminar índices que son prefijos de otros índices. Es óptimo usar un índice `{"nombre" : 1, "apellidos" : 1}` para filtrar solo por nombre.
- Usar almacenamiento WiredTiger (desde v3.0): permite almacenar índices en volúmenes separados, compresión de índices en memoria y más.
- Evitar buscar con expresiones regulares sin anclaje a la izquierda: `^` es muy ineficiente si lo comparamos con `$`.

Tips y buenas prácticas con índices

- Usar `explain` desde el principio para estudiar cómo se resuelven las consultas:
  - Cuántos documentos se devuelven.
  - Si se usan índices y cuáles.
  - Si la consulta está cubierta por índices.
  - Si se realiza una costosa ordenación en RAM.
  - Cuántas entradas del índice de consultan.
  - Cuánto se tarda en ejecutar la consulta.

Disponibilidad mediante replicación: MongoDB Replica Sets

Replica Set

- Replica Set es el mecanismo de MongoDB para tener replicas (redundancia) de datos en varias instancias.

- Sirve para incrementar la disponibilidad de los datos.

- La redundancia protege frente a fallos de hardware e interrupciones de servicio que afecten a algunos servidores.

- En algunos casos permite incrementar la capacidad de lectura (throughput de consulta).

Replica Set

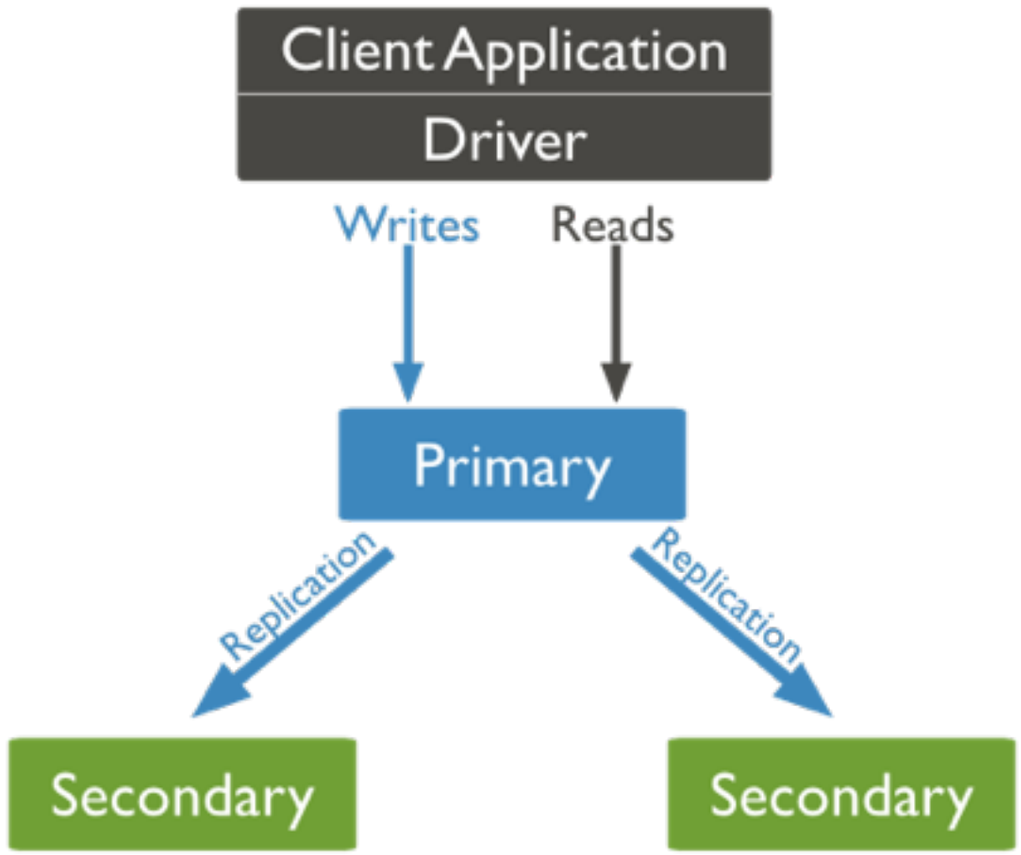
- Es un conjunto de instancias de `mongod` que alojan el mismo dataset. Hasta 50.

- Una instancia es el nodo primario:

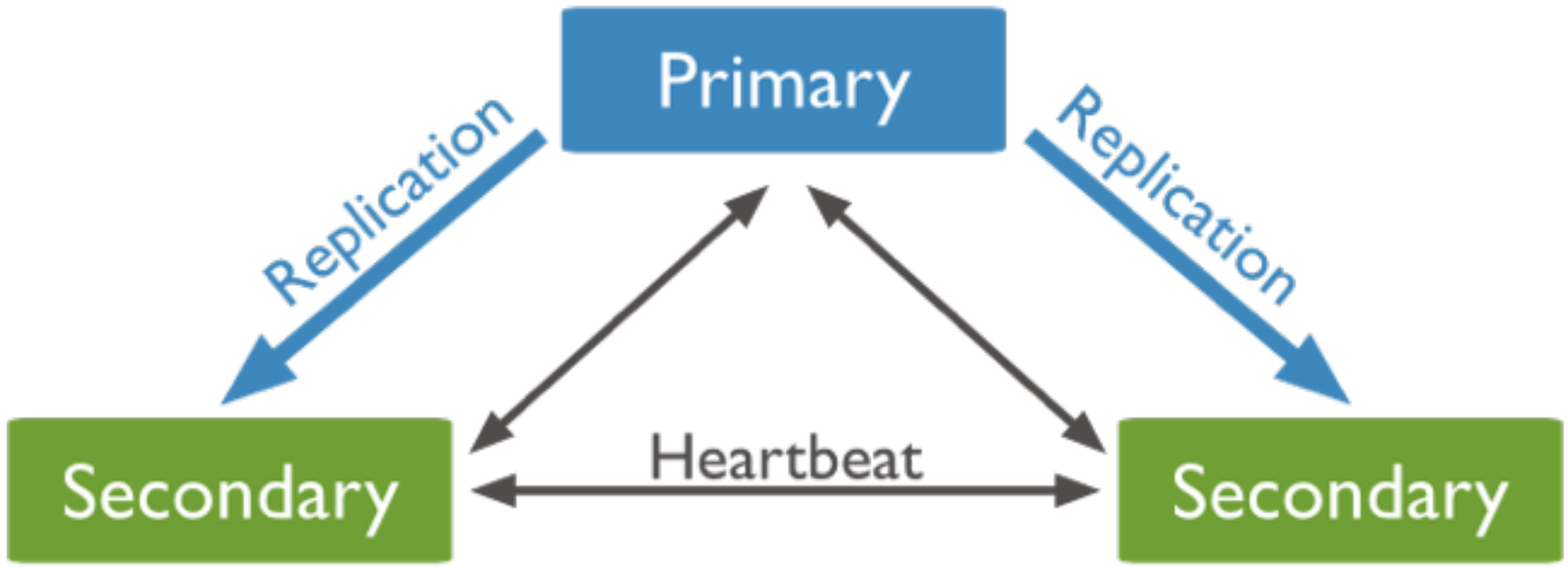
  - Ejecuta todas la operaciones de escritura.

  - Replica los cambios en los nodos secundarios.

- Los nodos del RS se instalarán en máquinas separadas.

