Universidad de Oviedo

Universidad de Oviedo
EPI GIJÓN

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

ÁREA DE PROYECTOS DE INGENIERÍA

TRABAJO FIN DE MÁSTER Nº 2018013

CONFIGURACIÓN DE UN ENTORNO BASADO EN CLOUD COMPUTING PARA TRANSFERENCIA DE DATOS DE CONSUMO ELÉCTRICO

Dña. Rodríguez Del Campo, Estefanía
TUTOR: D. Rodríguez Montequín, Vicente

FECHA: Julio 2018

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

1.- INTRODUCCIÓN

En la actualidad, gran parte de las conexiones a Internet se corresponden con dispositivos directamente utilizados por los seres humanos, tales como ordenadores y teléfonos móviles. Los usuarios de Internet mezclan el mundo físico y el mundo de la información. Se puede decir que estamos en la era del Internet de las cosas (IoT, *Internet of Things*), en la que las nuevas formas de comunicación entre humanos y las cosas, e incluso las cosas entre sí, ya son una realidad. Y en un futuro, los objetos previsiblemente podrán intercambiar información por sí mismos y el número de cosas conectados a Internet será mucho mayor que el número de personas.

Se prevé que miles de millones de objetos, conectados a Internet en tiempo real, serán configurados con diferentes tipos de sensores y actuadores, generando una elevada cantidad de flujo de datos que deben ser procesados de forma eficiente. Esto puede ser posible gracias a la integración de IoT con *Cloud Computing*, al permitir que estos datos se almacenen en Internet y a su vez estén disponibles desde cualquier lugar, proporcionando una infraestructura virtual de integración para dispositivos de almacenamiento.

El Internet de las Cosas se puede definir como el mundo en el que cada objeto tiene una identidad virtual propia y capacidad potencial para integrarse e interactuar de manera independiente en la Red con cualquier otro individuo, ya sea una máquina (M2M) o un humano. Cuenta con sensores o diversos objetos que generan información desde cualquier sitio accesible, o bien desde el interior de un dispositivo; los cuales requieren una interconexión a través de Internet. Con el aumento del IoT y el constante desarrollo de aplicaciones *Cloud Computing*, los problemas tecnológicos disminuyen y es posible una ampliación de servicios.

1.1.- Hipótesis de partida y alcance

La plataforma *cloud* (en este proyecto la denominaremos *Energy Analytics*) diseñada con el popular sistema de bases de datos MariaDB de la que se parte en este trabajo, tiene una estructura que permite el procesamiento masivo de los datos provenientes de múltiples suscriptores de servicio (clientes) y el procesamiento de estos con técnicas de análisis de

datos inteligenes para extraer la información de cara a la toma de decisiones, de forma que el sistema capture los datos generados por equipos de medida (dataloggers) situados en las instalaciones de los clientes y los almacene en la plataforma alojada en un servicio de hosting (Cloud Datacenter).

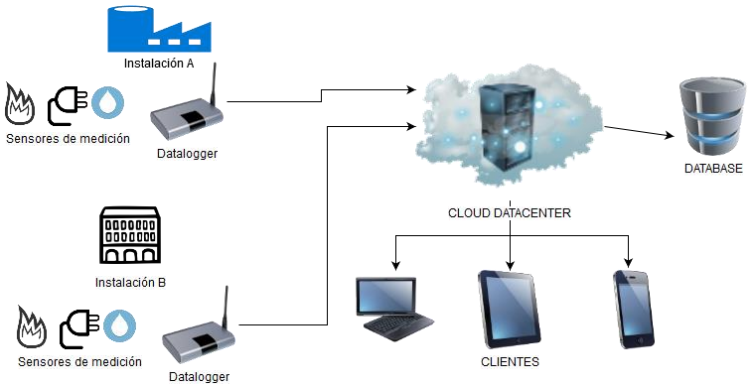

Figura 1.1. – Esquema general de la plataforma de partida.

El gestor de bases de datos empleado en esta plataforma permite trabajar con grandes cantidades de información. Pero al tratarse de MariaDB, una base de datos relacional presenta unos claros inconvenientes en cuanto a la escalabilidad y a la poca eficiencia del procesamiento masivo de datos.

