BASES DE DATOS AVANZADAS

ACTIVIDAD 3 - CONCEPTOS Y COMANDOS BÁSICOS DEL PARTICIONAMIENTO EN BASES DE DATOS NOSQL

PRESENTADO A: WILLIAM RUIZ

PRESENTADO POR: DIEGO FABIAN RICO CASTRO

UNIVERSIDAD IBEROAMERICANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERIA DE DATOS
BOGOTA D.C
2024-05

Introduccion.

En la era digital actual, las organizaciones enfrentan desafíos significativos en la gestión de grandes volúmenes de datos que se generan a un ritmo acelerado. Las bases de datos NoSQL, como MongoDB, han emergido como soluciones efectivas para manejar datos no estructurados y semiestructurados a gran escala. Una de las características clave que permite a MongoDB gestionar eficientemente estos volúmenes de datos es el particionamiento, o sharding. El particionamiento distribuye los datos a través de múltiples máquinas, permitiendo a las bases de datos escalar horizontalmente y manejar cargas de trabajo intensivas sin comprometer el rendimiento. Este trabajo explora la importancia del particionamiento en MongoDB, detallando los requerimientos no funcionales necesarios para su implementación y presentando casos de prueba para asegurar su efectividad. A través de esta investigación, se demostrará cómo el particionamiento puede mejorar significativamente la escalabilidad, disponibilidad y rendimiento de una base de datos NoSQL.

Requerimientos no funcionales para particionamiento en MongoDB.

1. Rendimiento:
R1.1: El sistema de particionamiento deberá mantener un rendimiento óptimo, con tiempos de respuesta de consulta que no excedan los 100 milisegundos para el 95% de las operaciones.
R1.2: Las operaciones de escritura y actualización deberán distribuirse uniformemente entre los shards para evitar cuellos de botella, manteniendo una latencia de escritura por debajo de los 50 milisegundos.

2. Escalabilidad:
R2.1: El sistema deberá ser capaz de escalar horizontalmente, permitiendo la adición de nuevos shards sin interrumpir el servicio ni degradar el rendimiento.
R2.2: El sistema de particionamiento deberá soportar la redistribución automática de datos al agregar o remover shards, equilibrando la carga de manera eficiente.

3. Disponibilidad:
R3.1: El sistema deberá garantizar una alta disponibilidad de los datos, con un tiempo de recuperación de menos de 1 minuto en caso de falla de un shard.
R3.2: Deberá implementarse un mecanismo de failover automático para shards, asegurando la continuidad del servicio en caso de fallos.

4. Consistencia:
R4.1: El sistema deberá asegurar la consistencia de los datos a través de todos los shards, manteniendo la coherencia en las operaciones de lectura y escritura.
R4.2: En caso de operaciones de actualización concurrentes, el sistema deberá garantizar la integridad de los datos mediante mecanismos de control de concurrencia.

5. Seguridad:
R5.1: Los datos en tránsito entre shards deberán estar cifrados para proteger la confidencialidad e integridad de la información.
R5.2: El acceso a los datos particionados deberá ser controlado mediante autenticación y autorización basadas en roles, asegurando que solo usuarios autorizados puedan acceder y modificar los datos.

6. Mantenimiento y Operación:
R6.1: El sistema de particionamiento deberá ser fácil de mantener, permitiendo la monitorización y gestión de los shards a través de una interfaz intuitiva y herramientas de administración.
R6.2: Se deberán proporcionar mecanismos automatizados para la reconfiguración y actualización de los shards sin interrumpir el servicio.

7. Monitoreo y Diagnóstico:
R7.1: Deberá existir un sistema de monitoreo en tiempo real que proporcione métricas detalladas sobre el estado y el rendimiento de cada shard.
R7.2: Se deberán generar alertas automáticas en caso de anomalías o problemas en el particionamiento, permitiendo una intervención rápida por parte de los administradores del sistema.

8. Documentación y Soporte:

R8.1: Deberá haber una documentación exhaustiva que describa la configuración, operación y mantenimiento del sistema de particionamiento.
R8.2: Se deberá proporcionar soporte técnico adecuado para resolver cualquier problema o duda relacionada con la configuración o uso del particionamiento en MongoDB.