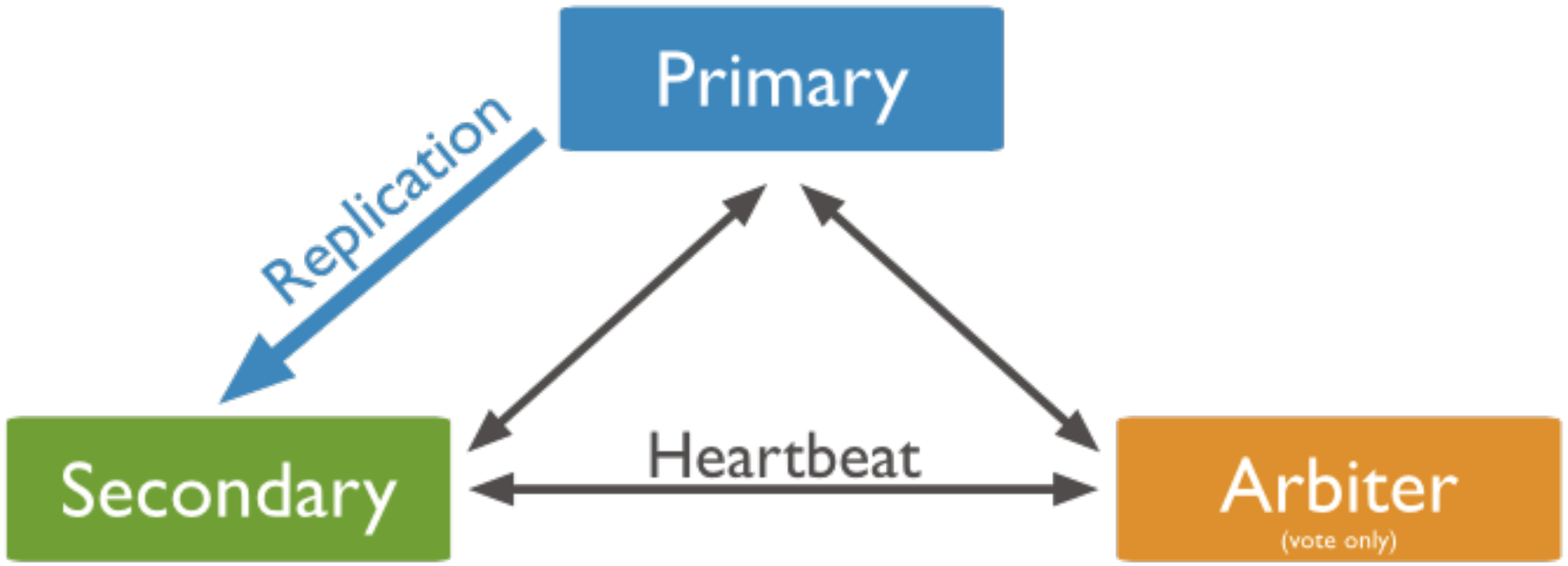


Replica Set



- Los secundarios replican el oplog del primario y aplican a sus datos los cambios ahí registrados.
- Los nodos del replica set se comunican entre sí…
- …de tal modo que si un primario se vuelve inaccesible (10-30 segundos), un secundario accesible pasará a ser primario.
- La conversión en primario se basa en votación de los nodos del replica set (otros 10-30 segundos).

Replica Set



- Para que en las votaciones a primario haya quórum se suelen añadir nodos árbitro al RS.
- Se recomienda que el número de nodos votantes de un RS sea impar y se consigue añadiendo árbitros si es necesario.
- Un árbitro no almacena datos ni puede convertirse en primario ni secundario, solo existe, atiene y vota.
- Los árbitros apenas consumen recursos y no se necesita instalarlos en hardware dedicado.

¿Cómo se replican los datos?

- La replicación/sincronización es asíncrona:
  - El primario aplica la operación y…
  - …la escribe en el `oplog`…
  - …los secundarios replican el `oplog` (de cualquier otro!)…
  - …y aplican los cambios ahí registrados

¿Cómo se replican los datos?

- Dos tipos de sincronización:
  - Sincronización completa: un miembro del RS copia el dataset entero de otro miembro
    - Primero clona las bases de datos
    - Luego aplica los cambios del `oplog`
    - Por último construye los índices
- Replicación
  - Aplica continuamente los cambios del `oplog`

¿Cómo se replican los datos?

- El `oplog`:
  - *Capped* collection: tamaño fijo y sobreescribe el elemento más antiguo cuando se llena la colección
  - Almacena solo las operaciones de modificación de datos
  - Las operaciones almacenadas son necesariamente idempotentes
  - En Linux se reserva el 5% del espacio disponible en disco, mínimo 1Gb hasta 50Gb
  - “Los secundarios deberán ser capaces de replicar y aplicar las operaciones del `oplog` a mayor ritmo que el primario”
  - “El tamaño del `oplog` no tiene porqué estar alineado con el tamaño del dataset”

Arquitecturas de RS

- Tres miembros: mínimo recomendado
- Cuatro o más miembros: ideal para gran redundancia y distribución de responsabilidades
- RS distribuidos geográficamente

Arquitecturas de RS

- "El tamaño importa"
  - La tolerancia a fallos de un RS es el número máximo de miembros que pueden quedar inaccesibles permitiendo que el RS sea capaz de elegir un primario
  - La tolerancia a fallos no crece al mismo ritmo que el número de miembros del RS

# miembros del RS	Mayoría necesaria	Tolerancia a fallos
3	3	1
4	3	1
5	3	2
6	4	2

Ejercicio 4

- Parar la instancia preconfigurada: `sudo service mongod stop`.
- Vamos a crear un RS de nombre `rs0` y tres nodos.
- Crear los directorios de cada uno de los nodos: `mkdir -p /srv/mongodb/rs0-0 /srv/mongodb/rs0-1 /srv/mongodb/rs0-2`
- Arrancar cada nodo en un terminal diferente:
  - `mongod --port 27017 --dbpath /srv/mongodb/rs0-0 --replSet rs0 --smallfiles --oplogSize 128`
  - `mongod --port 27018 --dbpath /srv/mongodb/rs0-1 --replSet rs0 --smallfiles --oplogSize 128`
  - `mongod --port 27019 --dbpath /srv/mongodb/rs0-2 --replSet rs0 --smallfiles --oplogSize 128`
- Abrir una sesión con el primer nodo: `mongo --port 27017`

Ejercicio 4

- Crear una configuración básica en la misma shell: `rsconf = {_id: "rs0", members:[{_id: 0, host: "localhost:27017"}]}`
- Inicializar el RS con un nodo: `rs.initiate( rsconf )`
- Añadir los dos nodos restantes:
  - `rs.add("localhost:27018")`
  - `rs.add("localhost:27019")`
- Verificar el estado del RS con `rs.status()`
- Hacerle una carga al RS: `mongoimport --host rs0/localhost:27017,localhost:27018,localhost:27019 --db test --collection inventory --file /home/usuario/Descargas/inventory-dataset.json`

Recapitulando RS

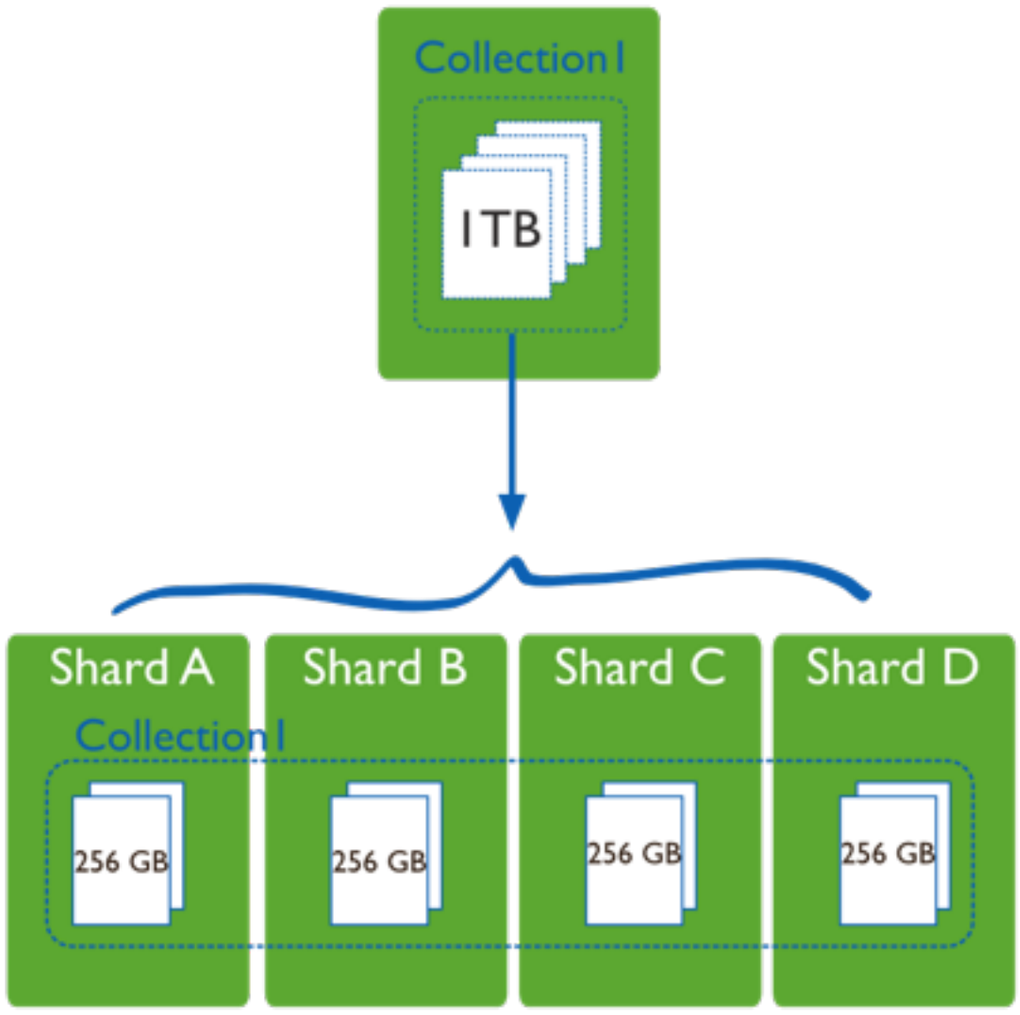
- Si el dataset cabe en un nodo no hay verdades necesidad de desplegar un RS.
- Los RS usan replicación de tipo master-slave.
- Solo el máster realiza operaciones de escritura.
- Puedo usar los esclavos para incrementar el throughput de lecturas. Pero la replicación es asíncrona!
- El proceso de elección de un nuevo primario no es inmediato y lleva unos cuantos segundos.
- Pasar de un despliegue de RS a un clúster es bastante sencillo.