El presente documento describe el trabajo fin de máster (TFM) consistente en el análisis y propuesta de mejora de esta plataforma, empleando el concepto de *Cloud Computing*, de forma que realice eficientemente el procesamiento masivo de los datos de consumo energético procedentes de los dispositivos de medición.

Esta mejora de la plataforma se basará en sistemas de base de datos NoSQL (Not Only SQL) para así poder combatir los inconvenientes que presenta MariaDB. Entre todos los existentes, se opta por MongoDB con el fin de introducir un conjunto de réplicas que proporcionen redundancia y alta disponibilidad (*replicación*), y la fragmentación de datos a través de varias máquinas para satisfacer las demandas de crecimiento de datos (*sharding*).

1.2.- Objetivos

Como objetivo principal de este trabajo se propone la mejora de la plataforma *cloud* para el procesamiento de masivo de datos de un sistema concreto con MariaDB,

incorporando los conceptos de replicación y sharding. Los objetivos concretos de este TFM son:

- Primeros pasos en MongoDB con la configuración de *replicación* y *sharding*, así como sus correspondientes pruebas.
- Realización de las pruebas de carga en la plataforma *cloud* ya existente.
- Configuración final del entorno utilizando ya todos los elementos de la plataforma inicial y realización de las pruebas de carga, tanto para *replicación* como para *sharding*. Deberá buscarse una mejora de los tiempos de procesamiento de los datos.

Además de los objetivos mencionados, también se tiene el objetivo académico del estudio de la familia de las bases de datos relacionales y especialmente las NoSQL (no relacionales), profundizando concretamente en MariaDB y MongoDB.

1.3.- Organización de la memoria

El documento presenta por un lado los aspectos conceptuales de los elementos a tener en cuenta para el desarrollo de la mejora de la plataforma. Por otro lado, se presenta una propuesta del entorno mejorado basado en *Cloud Computing* y, finalmente, se configuran y muestran los resultados de las pruebas de carga con el entorno final.

La documentación del trabajo estará estructurada según los siguientes apartados:

- **Capítulo 1:** Hipótesis de partida y alcance, objetivos y organización de la memoria.
- **Capítulo 2:** Se hace referencia a la Descripción de las tecnologías utilizadas, destacando especialmente el *Cloud Computing* y las técnicas de *replicación* y *sharding*.
- **Capítulo 3:** Se describe la configuración y las pruebas iniciales de *replicación* y *sharding* con la base de datos MongoDB. Se muestran los resultados obtenidos.
- **Capítulo 4:** Se describe ya la arquitectura final del sistema con la configuración de las máquinas del clúster, comprobando el funcionamiento del

sharding y, realizando también unas determinadas pruebas de carga. Se muestran los resultados obtenidos.

- **Capítulo 5**: Por último, se exponen las conclusiones extraídas a lo largo de todo el proceso de mejora del entorno, así como un apartado de posibles mejoras y líneas de trabajo futuro.

2.- DESCRIPCIÓN DE LAS TECNOLOGÍAS

A lo largo de este apartado se explicará las tecnologías necesarias para la realización y comprensión del nuevo entorno llevado a cabo.

2.1.- Bases de datos

En la década de los 70 nacen las bases de datos relacionales, que utilizan un lenguaje de consulta estructurado llamado SQL, con la idea de organizar la información en grupos de datos bajo una semejanza y así poder mantener una coherencia entre ellos. Pero el volumen de datos fue creciendo y empezaron los problemas, como el de escalabilidad que fue reconocido por empresas con grandes necesidades de datos e infraestructura, como Google, Amazon y Facebook.

Los problemas de SQL dieron lugar a una serie de sistemas de gestión de bases de datos NoSQL (DBMS), que se centraban en el rendimiento, la fiabilidad y la coherencia (integridad). En los últimos años ha habido una clara tendencia hacia la utilización de estas bases de datos.

La clave entre las bases de datos NoSQL y las bases de datos relacionales tradicionales es el hecho de que NoSQL es una forma de almacenamiento no estructurado. Lo cual significa que esta última no tiene una estructura de tabla fija como las que se pueden encontrar en las bases de datos relacionales.