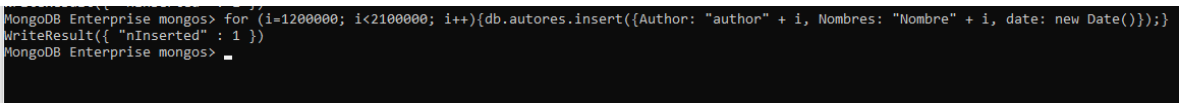
Comandos para ejecutar el particionamiento horizontal.

A continuación, una seria de comandos que nos ayudaran a crear una colección con datos ficticios, donde crearemos un total de 2.100.000 de documentos, este volumen de documentos nos ayudara a realizar y ver el funcionamiento de manera correcta del particionamiento.

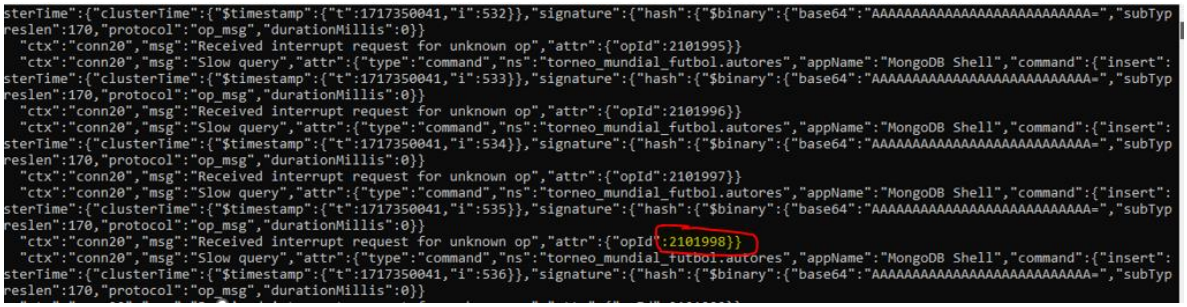
Crear Objeto usando la variable que nombramos cluster.
cluster = ShardingTest ({shards: 3, chunksize:1}) Con este creamos 3 shards con un pero maximo de 1mb.

Nos conectamos a la base creada en la actividd #1
db = (new Mongo("localhost:20006")).getDB("torneo_mundial_futbol")

Comando para ingresar 2.100.000 de registros en la colección "autores"
for (i=0; i<2100000; i++){db.autores.insert({Author: "author" + i, Nombres: "Nombre" + i, date: new Date()});}



En el Shell principal podemos ver como se crearon todos los registros dentro de la colección llamada "**autores**"



Con el comando
db.autores.find()
podemos ver los documentos creados dentro de la colección "autores"

Ahora crearemos una nueva variable que será del tipo de objeto de Mongo así:
shard1 = new Mongo("localhost:20000") --- Este nos conecta al localhost: 20000

```
---
MongoDB Enterprise > shard1 = new Mongo("localhost:20000")
connection to localhost:20000
```

Ahora con este comando nos conectaremos a la base “torneo_mundial_futbol"
shard1DB = shard1.getDB("torneo_mundial_futbol") ¿

```
MongoDB Enterprise > shard1DB = shard1.getDB("torneo_mundial_futbol")
torneo_mundial_futbol
```

Estando ya dentro de la base, usaremos el siguiente comando para explorar la colección “autores” y haremos un conteo para ver cuántos documentos tienes.
shard1DB.autores.count()

```
MongoDB Enterprise > shard1DB.autores.count()
2100000
MongoDB Enterprise >
```

Con los siguientes comandos que vemos en el shell nos conectaremos al shard2 donde veremos que allí aún no hay ningún documento.

```
MongoDB Enterprise > shard2 = new Mongo("localhost:20001")
connection to localhost:20001
MongoDB Enterprise > shard2DB = shard2.getDB("torneo_mundial_futbol")
torneo_mundial_futbol
MongoDB Enterprise > shard2DB = shard2DB.autores.count()
0
MongoDB Enterprise >
```

Hacemos lo mismo con el shard3.

```
MongoDB Enterprise > shard3 = new Mongo("localhost:20002")
connection to localhost:20002
MongoDB Enterprise > shard3DB = shard3.getDB("torneo_mundial_futbol")
torneo_mundial_futbol
MongoDB Enterprise > shard3DB = shard3DB.autores.count()
0
MongoDB Enterprise >
```

Hasta aquí como no hemos ejecutado ningún comando de partición vemos que todos los documentos se centran en un solo Shard.