Escalabilidad mediante clustering: MongoDB sharding

Sharding

- Sharding es el mecanismo de MongoDB para particionar datos y repartir peticiones entre varios servidores.

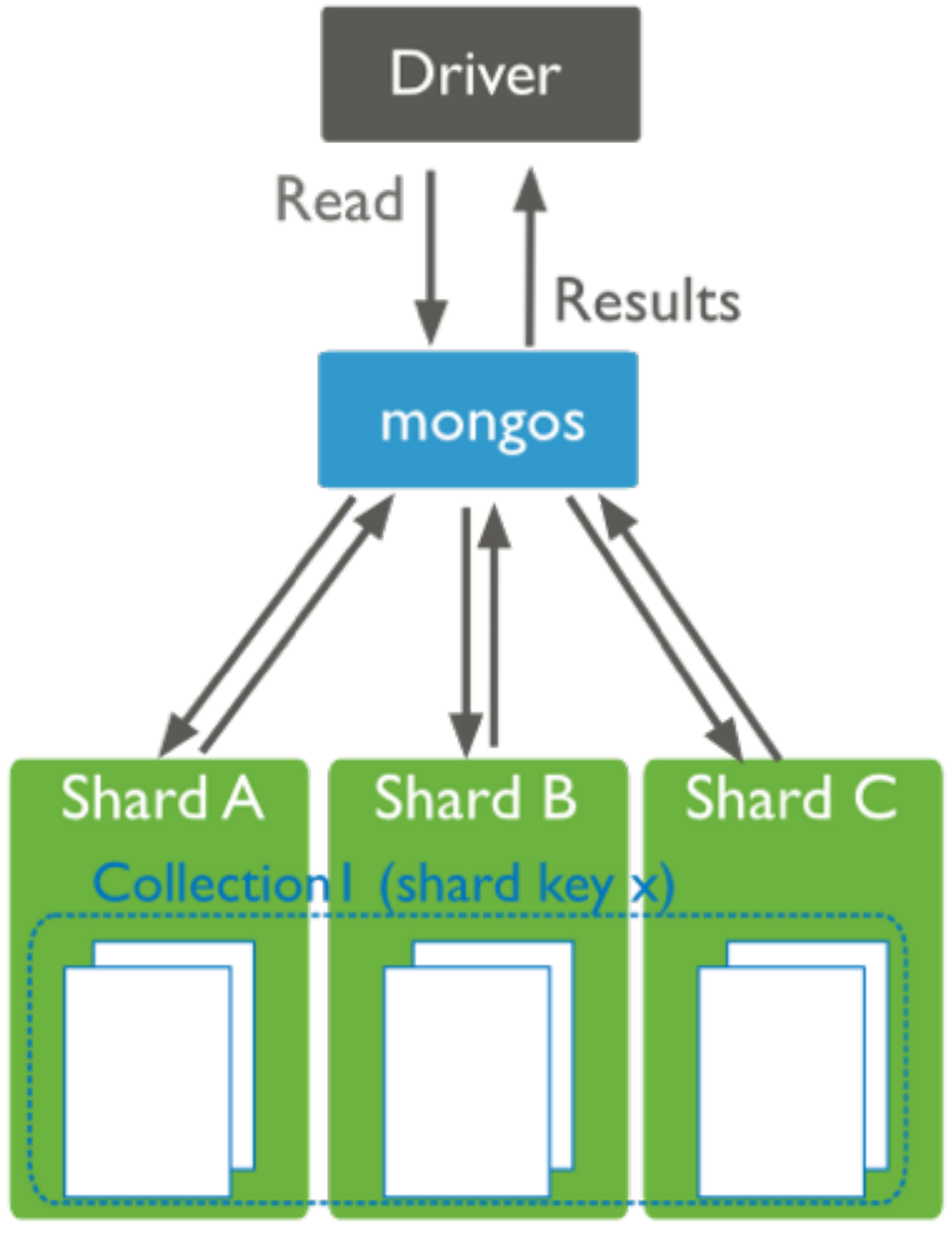
- Con sharding se consiguen almacenar enormes datasets dividiéndolos en múltiples instancias de MongoDB.



Sharding

- Con sharding se consigue aumentar el throughput cuando se ha llegado al límite de escalabilidad vertical.

- En definitiva, el sharding es el mecanismo de clustering de MongoDB.



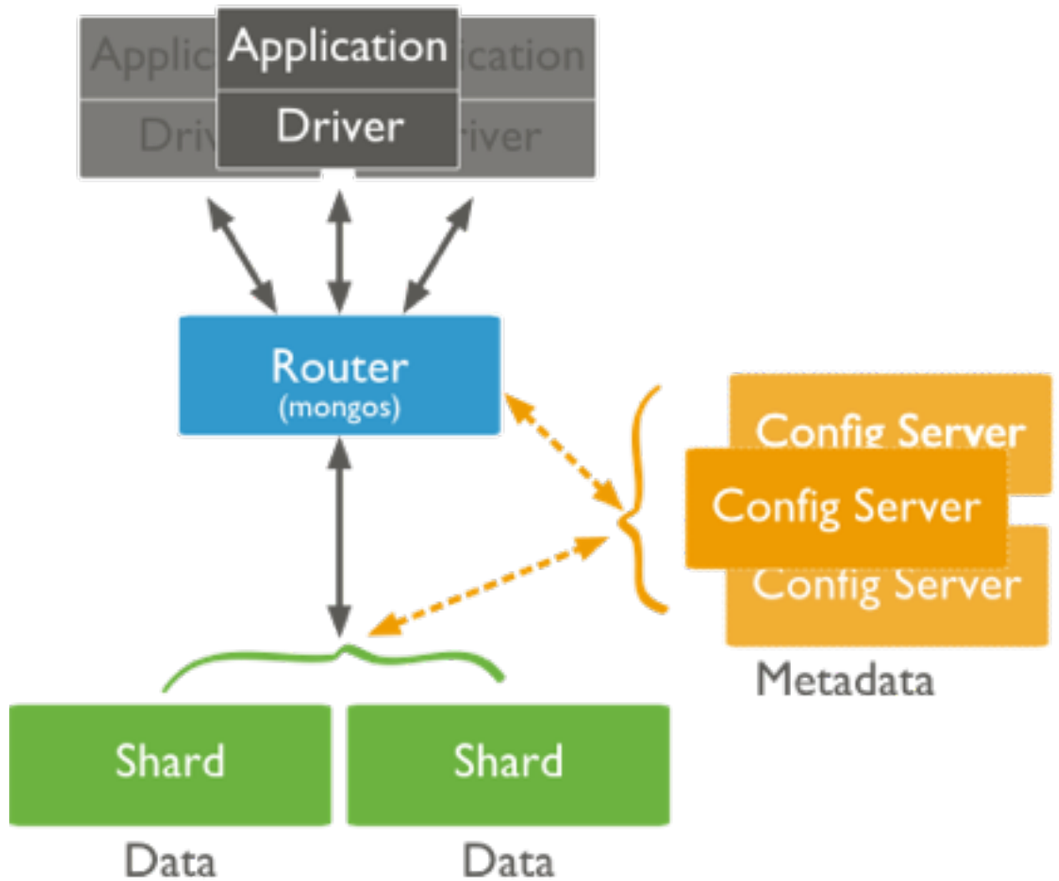
Shard

- Técnicamente hablando un shard es un replica set.

- Usa *config servers* para almacenar metadatos: la localización de los datos en el shard.

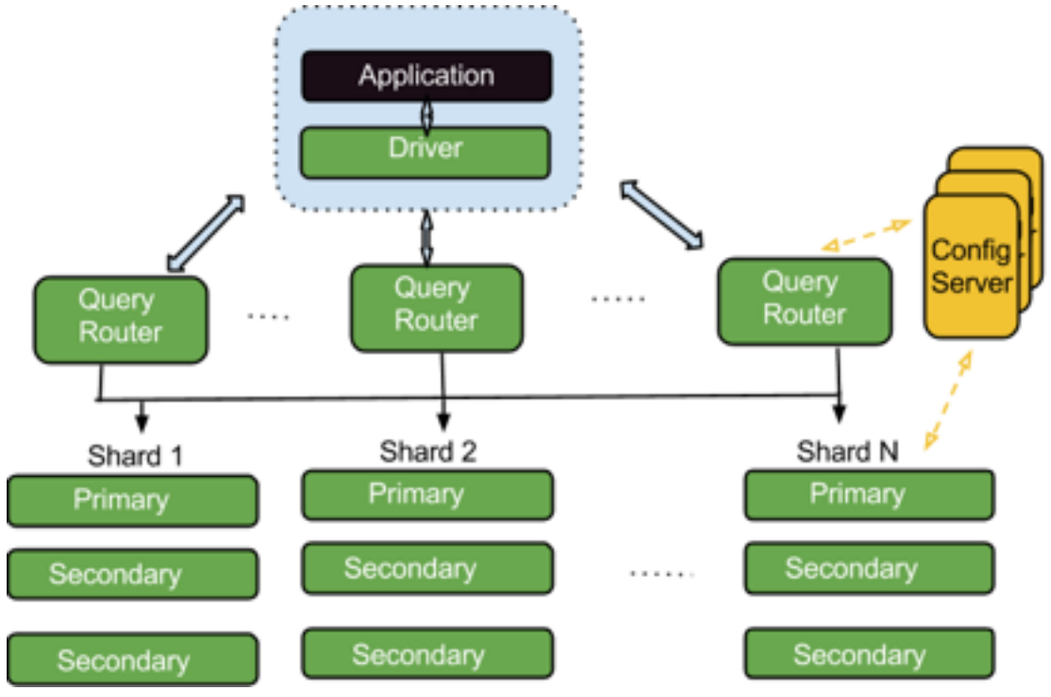
- Usa *query routers* (los `mongos`) para redirigir las operaciones a shards concretos.