2.1.1.- MariaDB

MariaDB [1] está basado en MySQL y es uno de los servidores de bases de datos más populares del mundo. Desarrollado, como software de código abierto y como base de datos relacional, por la comunidad en conjunto con Monty Program Ab como su principal encargado. Al estar basado en MySQL, posee las mismas órdenes, interfaces, APIs y bibliotecas. Introduce dos motores de almacenamiento nuevos, uno llamado XtraDB y otro llamado Aria.

Permite convertir los datos de una amplia gama de aplicaciones en información estructurada. Es uno de los mas populares debido a que es rápido, escalable, robusto, con

una gran variedad de motores de almacenamiento, complementos y muchas otras herramientas que lo hacen muy versátil para su uso.

2.1.2.- MongoDB

MongoDB [2] es actualmente el sistema de base de datos NoSQL más popular. El crecimiento ha sido impresionante en los últimos años como un sistema de base de datos orientado a documentos que utiliza objetos similares a JSON para describir el modelo de datos.

Los sistemas de bases de datos NoSQL son particularmente bienvenidos para servicios web que necesitan escalabilidad y también adaptarse rápidamente a cambios del modelo de datos. En lugar de guardar los datos en tablas como se hace en las DBs relacionales, MongoDB guarda estructuras de datos en documentos similares a JSON, consiguiendo que la integración de los datos sea más fácil y rápida.

Se ha creado para proporcionar disponibilidad, rendimiento y escalabilidad, de forma que partiendo de un servidor único se escale a grandes arquitecturas complejas de centros multidatos. Además, brinda un elevado rendimiento tanto para lectura como para escritura, potenciando la computación en memoria.

Algunas de las principales características de MongoDB son:

- Base de datos orientada a documentos, o lo que es lo mismo, no es relacional. Se puede comparar el concepto de fila en SQL al de documento, también el de colección al de tabla y el de base de datos a esquema.
- Fácilmente escalable.
- Es indexable, incluso permite índices geoespaciales.
- Sustituye el concepto de procedimientos almacenados por funciones JavaScript.
- Permite colecciones de tamaño fijo que ofrecen una alta velocidad de acceso y modificación.

La replicación, que es nativa de MongoDB, y la tolerancia a fallos ofrecen fiabilidad y flexibilidad a nivel empresarial. Muchas empresas líderes como Google, Facebook y Amazon ya están utilizando bases de datos “no-relacionales” en sus muchos clústeres.

2.2.- Big Data

El Big Data o tecnología de procesado de millones de datos en tiempo real. Estos sistemas tienen unas determinadas características que los hacen únicos:

- Volumen. El volumen de los datos es de gran tamaño.
  - Demasiado grandes para la memoria principal.
  - En el caso de que esté indexado, los índices también pueden ser demasiado grandes para la memoria principal.
- Velocidad. Los datos se generan/reciben muy rápido y deben procesarse rápidamente.
- Variedad. Hay muchos tipos de datos.
  - Los datos no tienen estructuras relativamente estables.
- Volatilidad. Los datos cambian rápidamente y eso hace que tengan una validez muy corta.

El tamaño utilizado para determinar si un conjunto de datos se considera Big Data no está firmemente definido, aunque la mayoría de los analistas y profesionales actualmente se refieren a conjuntos de datos que van desde los 30-50 Terabytes (Tb) a varios Petabytes (Pb).

La naturaleza compleja del Big Data se debe principalmente a la naturaleza no estructurada de gran parte de los datos generados por las tecnologías modernas, como la identificación por radiofrecuencia (RFID), los sensores incorporados en los dispositivos, las búsquedas en Internet, etc.

Teniendo en cuenta estas características definidas anteriormente, los sistemas Big Data deben cumplir una serie de objetivos:

- Proporcionar un alto rendimiento y capacidad de almacenamiento.
- Proporcionar una fiabilidad alta.
  - No debe existir un solo punto de fallo.
- Escalabilidad horizontal.
  - Debe permitir añadir más máquina cuando se necesite escalar.

Para conseguir estos objetivos se usan muchas técnicas, entre ellas las de replicación y sharding.

2.3.- Internet de las Cosas

El Internet de las Cosas (IoT) y Big Data se compenetran perfectamente, y es que una de las consecuencias de la llegada del IoT es la generación de muchos datos que son analizados a través de la tecnología Big Data con la finalidad de ofrecer un mejor servicio al usuario.

El IoT es un concepto que engloba infinitas posibilidades porque aporta la conectividad entre dispositivos.

