# Fundamentos MongoDB Enterprise 4.4

Instructor: Carlos Carreño
Email: ccarrenovi@Gmail.com


# Unidad 9 Monitoring and Tunning

## Fundamentos de MongoDB Enterprise 4.4

# Monitoring Tools

- Las herramientas principales para el monitoreo en MongoDB son estas dos:
  - **mongostat**
  - **mongotop**

# Tool mongostat

- La utilidad **mongostat** proporciona una visión general rápida del estado de una instancia mongod o mongos actualmente en ejecución. mongostat es funcionalmente similar a la utilidad del sistema de archivos UNIX / Linux vmstat, pero proporciona datos sobre instancias mongod y mongos.

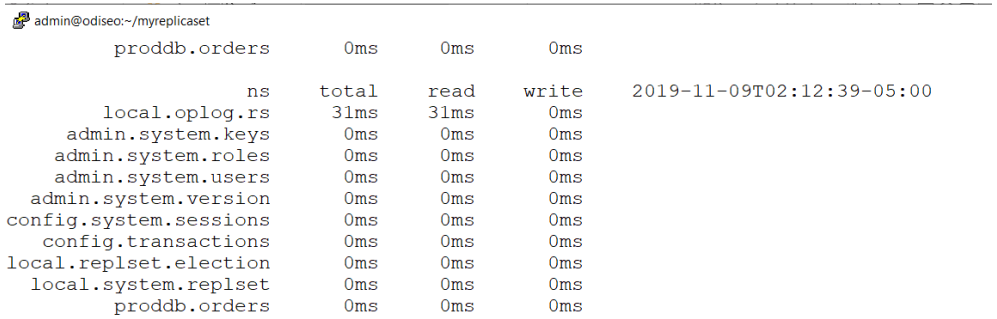
***mongostat*** --port 27020 --username adminClusterUser --password mongodb123 --authenticationDatabase admin

```
[admin@odiseo myreplicaset]$ mongostat --port 27020 --username adminClusterUser --password mongodb123 --authenticationDatabase adm
in
insert query update delete getmore command dirty used flushes vsize  res qrw arw net_in net_out conn          set repl
    time
    *0    *0     *0     *0       0     2|0  0.0% 0.0%       0 1.90G 121M 0|0 1|0   742b   37.0k   17 myreplicaset  PRI Nov  9 02:0
9:06.976
    *0    *0     *0     *0       0     2|0  0.0% 0.0%       0 1.90G 121M 0|0 1|0   542b   36.5k   17 myreplicaset  PRI Nov  9 02:0
9:07.974
    *0    *0     *0     *0       0     2|0  0.0% 0.0%       0 1.90G 121M 0|0 1|0   596b   37.1k   17 myreplicaset  PRI Nov  9 02:0
9:08.977
^C2019-11-09T02:09:09.104-0500  signal 'interrupt' received; forcefully terminating
```

# Tool mongotop

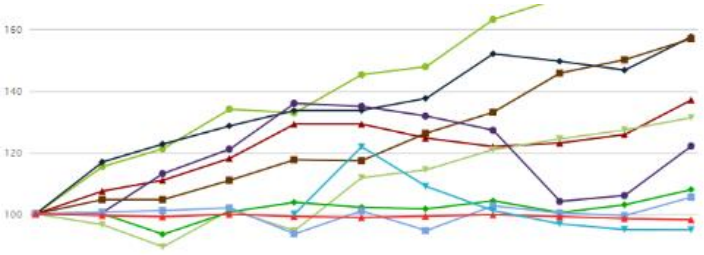
- **mongotop** proporciona un método para rastrear la cantidad de tiempo que una instancia de MongoDB mongod pasa leyendo y escribiendo datos. **mongotop** proporciona estadísticas en un nivel por colección. Por defecto, mongotop devuelve valores cada segundo.

mongotop --port 27020 --username adminClusterUser --password mongodb123 --authenticationDatabase admin 10