- Los query routers usan los metadatos que almacenan los config servers para redirigir las operaciones.

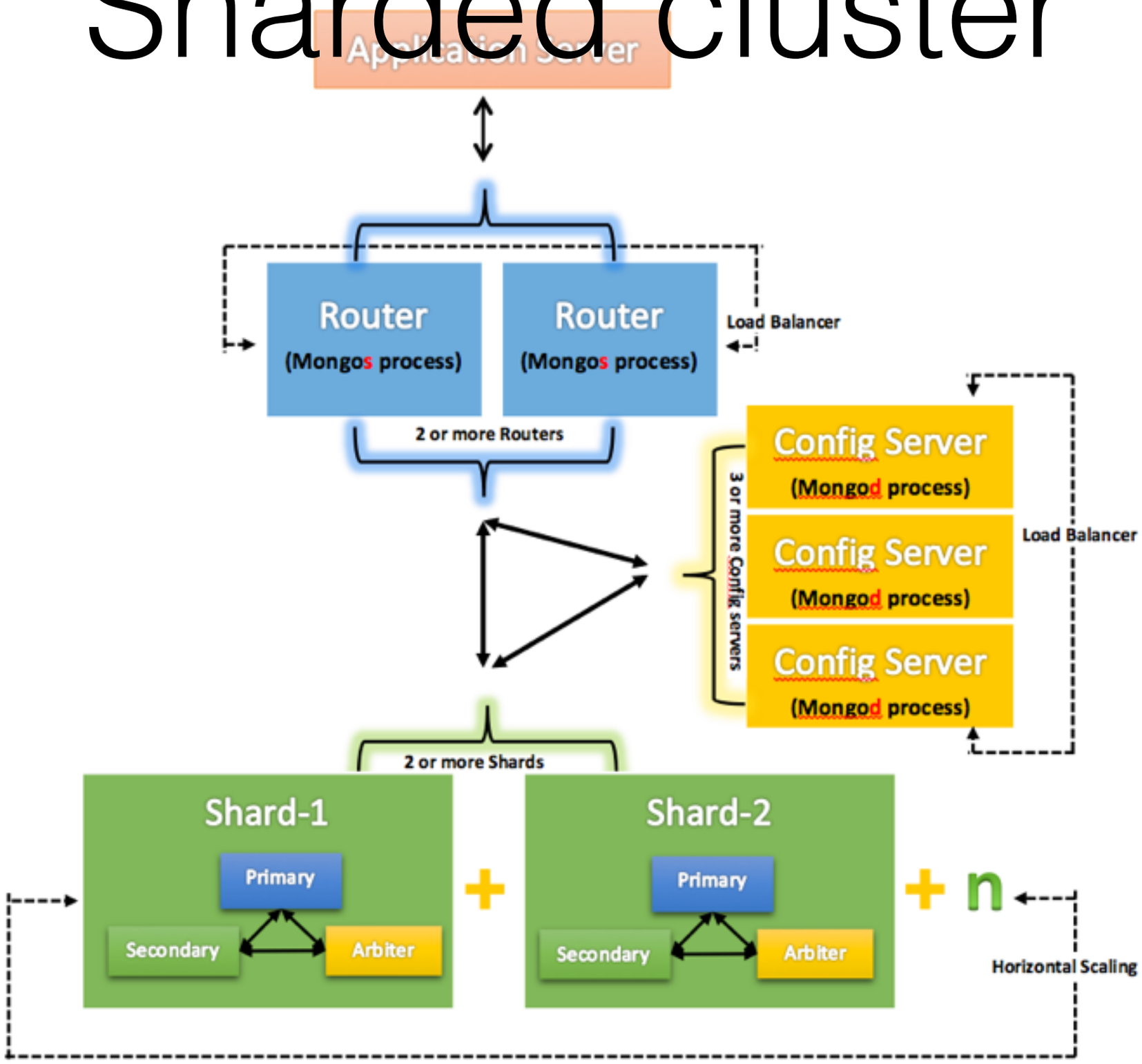


Shard

- Un sharded cluster usa siempre y exactamente 3 config servers.
- Los config servers operan en modo lectura principalmente.
- Importante hacer backups de los config servers! Si los perdemos el clúster no funcionará.
- Los query routers cachean la información de los config servers durante el arranque.



Sharded clúster



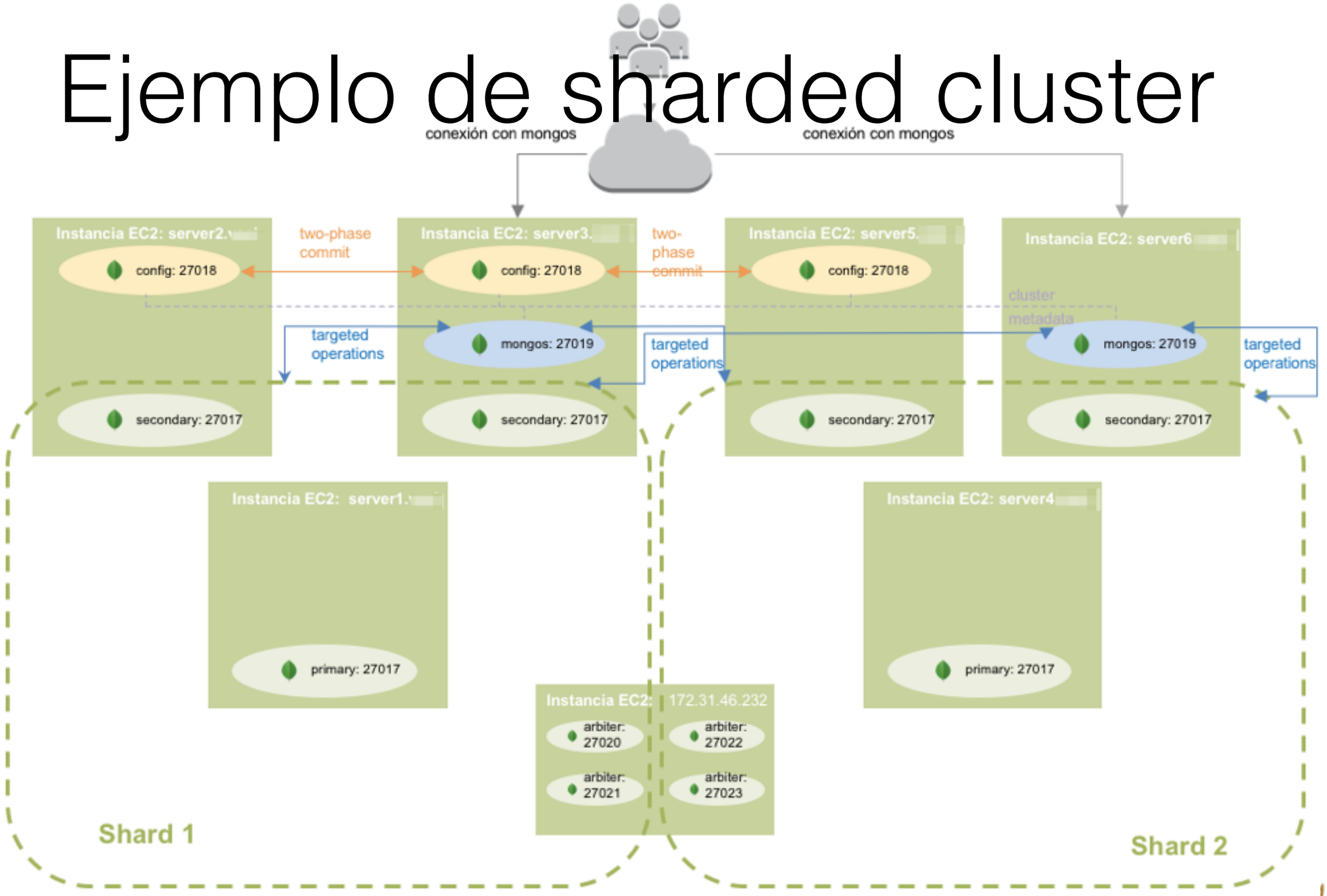
Recapitulando sharded clusters

- Un clúster de MongoDB lo forman varios shards.
- Cada shard almacena parte del dataset.
- Cada shard es el fondo un replica set.
- Un clúster tendrá varios query routers.
- Las aplicaciones clientes llaman a los query routers. Un cliente solo habla con un querrá router.
- Los query routers se reparten la carga de peticiones al clúster.
- Los query routers redirigen operaciones a los shards.
- La información de en qué shard residen los datos la alojan los config servers. Y son 3 en producción.