En 1999 Kevin Ashton acuñó este término de IoT y, desde entonces, el IoT se afianzó como un nuevo paradigma en las comunicaciones inalámbricas. El concepto se generaliza alrededor de una gran variedad de cosas u objetos de carácter cotidiano, como sensores, actuadores, smartphones, etiquetas de identificación por radiofrecuencia (RFID) y demás dispositivos.

El IoT surgió con la necesidad de la identificación de objetos, personas y animales mediante el uso de etiquetas inteligentes RFID. Con ello se consiguió otorgar de un identificador único a un objeto determinado. Pero para la existencia del IoT, son necesarios tres puntos:

- Inteligencia Embebida. La integración de inteligencias que hacen que se puedan llevar a cabo acciones automáticamente. Por ejemplo: el software de control de vuelo y sistemas integrados en aviones y misiles, la etiqueta RFID en los alimentos que puede grabar información de la comida gracias a un lector RFID, el controlador antibloqueo para móvil, etc.
- Conectividad. La posibilidad de conectar todos los dispositivos inteligentes ya sea por cable o de manera inalámbrica. Existen numerosas formas de conectarlos como, por ejemplo: WiFi, WiMax, LAN, WAN, WPAN, 3G, RFID, ZigBee, etc.
- Interacción. La propia interacción entre los objetos. Deben crearse objetos inteligentes que puedan procesar información, autoconfigurarse, auto-mantenerse y auto-repararse.

Si se combinan estos, se puede conseguir que un objeto concreto realice determinadas acciones en base a información recogida por otro, como puede ser un sensor u otro objeto.

En definitiva, nos dirigimos hacia un futuro en el que los objetos inteligentes detectan el entorno en el que se encuentran, interactuando inclusive con otros objetos. Y estos datos procedentes del IoT supondrán, a corto plazo, el mayor flujo de información que existe en Internet y, por consiguiente, el mayor proveedor para los sistemas Big Data, Inteligencia Artificial (IA) y Computación Cognitiva.

2.4.- Cloud computing

Cloud Computing, conocida también como servicios en la nube, informática en la nube, nube de conceptos o simplemente "la nube"; es un paradigma que permite ofrecer servicios informáticos (servidores, bases de datos, redes, software, etc) a través de la red, que normalmente es Internet.

Existen 6 razones principales por las que las organizaciones recurren a la computación en la nube:

- Coste. La computación en la nube elimina el gasto de comprar hardware y software, de la configuración y puesta en marcha, de electricidad y refrigeración diaria y de los expertos IT para la gestión de la infraestructura.
- Velocidad. Muchos de los servicios de computación en la nube son autoservicio y bajo demanda, por lo que gran cantidad de recursos se proporcionan en minutos, permitiendo a las empresas gran flexibilidad y mejorando la planificación de la capacidad.
- Escalabilidad global. Los beneficios de computación en la nube incluyen la habilidad de escalar fácilmente. En otras palabras, proporcionar la cantidad correcta de recursos IT (almacenamiento, ancho de banda, capacidad de cómputo, etc.) exactamente cuando son necesarios y desde la mejor ubicación geográfica.
- Productividad. Los centros de datos típicamente requieren una gran cantidad de "racking" y apilamiento que consumen mucho tiempo. Con esta computación se elimina la necesidad de muchas de estas tareas, de forma que los equipos de IT pueden aprovechar ese tiempo en objetivos empresariales más importantes.

- Rendimiento. La mayoría de los servicios de computación en la nube se ejecutan en una red mundial segura de centros de datos, los cuales son regularmente actualizados a la última versión de hardware más rápida y eficiente. Esto ofrece bastantes beneficios sobre un centro de datos, incluyendo la reducción de la latencia en aplicaciones y mejorando la economía de escala.
- Fiabilidad. Hace que la copia de seguridad de datos, la recuperación ante desastres y la continuidad del negocio más fácil y menos cara, porque los datos pueden ser replicados en varios sitios en la nube del proveedor de la red.

La elección de este modelo de computación para este proyecto es debida a varias razones. En primer lugar, se considera que uno de los factores clave para el éxito del sistema es proporcionar a los clientes los informes y análisis de la información de una manera ubicua, característica esencial asociada a este paradigma de computación. Por otro lado, la empresa que comercializa este servicio no quiere estar atada por las limitaciones que supone depender de sistemas propios para prestar el servicio, prefiriendo sacar partido de los servicios de prestación de altas capacidades de computación que se ofrecen hoy en día.