Lo primero que debemos hacer es generar la activación del Sharding. Así:

```
MongoDB Enterprise mongos> shard1 = new Mongo("localhost:20006")
connection to localhost:20006
```

Usamos la siguiente funciona para indicar sobre que base de datos queremos trabajar.
sh.enableSharding("torneo_mundial_futbol")

```
MongoDB Enterprise mongos> sh.enableSharding("torneo_mundial_futbol")
{
        "ok" : 1,
        "operationTime" : Timestamp(1717357094, 4),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1717357094, 4),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
MongoDB Enterprise mongos>
```

A continuación, vemos que en el puerto 20000 crearemos una indexación con el siguiente comando.
db.autores.ensureIndex({author: 1})

```
MongoDB Enterprise mongos> db.autores.ensureIndex({author: 1})
{
        "raw" : {
                "__unknown_name__-rs0/LAPTOP-UDCL5TFG:20000" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1717357355, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1717357355, 3),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
MongoDB Enterprise mongos>
```

Con el siguiente comando le indicamos que empiece a compartir la información y creamos un shard collection.
sh.shardCollection("torneo_mundial_futbol", {author: 1})

```
MongoDB Enterprise mongos> sh.shardCollection("torneo_mundial_futbol.autores", {author: 1})
{
        "collectionsharded" : "torneo_mundial_futbol.autores",
        "collectionUUID" : UUID("4c5a0303-7e84-42a3-b84e-d1f84b2c3a8d"),
        "ok" : 1,
        "operationTime" : Timestamp(1717357674, 11),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1717357674, 11),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
MongoDB Enterprise mongos>
```

Hasta aqui el balanfceador no esta ejecutando aun su función, esto lo podemos comprobar con elasiguietnet comando.
sh.getBalancerState()

```
MongoDB Enterprise mongos> sh.getBalancerState()
false
MongoDB Enterprise mongos>
```

Para activar su funcionamiento lo haremos con el siguiente comando.
sh.setBalancerState(true)

```
MongoDB Enterprise mongos> sh.setBalancerState(true)
{
        "ok" : 1,
        "operationTime" : Timestamp(1717358061, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1717358061, 3),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
MongoDB Enterprise mongos>
```

Volver a validar el estado así:

```
MongoDB Enterprise mongos> sh.getBalancerState()
true
MongoDB Enterprise mongos>
```

Con el siguiente comando validamos si ya esta corriendo, ya que puede estar con estado activo pero sin correr.
sh.isBalancerRunning()

```
MongoDB Enterprise mongos> sh.isBalancerRunning()
false
MongoDB Enterprise mongos>
```

Con estos comandos logramos el perfecto particionamiento de nuestra base de datos, lo que nos ayudar a tener mejores rendimientos al momento de generar consultas. Para verificar esto podemos usar los siguientes comandos.

shard1 = new Mongo("localhost:20000")
shard1DB = shard1.getDB("torneo_mundial_futbol")
shard1DB.autores.count()

```
MongoDB Enterprise mongos>
MongoDB Enterprise mongos> shard1DB.autores.count()
700000
MongoDB Enterprise mongos>
```

shard2 = new Mongo("localhost:20001")
shard2DB = shard2.getDB("torneo_mundial_futbol")
shard2DB.autores.count()

```
MongoDB Enterprise mongos> shard2DB.autores.count()
700000
MongoDB Enterprise mongos>
```

shard1 = new Mongo("localhost:20000")
shard1DB = shard1.getDB("torneo_mundial_futbol")
shard2DB.autores.count()

```
MongoDB Enterprise mongos> shard3DB.autores.count()
700000
MongoDB Enterprise mongos>
```

Conclusiones

En conclusión, el particionamiento en MongoDB es una técnica esencial para gestionar grandes volúmenes de datos en entornos distribuidos, proporcionando una solución robusta y escalable para los desafíos de la era moderna. A través de la definición de requerimientos no funcionales claros y la realización de casos de prueba exhaustivos, se puede garantizar que el sistema de particionamiento cumpla con los estándares de rendimiento, disponibilidad y consistencia esperados. La capacidad de MongoDB para escalar horizontalmente mediante el particionamiento no solo mejora el rendimiento y la eficiencia del sistema, sino que también asegura la alta disponibilidad y tolerancia a fallos, aspectos críticos en el manejo de datos empresariales. En definitiva, el particionamiento permite a MongoDB ofrecer una plataforma de datos flexible y resiliente, adaptándose a las necesidades cambiantes y crecientes de las organizaciones en un mundo cada vez más digitalizado.