Recapitulando sharded clusters

- Necesitaremos clusterizar…
  - Cuando el dataset excede la capacidad de un nodo.
  - Cuando el working set más los índices exceden la RAM disponible.
  - Cuando se detecta un cuello de botella en operaciones de escritura.
  - Cuando necesitamos distribución geográfica de los datos.

Ejemplo de sharded cluster



Particionando los datos en shards

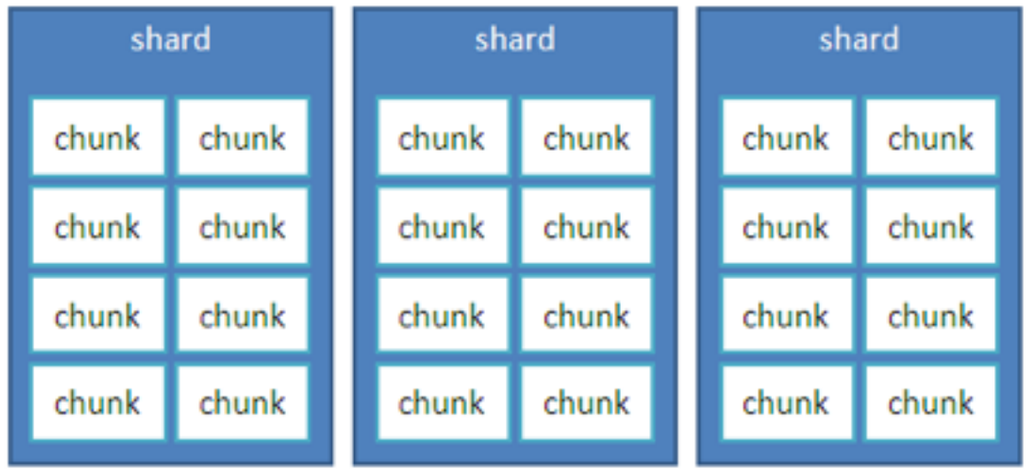
- MongoDB particiona a nivel de colección.

- Se requiere una clave de particionamiento o *shard key*.

- Se puede usar como shard key un campo indexado o una composición de campos indexados.

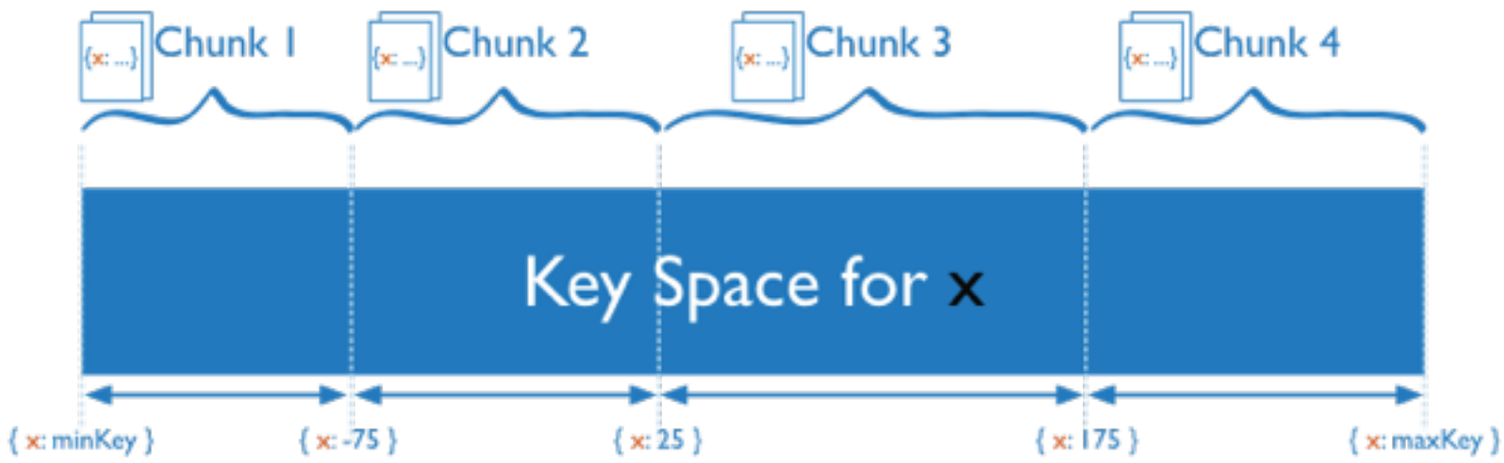
- Los campos para la shard key deberán existir en todos los documentos de la colección.

- MongoDB particiona el espacio de valores de la shard key en trozos o *chunks*.



Dos tipos de particionamiento para escoger

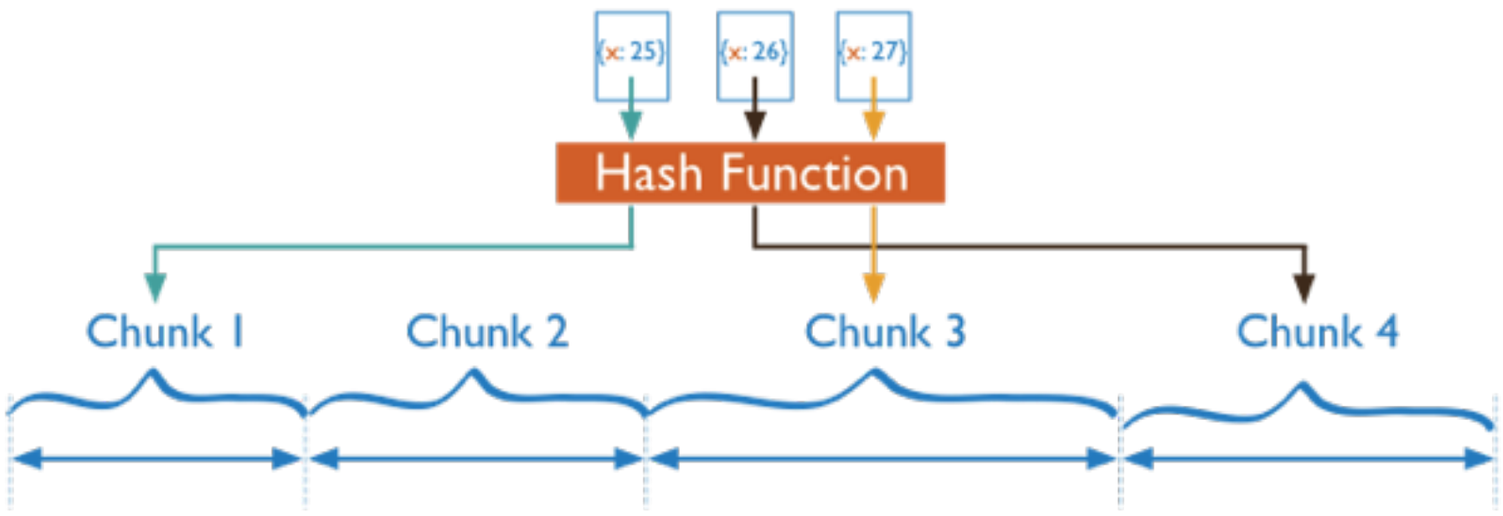
- Range based sharding:
  - Se divide en chunks a partir del valor de la clave de particionamiento. Cada chunk contendrá un rango de valores.



Dos tipos de particionamiento para escoger

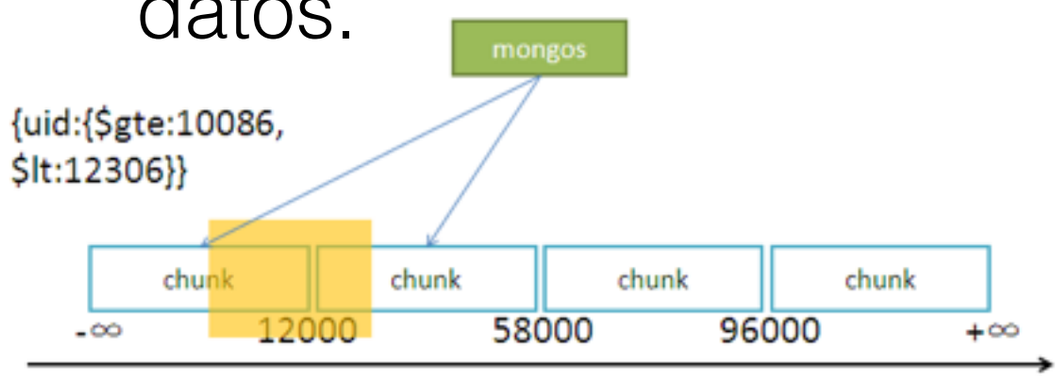
- Hash based sharding

  - Se aplica una función hash al valor de la clave de particionamiento para determinar a qué chunk pertenece el documento.