Si siguiera un modelo de software tradicional, se vería obligada a mantener sus propios servidores, disponer de una infraestructura de conexión a Internet dimensionada adecuadamente para recibir y procesar los datos (que como se ha comentado implica volúmenes de datos elevados) y disponer de personal dedicado exclusivamente a labores de mantenimiento informático.

El modelo basado en *Cloud Computing* permite contratar el alojamiento en un proveedor especializado (lo que se conoce habitualmente como un servicio de alojamiento o *hosting*), pero además las características de provisión de servicios permiten que sea algo adaptable dinámicamente, escalándose de manera automática en función del número de clientes y servicios a proporcionar. Por lo que la utilización de este paradigma permitirá a la empresa contratar en cada momento la capacidad de cómputo adecuada a cada situación, en función del número de clientes y volumen de datos a tratar, suponiendo una importante optimización de costes.

2.5.- Clusterización

Un clúster es un grupo de servidores independientes interconectados a través de una red dedicada, que trabaja como si fuera un recurso de procesamiento de datos centralizado. Los clústeres son capaces de realizar multitud de instrucciones complejas distribuyendo la carga entre todos los servidores conectados.

La clusterización es una técnica de minería de datos dentro de una disciplina de Inteligencia Artificial que identifica de forma automática agrupaciones o clústeres de elementos de acuerdo con una medida de similitud entre ellos. El objetivo fundamental de estas técnicas consiste en identificar grupos o clústeres de elementos tal que:

- La similitud media entre elementos del mismo clúster sea alta.
- La similitud media entre elementos de distintos clústeres sea baja.

2.6.- Replicación

La replicación, de manera genérica, se puede ver como la copia de una tabla entera o una base de datos en múltiples servidores. Se utiliza para mejorar la velocidad de acceso a los registros de referencia tales como datos maestros.

2.6.1.- En MongoDB

Consiste en un conjunto de réplicas de datos, que a su vez son un conjunto de instancias con los mismos datos. Dentro del conjunto existen instancias o nodos primarios y secundarios, como se puede ver en la siguiente imagen:

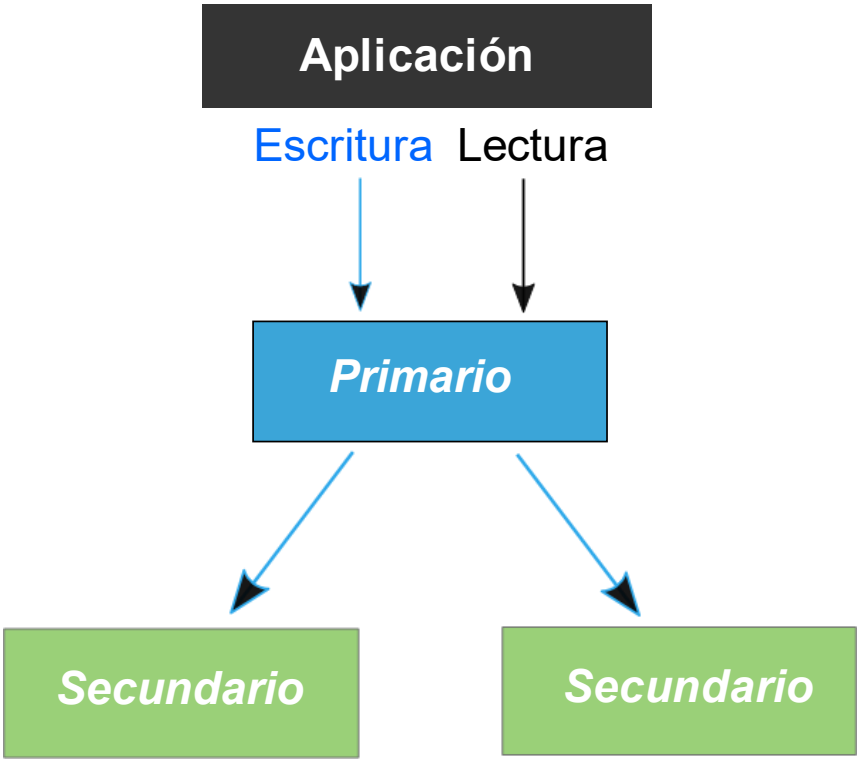

Figura 2.1. – Esquema Replicación.

