uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática
2024-2025

Arquitectura de Datos

"Practica 1.2 (FRAGMENTACIÓN)"

Adrian Cortázar Leiva

Jaime Vaquero Rabahieh

Tomás Mendizábal

Índice

1. Diseño de la arquitectura del cluster

Para esta sección, se documentará el diseño del cluster usado para el modo de replicación, en el cual se configuró de la siguiente manera:

- Nº Nodos: usamos un cluster de 3 nodos para soportar un número de dos réplicas de servidores secundarias en el caso de que un nodo principal falle.
- Puertos: Los puertos usados para los servicios correspondientes a cada uno de los nodos son los siguientes: 27018, 27019, 27020.
- Nombre de los servicios correspondientes: los nodos de réplica y el primario se lanzarán bajo los servicios de "mongocluster@i" donde la "i" será el número correspondiente al nodo del servidor de nuestro cluster.

Esta distribución del cluster es realizada en 3 nodos porque se considera que a partir de este número, las ventajas de la distribución no compensarán la mala centralización de los datos en este contexto.

De esta manera, se realiza la comparación de clústeres con datos de prueba pertenecienes a nuestros datos generados.

Nos dimos cuenta de que con un nodo no se iba a tener los beneficios de la replicación de nodos como la capacidad de asignar nodos primarios automática cuando el primario por defecto queda indispuesto.

Con dos nodos en el cluster, ya se pueden adquirir las características de replicación de nodos. Aunque decidimos añadir un nodo más por el hecho de tener seguridad de la disponibilidad de los datos en territorios más grandes (Ciudades o Paises). A demás, esta estructura triangular es resistente contra todo tipo de inconvenientes (hackeos físicos o digitales a nivel de software).

La configuración final se puede ver en la imagen siguiente:

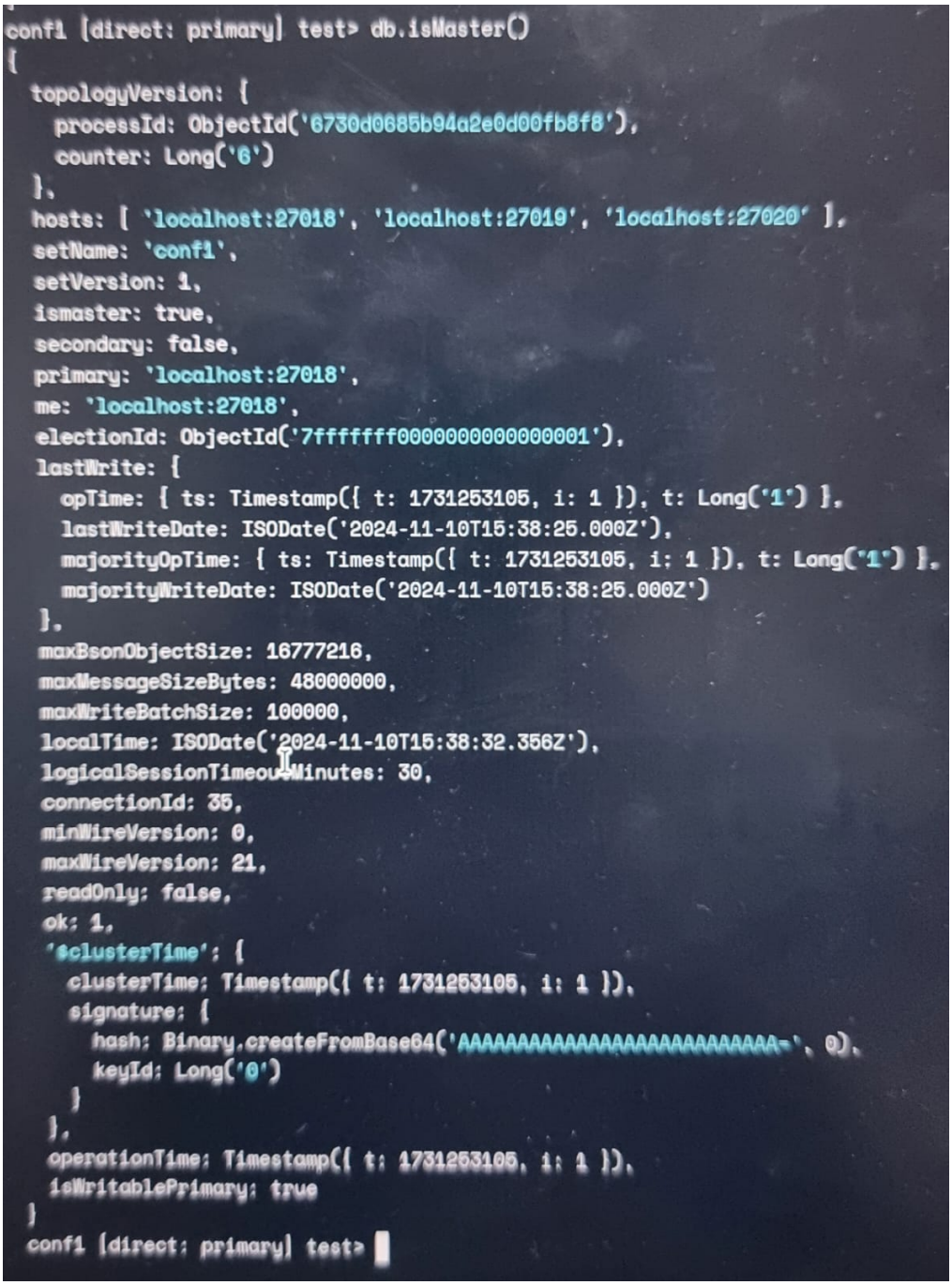

Figura 1: Configuración final

1.1. Fragmentos Creados

Se ha creado un fragmento (*shard*) por cada colección de nuestra base de datos. A continuación se detalla el código necesario para crear dichos clusters.

```
    db.createCollection("Juegos")
db.Juegos.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.Juegos", { "id": "hashed" })
// Areas
// crear un indice de tipo hashed para ID
db.createCollection("Areas")
db.Areas.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.Areas", { "id": "hashed" })
// IncidenteSeguridad
// crear un indice de tipo hashed para ID
db.createCollection("IncidenteSeguridad")
db.IncidenteSeguridad.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.IncidenteSeguridad", { "id": "hashed" })
// EncuestaSatisfaccion
// crear un indice de tipo hashed para ID
db.createCollection("EncuestaSatisfaccion")
db.EncuestaSatisfaccion.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.EncuestaSatisfaccion", { "id": "hashed" })
// EncuestaSatisfaccion
// crear un indice de tipo hashed para ID
db.createCollection("EncuestaSatisfaccion")
db.EncuestaSatisfaccion.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.EncuestaSatisfaccion", { "id": "hashed" })
// RegistroClima
// crear un indice de tipo hashed para ID
db.createCollection("RegistroClima")
db.RegistroClima.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.RegistroClima", { "id": "hashed" })
// Incidencia
// crear un indice de tipo hashed para ID
db.createCollection("Incidencia")
db.Incidencia.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.Incidencia", { "id": "hashed" })
// Usuario
// crear un indice de tipo hashed para NIF
db.createCollection("Usuario")
db.Usuario.createIndex({ "NIF": "hashed" })
sh.shardCollection("GameSystem.Usuario", { "NIF": "hashed" })
// Mantenimiento
// crear un indice de tipo hashed para ID
db.createCollection("Mantenimiento")
db.Mantenimiento.createIndex({ "id": "hashed" })
sh.shardCollection("GameSystem.Mantenimiento", { "id": "hashed" })
```

2. Criterios utilizados

2.1. Claves de fragmentación

La clave de fragmentación depende de como se vaya a utilizar la base de datos. Este se refiere a las distintas consultas que se vayan a realizar de manera frecuente en la BBDD. En nuestro caso para la fragmentación hemos decido usar los ids de cada colección. Esto se debe a que todos son valores únicos y serán fácilmente hasheables.

De este modo, se favorecerá el rendimiento de las consultas de lecturas en nuestra base de datos ya que cada uno de los datos tendrá un mayor índice de rápidez a la hora de ser leido. Esta elección también conlleva sus propias consecuencias. Por ejemplo, que al fragmentar por el campo "id", las inserciones de datos con los subcampos modificados o añadidos, resultarán más árduas.

2.2. Zonas de fragmentación para optimizar el rendimiento

Las zonas de fragmentación son una funcionalidad interesante para los clusters. Dichas zonas se pueden utilizar para acceder a los datos "calientes" o más utilizados con mayor facilidad. Debido a que no sabemos como se utilizará la BBDD no sabemos con exactitud las mejores zonas de fragmentación. Sin embargo, creemos que las colecciones que más se van a utilizar son Juegos, Áreas e incidencias por lo que creemos que ahi podrían estar las zonas de fragmentación.

2.3. Pre-splitting para evitar hotspots

Para evitar hotsposts se debe hacer las inserciones de manera inteligente para tener las filas que se quieran utilizar primero y no tener que esperar tanto a las inserciones, en especial si hay un gran número de ellas. En nuestro caso identificamos que las colecciones que tengan atributos de tipo fecha se filtraran para insertarse las más cercanas en el tiempo porque dichas filas son las que se van a utilizar a menudo. También se debería hacer un análisis de las zonas más utilizadas y priorizar la inserción de esas filas a la BBDD.

2.4. Número de fragmentos iniciales

Hemos decidido empezar con tres nodos para la fragmentación. Esta decisión se ha tomado debido a que en este momento no hay muchos datos en la BBDD. Sin embargo, a medida que se escala en complejidad y tamaño se deberán crear más nodos para poder fragmentar de manera eficiente la BBDD. Esto se dará por ejemplo cuando se agreguen más ciudades o cuando se tenga en cuenta más fechas (tiempo) para las distintas filas.

2.5. Activación y desactivación del balanceado

A medida que crezca el tráfico de operaciones en la BBDD será fundamental optimizar la activación y desactivación del balanceo de condiciones. Dicho balanceo se deberá activar en periodos de menor tráfico para mejorar las operaciones de nuestra BBDD. Sin embargo, cuando haya mucho trafico no deberá estas activado para poder evitar la sobrecarga.

2.6. Cantidad de instancias mongos necesarias

Por ahora se dispondrá de solo 2 instancias de Mongo para nuestro servicio. Esto es para garantizar la tolerancia a fallos por un lado y responder a la única funcionalidad que disponemos al momento lectura y escritura de datos. A medida que el número de aplicaciones para nuestros datos crezcan se deberán sumar más instancias, siempre teniendo en cuenta que esto sumará complejidad.

2.7. Número de servidores de configuración replicados

Se disponen de tres servidores de configuración de clusters para así garantizar la redundancia de nuestros datos. Se dispone de un número impar de servidores para poder hacer votaciones de quorum. Así garantizamos integridad y disponibilidad en un cluster, en especial cuando alguno de los servidores falla.