

**Actividad 3: Conceptos y comandos básicos del particionamiento en bases de datos
NoSQL**

Neyder Alexis Hernández Carrillo
Víctor Alfonso Méndez Palacios

Facultad de Ingeniería, Corporación Universitaria Iberoamericana

Ing. William Ruiz

17 de diciembre del 2023

Actividad 3: Conceptos y comandos básicos del particionamiento en bases de datos NoSQL

Enlace del repositorio Git-Hub con actividades y colecciones:

<https://github.com/NEYDER28/Evento-deportivo---MongoDB.git>

Enlace del video:

<https://drive.google.com/file/d/14HjfTac5fzrWv39SpXzwuw05DeGQrCJf/view?usp=sharing>

Requerimientos no funcionales para el proceso de particionamiento

Algunos de los escenarios en los que el particionamiento es necesario:

- Crecimiento esperado: cuando se anticipa un crecimiento del volumen de datos
- 1. Reducción de la latencia: en el caso de requerir distribuir los datos más cerca de los usuarios o aplicaciones que lo requieren.
- 2. Distribución geográfica: Cuando se hace necesario distribuir los datos geográficamente por causa de regulaciones, latencia, requisitos o redundancia.

Este documento detalla los criterios de calidad no funcionales relacionados con la redundancia y la disponibilidad 24x7 para el sistema que implica en MongoDB en el contexto del proyecto “Jankenpon”:

ID: RNF - 1	Nombre: Respuesta de las transacciones	Categoría: RENDIMIENTO
Descripción	El sistema debe garantizar, el ingreso, la consulta o la modificación de la información sin contratiempos de forma ágil.	
Criterio de medición	Los tiempos de respuesta promedio de una consulta en Mongo DB se encuentra entre los 112 y 3600 milisegundos por lo que no se deberá superar los 3600 milisegundos.	

ID: RNF - 2	Nombre: Disponibilidad del sistema	Categoría: DISPONIBILIDAD
Descripción	Mantener el funcionamiento permanente de la base de datos “Jankenpon” y sus colecciones (árbitros, deportistas y entrenadores).	
Criterio de medición	El sistema deberá proporcionar una disponibilidad del (99,9%) del tiempo	

ID: RNF - 3	Nombre: Respaldo continuo	Categoría: RESPALDO Y RECUPERACIÓN
Descripción	Se mantendrá el particionamiento en mínimo 3 shardings para mantener el balanceo de cargas y evitar caídas, con esto el sistema se puede recuperar de una sobrecarga.	
Criterio de medición	Distribuir la carga de datos a almacenar entre varios nodos.	

ID: RNF - 4	Nombre: Escalabilidad horizontal	Categoría: ESCALABILIDAD
Descripción	La base de datos deberá manejar la escalabilidad horizontal para mantener balanceado el rendimiento, capacidad y disponibilidad de los datos a medida que el tamaño de datos aumenta.	
Criterio de medición	Al aumentar la carga de datos, la base de datos deberá ser capaz de escalar sus recursos horizontalmente.	

ID: RNF - 5	Nombre: Recuperación	Categoría: TOLERANCIA A FALLOS / DISPONIBILIDAD
Descripción	El particionamiento deberá manejar las fallas de nodos individuales sin afectar los datos, balancear cargas para mantener el óptimo rendimiento y disponibilidad de la base de datos.	
Criterio de medición	Se deberá manejar las cargas para mantener el sistema balanceado	

Comandos para el particionamiento

Paso 1 -Arranque de la consola de Mongo

Instancia de MongoDB previamente arrancada

>

Paso 2 -Creación del grupo de particionado para pruebas

Para levantar el clúster de réplica, desde la consola escribiremos el siguiente comando:







➤ **cluster=new ShardingTest ({shards: 3, chunksize:1})**

Ejecución del Shell “segundo plano

```

> cluster=new ShardingTest ({shards: 3, chunksize:1})
Starting new replica set __unknown_name__-rs0
ShardingTest starting replica set for shard: __unknown_name__-rs0
ReplSetTest starting set '__unknown_name__-rs0'
ReplSetTest n is : 0
{
  "useHostName" : true,
  "oplogSize" : 16,
  "keyFile" : undefined,
  "port" : 20000,
  "replSet" : "__unknown_name__-rs0",
  "dbpath" : "$set-$node",
  "useHostname" : true,
  "shardsvr" : "",
  "pathOpts" : {
    "testName" : "__unknown_name__",
    "shard" : 0,
    "node" : 0,
    "set" : "__unknown_name__-rs0"
  },
  "setParameter" : {

```

Nombre	Fecha de modificación	Tipo	Tamaño
 __unknown_name__-configRS-0	17/12/2023 06:18 p. m.	Carpeta de archivos	
 __unknown_name__-configRS-1	17/12/2023 06:18 p. m.	Carpeta de archivos	
 __unknown_name__-configRS-2	17/12/2023 06:18 p. m.	Carpeta de archivos	
 __unknown_name__-rs0-0	17/12/2023 06:18 p. m.	Carpeta de archivos	
 __unknown_name__-rs1-0	17/12/2023 06:18 p. m.	Carpeta de archivos	
 __unknown_name__-rs2-0	17/12/2023 06:18 p. m.	Carpeta de archivos	

Paso 3 -Inserción de datos sobre el balanceador

Para actuar sobre el conjunto de Sharding, arrancaremos una nueva consola cliente de mongo contra el balanceador:

```
> db = (new Mongo("localhost:20006")).getDB("Jankenpon")
```

```

=====
> db = (new Mongo("localhost:20006")).getDB("Jankenpon")
Jankenpon
mongos> |

```

Ingresamos 200,000 registros en la tabla árbitros

```
for (i= 0; i < 200000; i++) {db.arbitros.insert({_id: "arbitro" +i, nombre:"nombre_arbitro" +i});}
```

```
> db = (new Mongo("localhost:20006")).getDB("Jankenpon")
Jankenpon
mongo> for (i= 0; i < 200000; i++) {db.arbitros.insert({_id: "arbitro" +i, nombre:"nombre_arbitro" +i});}
WriteResult({ "nInserted" : 1 })
mongo> |
```

Ingresamos 100,000 registros en la tabla posiciones

```
for (i= 0; i < 100000; i++) {db.posiciones.insert({competidor : "Competidor" +i, posicion : "posicion " +i});}
```

```
mongo> for (i= 0; i < 100000; i++) {db.posiciones.insert({competidor : "Competidor" +i, posicion : "posicion " +i});}
WriteResult({ "nInserted" : 1 })
mongo> |
```

Ingresamos 50,000 registros en la tabla entrenadores

```
for (i= 0; i < 50000; i++) {db.entrenadores.insert({ nombre : "nombre" +i, nacionalidad: "colombiano" });}
```

```
mongo> for (i= 0; i < 50000; i++) {db.entrenadores.insert({nombre : "nombre" +i, nacionalidad: "colombiano" +i });}
WriteResult({ "nInserted" : 1 })
mongo> |
```

Verificamos la inserción

```
>db.arbitros.count()
```

```
mongo> db.arbitros.count()
200000
```

```
>db.posiciones.count()
```

```
mongo> db.posiciones.count()
100000
```

```
>db.entrenadores.count()
```

```
mongo> db.entrenadores.count()
50000
```

Paso 4 -Comprobación de la distribución de datos en los nodos

> shard1 = new Mongo("localhost:20000")

```
> shard1 = new Mongo("localhost:20000")
connection to localhost:20000
> |
```

Me conecto a la base de datos Jankenpon

>shard1DB = shard1.getDB("Jankenpon")

```
> shard1DB = shard1.getDB("Jankenpon")
Jankenpon
> |
```

Verifico la inserción de registros

> shard1DB.arbitros.count()

```
> shard1DB.arbitros.count()
2000000
> |
```

Repetimos la misma secuencia para comprobar los registros que se han almacenado en la base de datos del segundo nodo del Shard:

> shard2= new Mongo("localhost:20001")

```
> shard2= new Mongo("localhost:20001")
connection to localhost:20001
> |
```