La replicación se configura a nivel de cada instancia de MongoDB. Los elementos imprescindibles son:

- Nodo primario: acepta todas las operaciones de escritura del cliente.
  - Sólo hay un nodo primario en el conjunto de réplicas
  - Acepta todas las operaciones de lectura
  - Si falla el primario, el conjunto elige un nuevo primario
- Nodos secundarios: replica el conjunto de datos del primario.
  - Se propagan las actualizaciones asíncronamente a todos los secundarios
  - Pueden aceptar solicitudes de lectura, pero no de escritura
  - De forma predeterminada, se hacen copias de seguridad de las instancias sin servicio
- Heartbeats (latidos): los miembros del conjunto de réplicas envían *pings* (latidos) cada dos segundos. Si uno de estos *pings* no vuelve dentro de 10 segundos, los otros miembros marcan a este como inaccesible.

2.7.- Sharding

Con el fin de lograr los objetivos de alto rendimiento, escalabilidad horizontal y capacidad de almacenamiento, se puede complementar la replicación con el *sharding* o particionado de información.

Es una técnica que consiste en particionar los datos de una base de datos horizontalmente agrupándolos de forma que se consiga un enrutamiento más rápido.

2.7.1.- En MongoDB

Primeramente, hay que entender las variables que es necesario gestionar en Mongo. Un clúster particionado aquí consta de los siguientes componentes:

- Shard. Cada shard está formado por uno o más servidores que almacenan los datos usando los procesos mongod. En una situación de producción, cada shard estará formado por una réplica para asegurar la disponibilidad y failover (conmutación por error) automatizado.
- Mongos. Actúa como un enrutador de consultas, proporcionando una interfaz entre el propio cliente (aplicación) y el clúster con los shards.
- Servidores de configuración. Almacenan metadatos y las configuraciones para el clúster. A partir de una determinada versión, estos deben implementarse como un conjunto de réplicas.

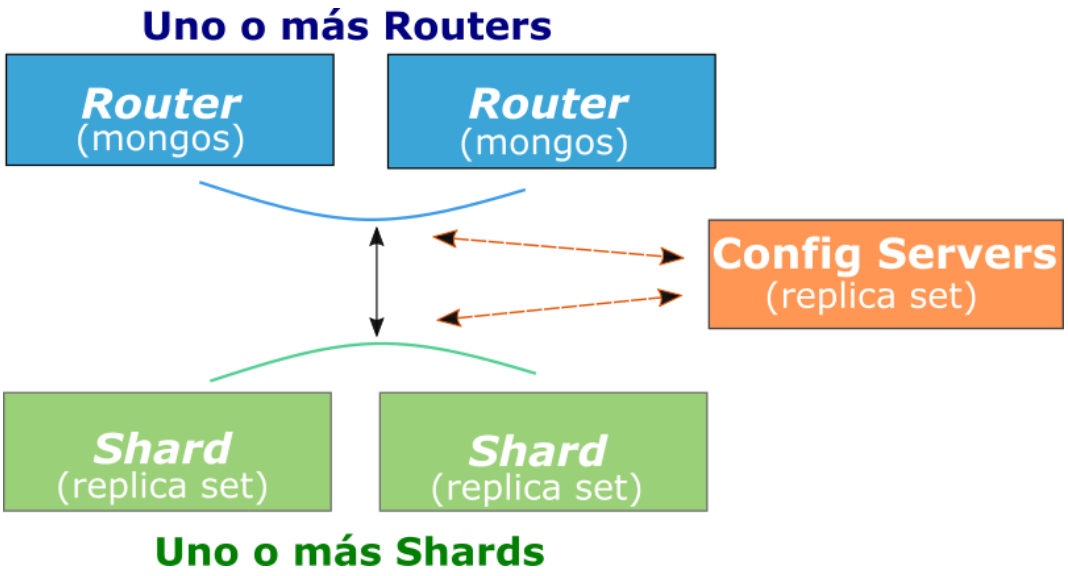

Figura 2.2. – Esquema Sharding.