Dos tipos de particionamiento para escoger

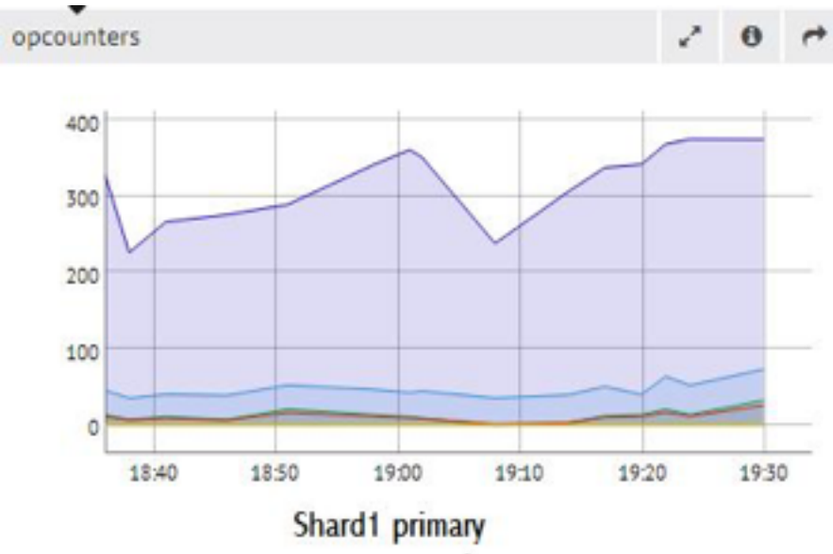
- Range based
  - Ideal para consultas basadas en rangos de valores.
  - Susceptible de distribución desbalanceada de los datos.

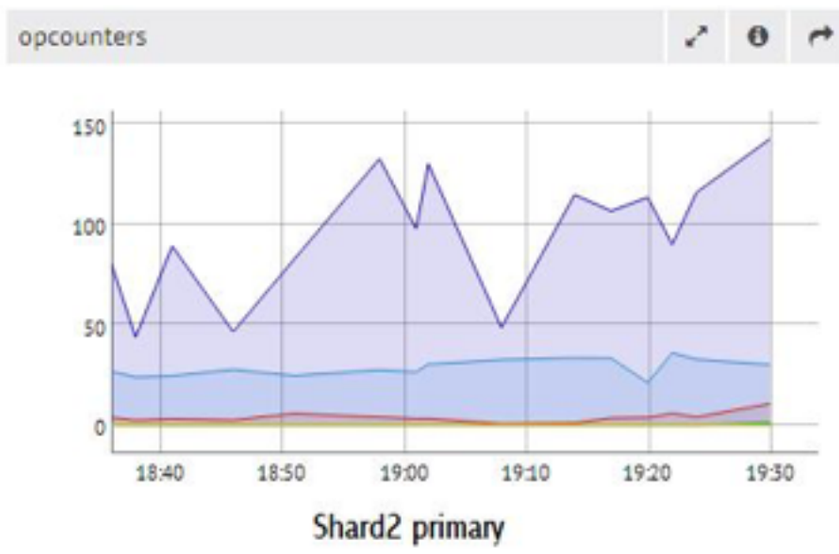


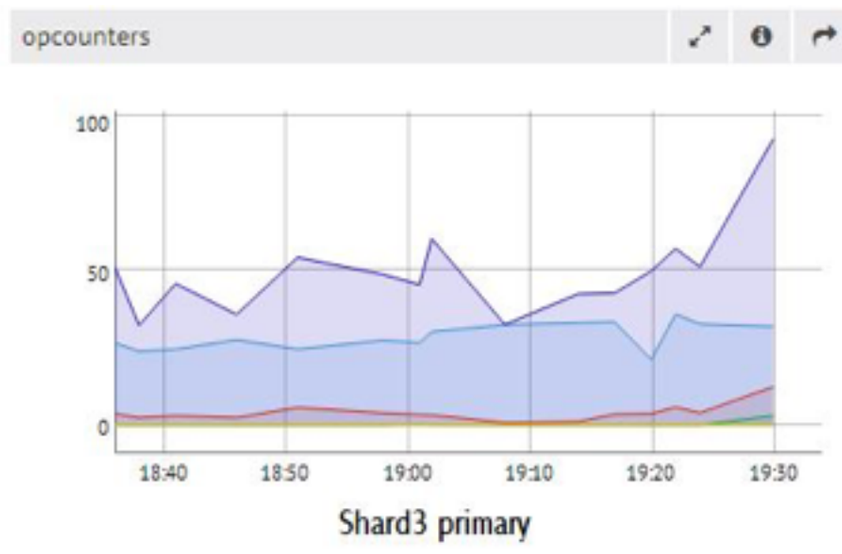
- Hash based:
  - Distribuye más aleatoriamente los datos.
  - Penaliza las consultas basadas en rangos.

Ejemplo de clúster desbalanceado

- Clúster de 3 shards







- El primer shard procesa 3 veces más operaciones que el segundo…
- …y 6 veces más que el tercero.

Monitorización de un clúster

- `db.collection.getShardDistribution()` permite ver cómo se distribuye una colección entre los shards y detectar desbalanceamiento.
- `db.stats()` nos da información por cada shard.
- `sh.status()` nos da la distribución de chunks en el cluster.
- `mongostat --discover` sobre un mongos nos da métricas en tiempo real de cada máquina del clúster.

Planificación del tamaño de un clúster

- Buena argumentación aunque antigua de capacity planning: https://lotusinmud.wordpress.com/2013/08/24/mongodb-capacity-planning/

- http://es.slideshare.net/mongodb/capacity-planning-for-your-growing-mongodb-cluster

- https://www.mongodb.com/blog/post/capacity-planning-and-hardware-provisioning-mongodb-ten-minutes habla de la PoC de un banco español.

Herramientas para administración y operación

Herramientas

- Múltiples herramientas: https://docs.mongodb.org/ecosystem/tools/administration-interfaces/
- MongoDB Ops Manager:
  - https://www.mongodb.com/products/ops-manager
  - Demo: https://youtu.be/AkqJ98a6v5Y
- Herramientas del SO: `iostat`, `vmstat`, `netstat`

Parámetros a monitorizar

- RAM (`memory`)
- Número de operaciones en el tiempo (`opcounters`)
- Paginación a disco (`page faults`)
- Tiempo de escritura en disco (`background flush avg`)
- Ejemplo de uso: si aumenta la paginación a la vez que caen las operaciones, se necesitará más RAM

Seguridad en MongoDB

Seguridad en MongoDB

Welcome, hackers!

Up to 40,000 MongoDB databases left unsecured online

February 12, 2015 Natali Vlatko

#databases #mongodb #nosql #security

reddit 62 LinkedIn Facebook Google+

MongoDB are back in the spotlight for all the wrong reasons. Up to 40,000 MongoDB databases have been found open and unsecured by a team of students, showing us how potential cybercriminals could access sensitive and important data.

Warning image via Shutterstock

A major security issue has been unveiled at MongoDB with 40,000 databases discovered unsecured on the internet. The discovery was made by university students in Germany who were able to gain easy access by running a port scan on the internet to find openly accessible databases.

https://jaxenter.com/mongodb-databases-left-unsecured-internet-114500.html

Checklist de seguridad en MongoDB

- Control de acceso y autenticación a los servidores
- Control de acceso a los datos basado en roles
    - Crear primero un súper-usuario administrador
    - Crear nuevos usuarios y asignar permisos siguiendo PoLP
- Encriptar todas las comunicaciones con TLS/SSL
    - No olvidar las comunicaciones entre `mongod` y `mongos`!
- Restringir el acceso a las máquinas donde corren los procesos de MongoDB
- Encriptación a nivel de fichero: **`https://youtu.be/9Gy999RO7Z0`**
- No ejecutar `mongod` y demás procesos con privilegios de administrador del sistema!!!

opmgr.sademo.mongodb.com:8080/host/list/55f6847ee4b0397e08ae97b6#roles

Ops Manager 1.8.0.276 GROUP ProductionDevices

Edinburgh Ops

Deployment
Activity
Backup
Administration

Deployment
Host Mappings
Security
Version Manager
MongoDB Users
MongoDB Roles
MongoDB Logs
Pings

+ ADD ROLE

Add Role

Identifier POCDB appuser

Roles are identified by a combination of their name and the database to which they belong.