# Collection stat

- El método stats, en cualquier colección muestra las estadísticas de la colección
- Ejemplo: ***db.people.stats()***



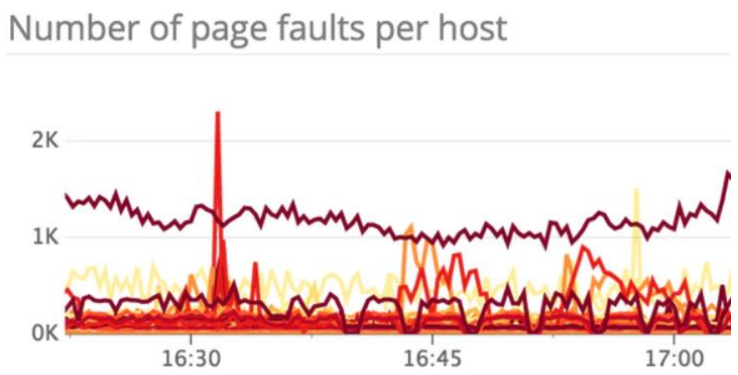
# Database stats()

- Usa el método ***db.stats()*** para obtener estadísticas de la base de datos

```
MongoDB Enterprise myreplicaset:PRIMARY> db.stats();
{
        "db" : "test",
        "collections" : 0,
        "views" : 0,
        "objects" : 0,
        "avgObjSize" : 0,
        "dataSize" : 0,
        "storageSize" : 0,
        "numExtents" : 0,
        "indexes" : 0,
        "indexSize" : 0,
        "scaleFactor" : 1,
        "fileSize" : 0,
        "fsUsedSize" : 0,
        "fsTotalSize" : 0,
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1573284039, 1),
                "signature" : {
                        "hash" : BinData(0,"stT5fbg14C+THY+4aPoEA1W1zVU="),
                        "keyId" : NumberLong("6757100127845875714")
                }
        },
        "operationTime" : Timestamp(1573284039, 1)
}
MongoDB Enterprise myreplicaset:PRIMARY>
```

# Tuning MongoDB

- MongoDB es una base de datos NoSQL extremadamente rápida pero como todo motor necesita monitorearse y afinar para mejorar el desempeño en determinadas configuraciones.
- Para el monitoreo y afinamiento nos enfocaremos en lo siguiente:
  - Performance of locking
  - Memory usage
  - Connection handling
  - Issues with replica sets
  - Collection Index



# Analizando el Bloqueo

- Cuando ocurre un bloqueo, ninguna otra operación puede leer o modificar los datos hasta que la operación que inició el bloqueo haya finalizado. Esto evita conflictos. Pero también puede degradar severamente el rendimiento de la base de datos.

# Monitoreando las métricas de bloqueo

- MongoDB proporciona algunas métricas útiles para ayudar a determinar si el bloqueo está afectando el rendimiento de la base de datos.

**db.serverStatus().globalLock**

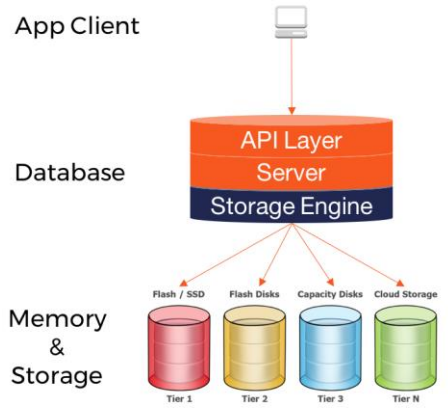
**db.serverStatus().locks**

# Metricas globales de bloqueo

- ***globalLock.currentQueue.total***: este número puede indicar un posible problema de concurrencia si es constantemente alto. Esto puede suceder si muchas solicitudes están esperando que se libere un bloqueo.
- ***globalLock.totalTime***: si es superior al tiempo de actividad total de la base de datos, la base de datos ha estado en estado de bloqueo durante demasiado tiempo.