Me conecto a la base de datos Jankenpon

>shard2DB = shard2.getDB("Jankenpon")

```
> shard2DB = shard2.getDB("Jankenpon")
Jankenpon
> |
```

Verifico la inserción de registros

```
> shard2DB.arbitros.count()
```

```
> shard2DB.arbitros.count()
0
```

Repetimos la misma secuencia para comprobar los registros que se han almacenado en la base de datos del tercer nodo del Shard:

```
> shard3= new Mongo("localhost:20002")
```

```
> shard3= new Mongo("localhost:20002")
connection to localhost:20002
> |
```

Me conecto a la base de datos Jankenpon

```
>shard3DB = shard3.getDB("Jankenpon ")
```

```
> shard3DB = shard3.getDB("Jankenpon")
Jankenpon
```

Verifico la inserción de registros

```
> shard3DB.arbitros.count()
```

```
> shard3DB.arbitros.count()
0
```

Según esta comprobación, todos los documentos que insertamos se han almacenado en el primer nodo, pero el nodo 2 y 3 los cuales no contienen ningún documento en la colección Autores.

Paso 5 -Activación del Sharding

Para activar la funcionalidad de particionado de datos, debemos actuar sobre el balanceador. Por tanto, volveremos a la consola de mongo que hemos arrancado contra la instancia de mongos (que corría en el puerto 20006).

En dicha consola, podremos comprobar, a través de la función status(), el estado del grupo de Sharding:

```
> shard1 = new Mongo("localhost:20006")
```

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

```
>sh.status()
```

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("657f90feae9fd36bc456ce7e")
  }
  shards:
    { "_id" : "__unknown_name__-rs0", "host" : "__unknown_name__-rs0/DESKTOP-4VE9656:20000", "state" : 1 }
    { "_id" : "__unknown_name__-rs1", "host" : "__unknown_name__-rs1/DESKTOP-4VE9656:20001", "state" : 1 }
    { "_id" : "__unknown_name__-rs2", "host" : "__unknown_name__-rs2/DESKTOP-4VE9656:20002", "state" : 1 }
  active mongoses:
    "4.4.25" : 1
  autosplit:
    Currently enabled: no
  balancer:
    Currently enabled: no
    Currently running: no
    Failed balancer rounds in last 5 attempts: 0
    Migration Results for the last 24 hours:
      No recent migrations
  databases:
    { "_id" : "Jankenpon", "primary" : "__unknown_name__-rs0", "partitioned" : false, "version" : { "uuid" : UUID("694543ea-dd1e-42d1-a663-5cfc3eff100a"), "lastMod" : 1 } }
    { "_id" : "config", "primary" : "config", "partitioned" : true }
mongos> |
```

En este caso vemos, efectivamente, que el balanceador (el proceso que reparte la carga de datos entre los distintos nodos) no está activo (propiedad balancer: Currently enabled : no y balancer: Currently running: no) y, de hecho, el particionado de datos para el shard0000 (el nombre que se le ha dado automáticamente a nuestro shard al crear el ShardingTest) no está particionado (partitioned : false).

Para activar el Sharding utilizaremos la función enableSharding() sobre la base de datos que queremos que reparta sus datos entre todos los nodos:

```
>sh.enableSharding("Jankenpon")
```

```
mongos> sh.enableSharding("Jankenpon")
{
  "ok" : 1,
  "operationTime" : Timestamp(1702861553, 4),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1702861553, 4),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"),
      "keyId" : NumberLong(0)
    }
  }
}
mongos> |
```

Antes de habilitar los fragmentos, creará un índice en la clave que desea usar como shard:


```
>db.arbitros.ensureIndex({_id : 1})
```

```
mongos> db.arbitros.ensureIndex({_id : 1})
{
  "raw" : {
    "__unknown_name__-rs0/DESKTOP-4VE9656:200000" : {
      "numIndexesBefore" : 1,
      "numIndexesAfter" : 1,
      "note" : "all indexes already exist",
      "ok" : 1
    }
  },
  "ok" : 1,
  "operationTime" : Timestamp(1702860708, 109),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1702861736, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