Inherits From

A list of roleName@dbName pairs; e.g., readWrite@userDB

Privilege Actions by Resource

Resource

POCDB | Collection Name (leave blank for any)

Available Privileges

- Database Management Actions
  - changeCustomData
  - changeOwnCustomData
  - changeOwnPassword
  - changePassword
  - createCollection
  - createIndex
  - createRole
  - createUser
  - dropCollection
  - dropRole
  - dropUser
  - emptycapped
  - enableProfiler
  - grantRole
  - killCursors
  - revokeRole
  - viewRole
  - viewUser
- Query and Write Actions
  - find
  - insert
  - remove
  - update
- Server Administration Actions
  - collMod
  - compact
  - convertToCapped
  - dropDatabase
  - dropIndex
  - reIndex
  - renameCollectionSameDB
  - repairDatabase
- Deployment Management Actions
  - planCacheRead
  - planCacheWrite
  - storageDetails
- Sharding Actions
  - getShardVersion
- Diagnostic Actions
  - collStats
  - dbHash
  - dbStats
  - indexStats
  - validate

CANCEL ADD PRIVILEGES

CANCEL ADD ROLE

https://youtu.be/AkqJ98a6v5Y?t=7m50s

MongoDB tips and tricks

Tips & tricks

- Cuellos de botella habituales en MongoDB:
  - Esquema desalineado con el patrón de acceso a los datos
  - Indexado pobre o inexistente; índices innecesarios
  - Discos lentos y/o pocos IOPS para la carga real
  - Los índices no caben en la RAM (paginación)

Tips & tricks

- Fundamental: Un buen esquema + una buena estrategia de indexación + hardware adecuado
- DevOps!!!! NoSQL != NoDBA
- Monitorizar y monitorizar para anticipar la necesidad de crecimiento: no se puede optimizar lo que no se mide.
- El sharding no soluciona necesariamente un problema de lentitud en MongoDB
- Acudiremos al sharding cuando detectamos cuellos de botella en el hardware y el escalado vertical es demasiado oneroso. Si no, debería bastar con un único RS.

Tips and tricks

- Un mito: "*para que el rendimiento de MongoDB sea espectacular, la base de datos deberá caber en RAM*"

- Separar el journal y los datos en volúmenes diferentes

- Las réplicas del RS no eliminan la necesidad de backups!!!

- Las réplicas del RS tienen que tener una buena ventana temporal de visibilidad del *oplog* del primario para que les de tiempo a sincronizar

Tips and tricks

- Securizar los datos usando el “Principio de mínimo privilegio”

- Bloquear accesos fuera de las IPs en las que corren los clientes (firewall, `iptables`)

- Perfilar periódicamente en producción las operaciones que MongoDB ejecuta usando la información de la colección `system.profile` y `explain()` en tiempo de desarrollo

Tips and tricks

- https://docs.mongodb.org/manual/data-modeling/
- http://blog.mongodb.org/post/38467892360/mongodb-schema-design-insights-and-tradeoffs-from
- http://www.slideshare.net/jrosoff/mongodb-advanced-schema-design-inboxes

Tips and tricks

- Evitar modelar documentos que crecen con el tiempo
- Evitar sobreescribir un documento solo por que necesitemos actualizar parte de él. Usar modificadores: `$set`, `$unset`, `$inc`...
- Tratar siempre del mismo modo los tipos de datos. Escribir un valor `(int)0` en un campo con `(float)0.0` puede llevar a mover el documento en la colección
- Preescribir campos del documento aunque no se conozca su valor (*placeholders*), pero no usar `null` como valor
- No abusar de la longitud de los nombres de los campos del documento: 2 caracteres adicionales * 100M de documentos son Mbs de espacio adicional en la colección

Tips and tricks

- Aprovecharse de `_id` y su índice por defecto para almacenar información del documento

- Maximizar el uso de *covered queries* para minimizar lecturas de documentos

Escalabilidad en MongoDB: controversia

"MongoDB has always scaled—there are many, many examples, but in prior releases, doing so required a level of expertise that not everyone has. With MongoDB 3.0 it's a lot easier to scale your system."

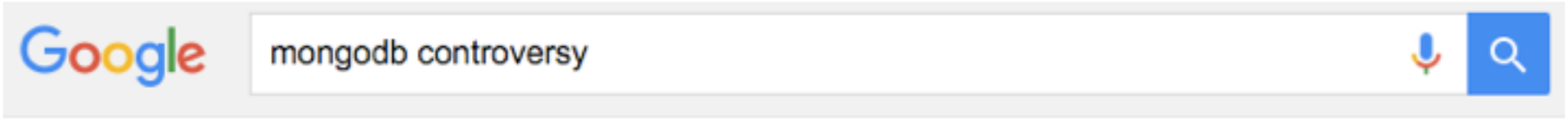
– Kelly Stirman, MongoDB's director of products

Controversias MongoDB

By installing more tollbooths, mostly.

440
SHARES

"MongoDB doesn't scale." The nagging criticism that the open-source database can't handle larger data workloads, propelled by a viral video roughly five years ago, continues to haunt it to this day.



Aproximadamente 27.600 resultados (0,46 segundos)

Does everyone hate MongoDB? - Server Density Blog
https://blog.serverdensity.com/does-everyone-hate-... ▾ Traducir esta página
24 sept. 2012 - For a guaranteed surge of traffic and to hit the Hacker News homepage, all you need to do is write about why you hate **MongoDB** and/or ...

MongoDB Inc. - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**MongoDB**_Inc. ▾ Traducir esta página
Ir a CIA backing and **controversy** - [edit]. **MongoDB** Inc., then known as 10gen, has received funding from the U.S. Government through the ...

Real world use Cases of MongoDB | Edureka
www.edureka.co/.../real-world-use-cases-of-**mongod**... ▾ Traducir esta página
22 ene. 2014 - Adhar is an excellent example of real world use cases of **MongoDB**. In recent times, there has been some **controversy** revolving around CIA's ...

How The MongoDB Database Learned To Scale - ReadWrite
readwrite.com/.../**mongodb**-scale-document-database... ▾ Traducir esta página
9 feb. 2015 - With **MongoDB** 3.0 it's a lot easier to scale your system." ... Needless to say, **controversy** goes with the territory. "Every piece of the technology ...

Inside India's Aadhar, The World's Biggest Biometrics ...
techcrunch.com/.../inside-indias-aadhar-the-worlds-b... ▾ Traducir esta página
6 dic. 2013 - **MongoDB**, a NoSQL database startup, last year raised funding from the ... Some of the reports have linked the **controversy** with **MongoDB**.

Don't use MongoDB | Hacker News

Controversias MongoDB

Controversias MongoDB

Comparative Benchmarks: MongoDB vs. Couchbase vs. Cassandra

Independent evaluators United Software Associates demonstrate that MongoDB overwhelmingly outperforms key value stores Couchbase and Cassandra. The results show that MongoDB smokes NoSQL vendors in terms of both throughput and latency across a number of configurations.

Download this comprehensive report to get access to:

- Performance test results comparing Couchbase 3.0.2, Cassandra 2.12 and MongoDB 3.0.1 with WiredTiger
- United Software Associates' benchmarking methodology, including the different configuration options tested
- Detailed discussion on the trade-offs between performance and durability

Research published in March 2015.

Controversias MongoDB

Couchbase Why NoSQL? Products Use Cases Training Developers Resources Subscr DOWNLOAD

Search

BACK TO BLOGS

What does MongoDB have against transparency?

Shane Johnson of Couchbase • Published June 2, 2015

MongoDB published another benchmark performed by United Software Associates.

Benchmarks are a useful tool to evaluate database performance. But to be useful, they must be transparent and repeatable. If they fail to meet these standards, the results are questionable.