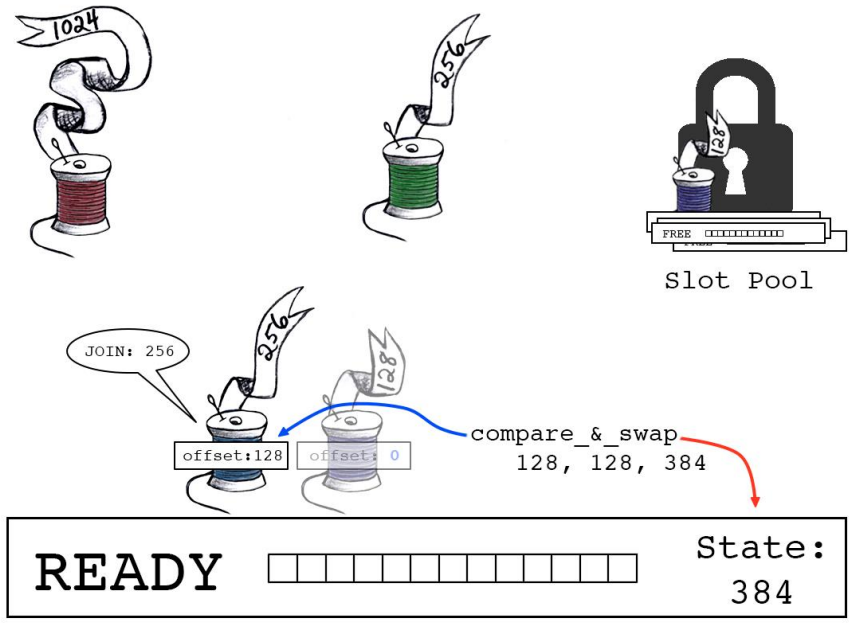
# Storage Engine

- El motor de almacenamiento es el componente de la base de datos que ***se encarga de administrar cómo se almacenan los datos***, tanto en la memoria como en el disco.
- MongoDB admite múltiples motores de almacenamiento, ya que diferentes motores funcionan mejor para cargas de trabajo específicas. Elegir el motor de almacenamiento adecuado para su caso de uso puede afectar significativamente el rendimiento de sus aplicaciones.



# Storage Engine y el Bloqueo

- ¿La base de datos se bloquea frecuentemente en las consultas? Esto podría indicar problemas con el diseño del esquema, la estructura de consulta o la arquitectura del sistema. El storage Engine actual de mongoDB es WiredTiger



# Examinando el uso de la memoria

- Podemos usar las métricas en la sección de memoria del documento **serverStatus** para comprender cómo MongoDB está usando la memoria del sistema.

```
> db.serverStatus().mem
{ "bits" : 64, "resident" : 14, "virtual" : 5484, "supported" : true }
```

# Métricas de Uso de Memoria

- mem.resident: aproximadamente equivalente a la cantidad de RAM en megabytes que utiliza el proceso de la base de datos

# Interpretando Métricas de Uso de Memoria

- Para ver si hemos excedido la capacidad de nuestro sistema, podemos comparar el valor de mem.resident con la cantidad de memoria del sistema. Si mem.resident excede el valor de la memoria del sistema y hay una gran cantidad de datos sin asignar en el disco, lo más probable es que hayamos excedido la capacidad del sistema.

# Tune the WiredTiger cache

- El motor de almacenamiento ***WiredTiger es una mejora significativa sobre MMAPv1*** (antiguo motor de versiones anteriores a 4.0) en rendimiento y concurrencia.
- MongoDB reservará el 50 por ciento de la memoria disponible para el caché de datos WiredTiger. El tamaño de este caché es importante para garantizar que WiredTiger funcione adecuadamente. Merece la pena echarle un vistazo para ver si debe modificarlo de forma predeterminada.