Ahora puede determinar la colección de acuerdo con "_id":

```
>sh.shardCollection("Jankenpon.arbitros", {_id: 1})
```

```
mongos> sh.shardCollection("Jankenpon.arbitros", {_id: 1})
{
  "collectionsharded" : "Jankenpon.arbitros",
  "collectionUUID" : UUID("d39007d6-8c6c-427c-9d23-f09ed8c66fda"),
  "ok" : 1,
  "operationTime" : Timestamp(1702861940, 8),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1702861940, 8),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
mongos> |
```

Después de unos minutos, ejecute sh.status () nuevamente, puede ver que esta información de salida es más explícita.

Paso 6 -Activación del balanceador de carga

En este caso el shard0000 está en modo particionado (partitioned : true), sin embargo, el balancer (que es quien de forma efectiva moverá los datos) no se ha llegado a ejecutar. De hecho, podemos comprobar su estado de ejecución con getBalancerState().

```
> sh.getBalancerState()
```

```
>false
```

```
mongos> sh.getBalancerState()
false
mongos> |
```

Ahora sólo nos queda conseguir que el balancer comience a ejecutarse, para ello, utilizamos la función setBalancerState(boolean).

```
> sh.setBalancerState(true)
```

```
mongos> sh.setBalancerState(true)
{
  "ok" : 1,
  "operationTime" : Timestamp(1702862165, 3),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1702862165, 3),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
mongos> |
```

```
> sh.getBalancerState()
```

```
>true
```

```
mongos> sh.getBalancerState()
true
mongos> |
```

```
balancer:
  Currently enabled:  yes
  Currently running:  no
  Failed balancer rounds in last 5 attempts:  0
  Migration Results for the last 24 hours:
    No recent migrations
databases:
  { "_id" : "Jankenpon", "primary" : "__unknown_name__-rs0",
```

```
>sh.isBalancerRunning()
```

```
>true
```

```
mongos> sh.isBalancerRunning()
false
mongos> |
```

En este momento, si nos movemos a la consola de mongo donde creamos el objeto ShardingTest, veremos que las trazas nos indican que el balanceador ha detectado el desequilibrio y está compensándolo:

Paso 7 - Parada del clúster de particionado de datos

Por último, para parar nuestro ShardingTest, volveremos a la consola inicial, donde creamos el objeto que levantó todos los nodos y ejecutaremos la función stop() sobre el clúster que tenemos:

```
> cluster.stop()
```

Paso 8-Conocer el listado de los Shards activos

```
>db.adminCommand( { listShards: 1 } )
```

```

        balancing: true
        chunks:
            __unknown_name__-rs0    1
            { "_id" : { "$minKey" : 1 } } --> { "_id" : { "$maxKey"
{ "_id" : "config", "primary" : "config", "partitioned" : true }
mongos> db.adminCommand( { listShards: 1 } )
{
  "shards" : [
    {
      "_id" : "__unknown_name__-rs0",
      "host" : "__unknown_name__-rs0/DESKTOP-4VE9656:20000",
      "state" : 1
    },
    {
      "_id" : "__unknown_name__-rs1",
      "host" : "__unknown_name__-rs1/DESKTOP-4VE9656:20001",
      "state" : 1
    },
    {
      "_id" : "__unknown_name__-rs2",
      "host" : "__unknown_name__-rs2/DESKTOP-4VE9656:20002",
      "state" : 1
    }
  ],
  "ok" : 1,
  "operationTime" : Timestamp(1702862488, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1702862488, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
mongos> |

```

Conclusiones

En el marco del proyecto "Jankenpon", los requisitos no funcionales para el particionamiento en MongoDB establecen criterios cruciales para garantizar el rendimiento, la disponibilidad y la integridad de la base de datos, por ello para el proceso de particionamiento se realizó un marco de criterios para la validación de dichos requisitos no funcionales.

Referencias

Mongodb. (30 de 11 de 2023). *MongoDB community*. Obtenido de <https://www.mongodb.com/try/download/community>

Sarasa, A. (2016). *Introducció a las bases de datos NoSQL usando MongoDB*. UOC.