In recent benchmarks, Couchbase and MongoDB took two different approaches. Couchbase clearly documented the full configuration *and* included the results of

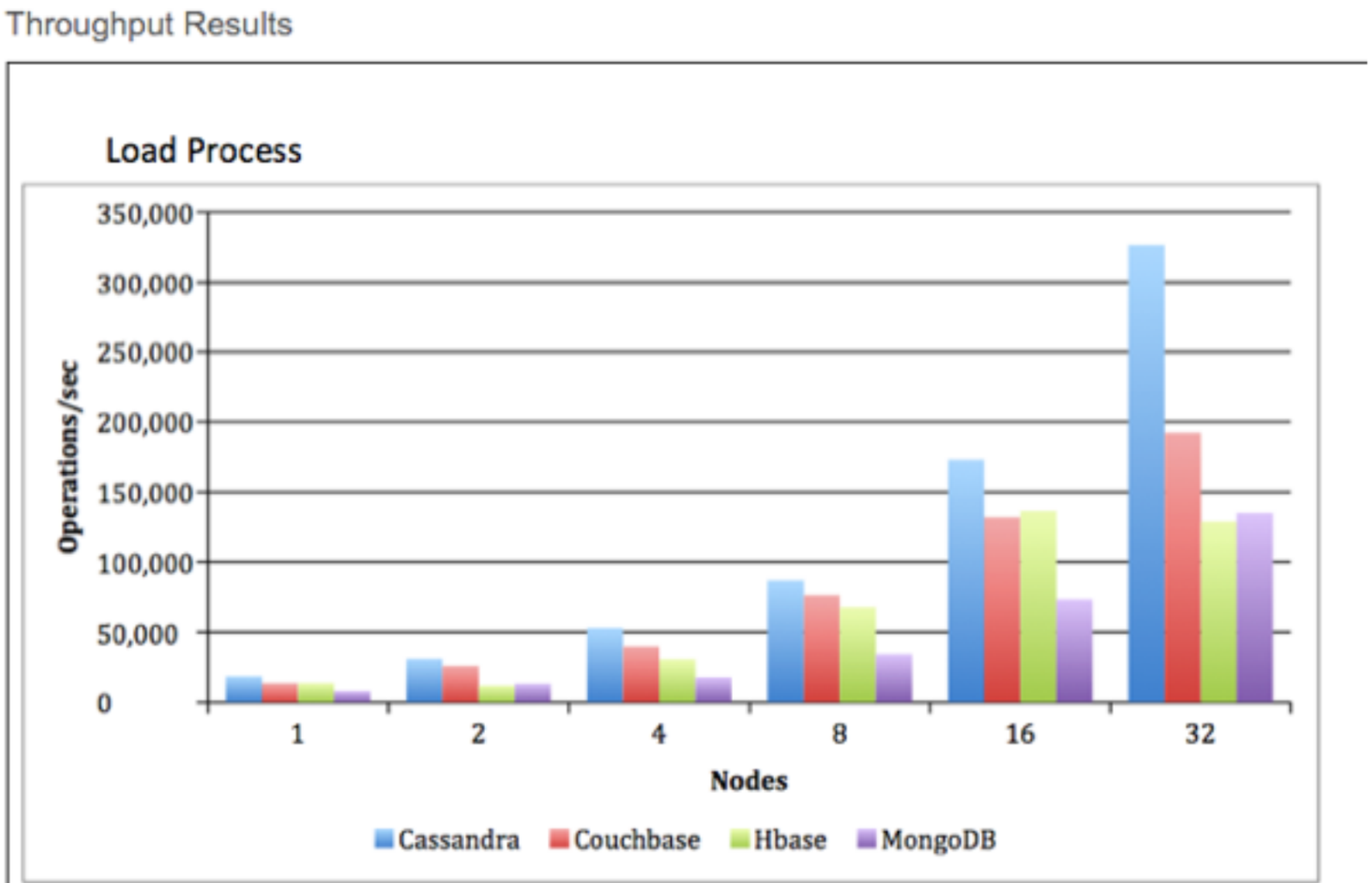
Shane Johnson

Sr. Product Marketing Manager
Couchbase

Shane K Johnson is a former developer,

Controversias MongoDB

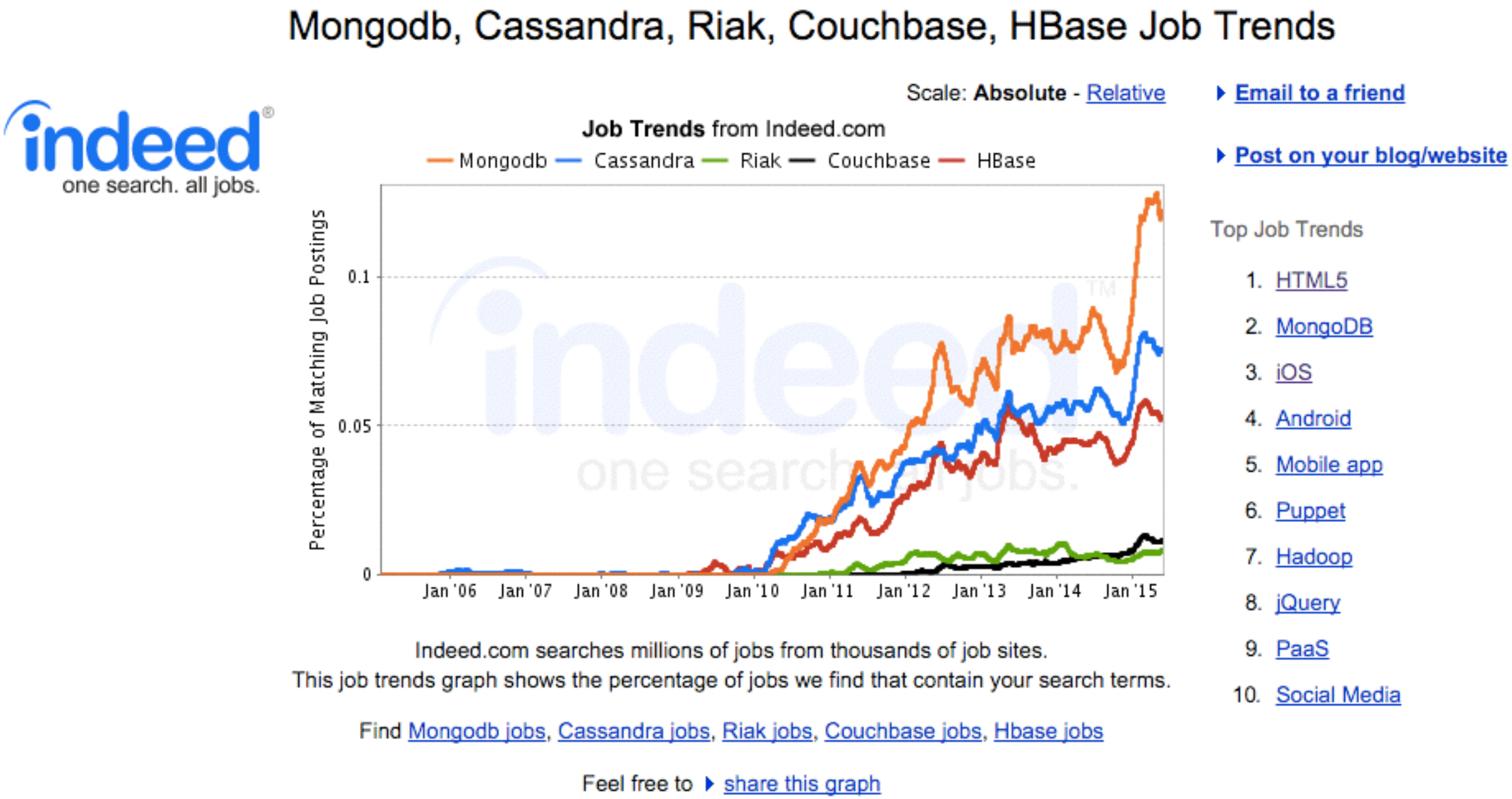
PLANET CASSANDRA
A DATASTAX COMMUNITY SERVICE



Nodes	Cassandra	Couchbase	HBase	MongoDB
1	18,683.43	13,761.12	13,884.91	8,368.44
2	31,144.24	26,140.82	12,258.10	13,462.51
4	53,067.62	40,063.34	30,842.32	18,038.49
8	86,924.94	76,504.40	67,966.56	34,305.30
16	173,001.20	131,887.99	136,419.37	73,335.62
32	326,427.07	192,204.94	128,918.77	134,968.87

http://www.planetcassandra.org/nosql-performance-benchmarks/

Controversias MongoDB



Referencia

Referencia

- Documentación
- MongoDB Performance Best Practices
- MongoDB Operations Best Practices
- Guía de securización

Lecturas recomendadas

- Please stop calling databases CP or AP
- On Distributed Consistency
- Achieving scale with MongoDB
- Call me maybe: MongoDB stale reads

Anexo: operaciones de escritura

Operaciones de escritura

- Operaciones de escritura:
  - Adición de documentos: `db.collection.insert()`
  - Actualización de documento: `db.collection.update()`
  - Actualización de varios documentos: `db.collection.update({…},{…}, {multi:true})`
  - Actualización con inserción: `db.collection.update({…},{…}, {upsert:true})`
  - Borrado de documentos: `db.collection.remove()`

Atomicidad o aislamiento de operaciones de escritura

- Las operaciones de escritura son atómicas al nivel de un único documento.

- Si la operación de escritura involucra varios documentos, la operación total no es atómica.

Garantías de escritura

- MongoDB denomina “write concern” a la garantía de escritura de un dato.
- Existen diferentes niveles de “write concern” disponibles:

Write concern	Descripción
Unacknowledged	MongoDB no confirma la recepción de operaciones de escritura
Acknowledged	**Comportamiento por defecto.** `mongod` confirma que ha recibido la operación y la ha aplicado en memoria. No significa que haya aplicado la operación en disco.
Journaled	Confirma que ha aplicado la operación en el *journal*.
Replica Acknowledged	En caso de usar un RS, confirma que la operación se ha propagado a al menos un nodo secundario.