**db.serverStatus().wiredTiger.cache**

# Examinando las Métricas WiredTiger cache

- wiredTiger.cache.maximum bytes configurados: este es el tamaño máximo de caché.
- wiredTiger.cache.bytes actualmente en el caché: este es el tamaño de los datos actualmente en el caché. Esto no debe ser mayor que los bytes máximos configurados.
- wiredTiger.cache.tracked bytes sucios en el caché: este es el tamaño de los datos sucios en el caché. Este valor debe ser menor que los bytes actualmente en el valor de caché.

# Interpretando Metricas de WiredTiger cache

- Al observar estos valores, podemos determinar si necesitamos aumentar el tamaño de la memoria caché para nuestra instancia. Además, podemos ver el valor de wiredTiger.cache.bytes leído en caché para aplicaciones con mucha lectura. Si este valor es constantemente alto, aumentar el tamaño de la memoria caché puede mejorar el rendimiento general de lectura.

# storage.wiredTiger Options

- Storage.wiredTiger options del archivo de configuracion

```
storage:
   wiredTiger:
      engineConfig:
         cacheSizeGB: <number>
         journalCompressor: <string>
         directoryForIndexes: <boolean>
         maxCacheOverflowFileSizeGB: <number>
      collectionConfig:
         blockCompressor: <string>
      indexConfig:
         prefixCompression: <boolean>
```

**Nota:** Por defecto el sistema usara (50%*Total RAM – 1) GB

# Monitor the number of current connections

- A menos que los límites del sistema lo limiten, MongoDB no tiene límites en las conexiones entrantes. Pero hay una trampa. En algunos casos, una gran cantidad de conexiones entre la aplicación y la base de datos puede abrumar a la base de datos. Esto limitará su capacidad para manejar conexiones adicionales.

```
db.serverStatus().connections;
```

```
{ "current" : 1, "available" : 999999, "totalCreated" : 1, "active" : 1 }
```

# Monitor the number of current connections

- connections.current: el número total de clientes actuales conectados a esta instancia
- connections.available: el número total de conexiones no utilizadas disponibles para los clientes para esta instancia

# Tuning  Connection

- Si hay problemas de conexión, hay un par de estrategias para resolverlos:
  - Primero, verifique si la aplicación es de lectura pesada. Si es así, aumente el tamaño del conjunto de réplicas y distribuya las operaciones de lectura a los miembros secundarios del conjunto.
  - Si esta aplicación es pesada para escritura, use sharding dentro de un Sharded clúster para distribuir la carga.
- Los errores relacionados con la aplicación o el controlador también pueden causar problemas de conexión. Por ejemplo, una conexión puede eliminarse de manera incorrecta o puede abrirse cuando no sea necesario, si hay un error en el controlador o la aplicación. Verá esto si el número de conexiones es alto, pero no hay una carga de trabajo correspondiente.

# Watch replication performance

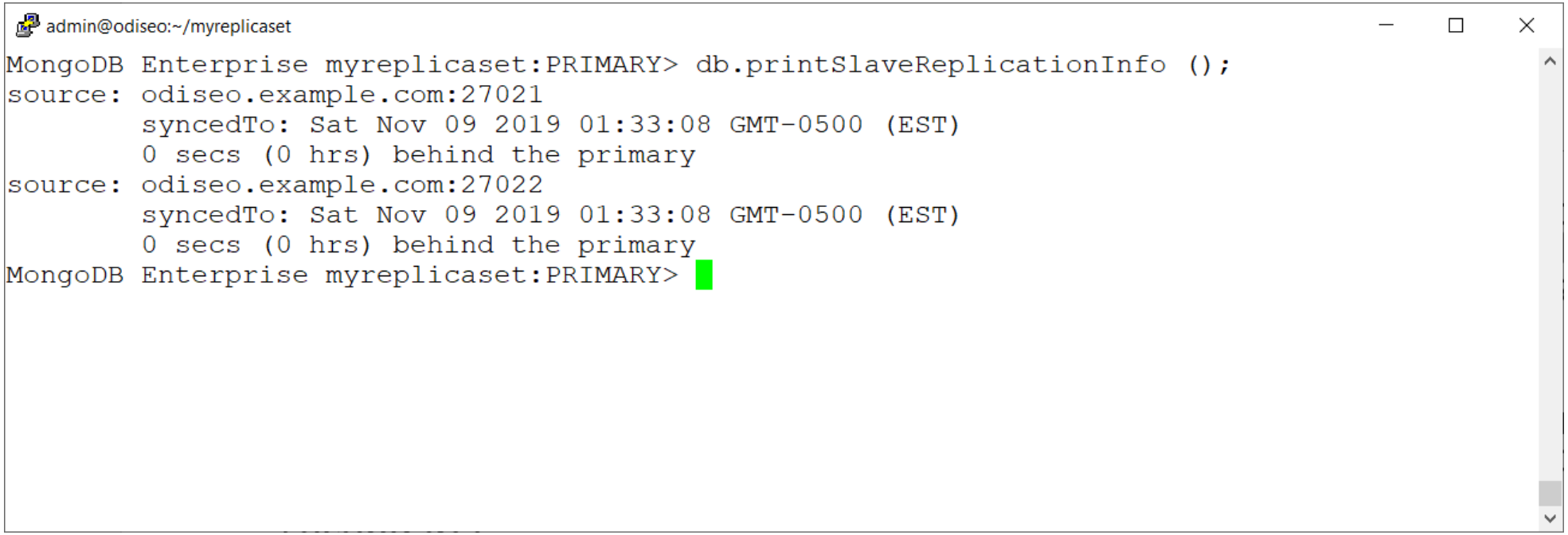
- La replicación es la ***propagación de datos de un nodo a otro***. Es clave para que MongoDB pueda enfrentar los desafíos de disponibilidad. A medida que los datos cambian, se propagan desde el nodo primario a los nodos secundarios. Los conjuntos de replicación manejan esta replicación.
- Estrategias:
  - Monitorear el retraso de replicación
  - Monitorear el estado de la replicación

# Monitorear el retraso de replicación

- En un mundo perfecto, los datos se replicarían entre nodos casi instantáneamente. Pero no vivimos en un mundo perfecto. A veces, los datos no se replican tan rápido como nos gustaría. Y, dependiendo del tiempo que tome una replicación, **corremos el riesgo de que los datos no estén sincronizados.**
- Este es un problema particularmente espinoso si el retraso entre un nodo primario y secundario es alto y el secundario se convierte en el primario. Debido a que la replicación no ocurrió lo suficientemente rápido, los datos se perderán cuando la primaria recién elegida se replique en la nueva secundaria.
- Podemos usar el comando ***db.printSlaveReplicationInfo ()*** o el comando ***rs.printSlaveReplicationInfo ()*** para ver el estado de un conjunto de réplicas desde la perspectiva del miembro secundario del conjunto.

# Monitorear el retraso de replicación

- Ejemplo: El valor en segundos de **"… behind the primary"** debe ser lo mas pequeño posible.



# Monitor replication state

- En condiciones normales, ***el estado del nodo asignado como "primary" rara vez debería cambiar***. Si se produce un cambio de rol, es decir, un nodo secundario se elige primario, queremos saber de inmediato. La elección de una nueva primaria generalmente ocurre a la perfección. Aún así, debe comprender qué causó el cambio de estado. Esto se debe a que podría deberse a una falla de la red o del hardware. Además, no es normal que los nodos cambien entre primario y secundario.

# Índices en MongoDB

- Los índices en **MongoDB** se generar en forma de *Árbol-B* o *B-Tree*. Es decir, que los datos se guardan en forma de árbol, pero manteniendo los nodos balanceados. Esto incrementa la velocidad a la hora de buscar y también a la hora de devolver resultados ya ordenados.

# Índices simples o de un solo campo

- Estos índices se aplican a un solo campo de nuestra colección.

```
db.users.ensureIndex( { "user_id" : 1 } )
```

# Índices compuestos

- En este caso el índice se generará sobre varios campos

```
db.users.ensureIndex( { "user_name" : 1, "age":-1 } )
```

"Antonio", 35
"Antonio", 18
"María", 56
"María", 30
"María", 21
"Pedro", 19
"Unai", 34
"Unai", 27

# Índices Únicos

- Los índices simples y múltiples, pueden estar obligados a contener valores únicos. Esto lo conseguimos añadiendo el parámetro *unique* a la hora de crearlos.

```
db.users.ensureIndex( { "user_id" : 1 }, {"unique":true} )
```

# Índices sparse

- Los índices que hemos mencionado antes, incluyen todos los documentos. También los documentos que no contengan el campo indexado. **MongoDB** no obliga a que usemos un esquema, así que no hay obligatoriedad en que los campos existan en el documento.

```
db.users.ensureIndex( { "user_name" : 1 }, {"sparse":true} )
```

# Indexación de subdocumentos

- No hemos hablado todavía de la posibilidad de indexar subdocumentos. Supongamos un documento con la siguiente estructura:

```
{
   "user_id": "DJK2312212",
   "info":
   {
      "user_name": "Pedro",
      "age": 22
   }
}
```

```
db.users.ensureIndex( { "info.user_name" : 1 } )
```

# Consultas totalmente cubiertas por los índices

- Aunque tengamos varios índices creados, no todas las consultas van a ser igual de eficientes. Las consultas más rápidas serán las denominadas **consultas totalmente cubiertas**. Son aquellas consultas cuyos campos consultados y devueltos están incluidas en el índice.

```
db.users.ensureIndex( { "user_name" : 1, "age":1 } )
```

En MongoDB 4.4

```
db.people.createIndex({name: 1});
```

# Analizando planes de ejecución en MongoDB

- Las bases de datos relacionales suelen poner a nuestra disposición alguna utilidad para consultar los planes de ejecución que utilizará el motor de consultas al realizar una consulta. Un plan de ejecución presenta una descripción, con los pasos que realizará el motor para buscar los datos. Es algo muy útil, ya que nos permite analizar si nuestros índices están funcionando correctamente, o si podemos añadir alguno más para incrementar la velocidad de las consultas.

# Mirando el Plan de Ejecucion

- **MongoDB**, como no podía ser menos, tiene una utilidad similar que podemos explotar con en el operador *$explain*.

```
db.colección.find().explain()
```

- Esta consulta nos devolverá un documento JSON con muchísima información útil. Entre ella el número de documentos procesados para devolver el resultado, si la consulta ha utilizado solo los índices, si ha tenido que ordenar los resultados en memoria, el tiempo que ha tardado la consulta en ejecutarse etc.

# Demostración

use devmongodb**;**

db.people.find**().limit(3);**

db.people.createIndex**({name: 1});**

db.people.createIndex**({name: 1, age: -1});**

db.people.getIndexes**();**

db.people.dropIndex**({name: 1});**

//Compatible con versiones previas
db.people.ensureIndex**({name: 1});**

// Consultas los documentos en people que al menos tenga una letra a en el atributo name.
// Mostrar el plan de ejecucion
db.people.find**({name: /a/}).explain();**

# Practica

- Practica 9 Monitoreo y Afinamiento de MongoDB

# Referencias


- https://stackify.com/mongodb-performance-tuning/
- https://docs.mongodb.com/manual/administration/free-monitoring/
- https://docs.mongodb.com/manual/administration/monitoring/
- https://blog.cloudacia.com/2019/05/15/mongobd-18-comandos-para-detectar-problemas-de-rendimiento/