Mongo divide los datos particionados en **chunks** o trozos, cuyo tamaño por defecto es de 64 MB. Cada uno de ellos tiene un rango inferior y superior, basado en la clave de shard.

Estos chunks se migran entre los diferentes shards utilizando un balanceador, intentando lograr un equilibrio entre todos los shards del clúster.

Clave de shard

Mongo distribuye los documentos de una colección según la clave de shard. Consiste en un campo o campos inmutables que existen en cada documento de la colección de destino.

Para escoger esta clave adecuadamente para su funcionamiento, se debe tener en cuenta lo siguiente:

- Está limitada a 512 bytes de tamaño
- Debe ser inmutable y estar indexada
- Solo la clave de shard puede ser única entre todos los shards
- Se usará para enrutar las "queries" o consultas

Existen una serie de buenas prácticas para escoger la clave de shard, como:

- Una que sea fácilmente divisible.
  - Hace que sea también fácil para MongoDB distribuir el contenido entre los shards.
  - Las claves de shard que tienen un número limitado de posibles valores, pueden dar como resultado trozos que no son divisibles.

- Una que tenga un alto grado de aleatoriedad.
    - Así se impide que cualquier fragmento se convierta en un cuello de botella y se distribuirán las operaciones de escritura entre el clúster.
- Una cuyo objetivo sea un fragmento único.
    - Una clave así hace posible que mongos devuelva la mayor parte de las consultas desde una instancia de mongod específica.
    - La clave de shard debe ser el campo principal utilizado por las queries.
- Shard con clave compuesta.
    - Un campo existente en la colección puede no ser la clave óptima.
    - A veces es mejor generar una clave de shard específica en un campo adicional o usar una clave compuesta.

Estrategias de sharding

- Hashed Sharding.

  Se calcula un hash del campo clave del shard. A cada shard se le asigna luego un rango basado en los valores de la clave hasheada del shard.

  La distribución de datos basada en este tipo de estrategia facilita una distribución más homogénea entre los elementos del clúster y, especialmente, en conjuntos donde la clave del shard cambia monótonamente.
- Sharding por rangos.

  Los datos se dividen en rangos basados en los valores clave del shard. A cada fragmento se le asigna un rango basado en los valores clave del fragmento.

  Con esta disposición se permiten operaciones específicas ya que los *mongos* pueden enrutar directamente las operaciones a los shards que contienen los datos requeridos.

  La eficacia del particionado depende de la clave de shard.
- Zonas de Sharding en los clústeres.

  Consiste en crear zonas de datos en función de la clave de shard, asociando una zona a uno o varios shards. De hecho, un mismo shard se puede asociar con varias zonas que no sean conflictivas.

  Cada zona cubre uno o varios rangos de claves de shard. Cada uno de estos rangos siempre incluye su límite inferior y excluye su límite superior.

Cuando se define un nuevo rango se deben usar los campos contenidos en la clave del shard para que abarque una zona. Y, si se usa una clave compuesta, el rango debe incluir el prefijo de la clave.

Al elegir la clave del shard, se debe de tener en cuenta la posibilidad de utilización de división de zonas en el futuro, ya que una vez particionada la colección, no se puede cambiar la clave del shard.

3.- ALTERNATIVAS TECNOLÓGICAS PARA LA MEJORA DE LA PLATAFORMA

Además de las tecnologías ya descritas de replicación y sharding, se podría pensar como posible mejora un cluster propiamente en MariaDB. Pero esto ya se ha estudiado y comprobado que no es una buena opción [3] por algunos motivos, como los siguientes:

- Hardware. El coste de la infraestructura y operaciones puede ser considerable, a pesar de tener el software gratuito. Y es que para la configuración de un clúster se necesitan muchos servidores, cada uno de ellos con bastante RAM; además de la necesidad de una posible subred dedicada, comprar NIC's (Network Interface Controller), etc.
- Esquema de diseño. Una arquitectura "sharded" necesita un buen esquema de diseño para conseguir todas las ventajas del *sharding*. En el caso de que en el esquema se incluya más de un shard, al realizar las consultas (queries) sobre instancias de cluster de MariaDB, éstas tienen un peor funcionamiento que si se usara una instancia convencional.
- Configuración y adaptación al sistema. NDB (Network Database) es el sistema de bases de datos subyacente a los clústers de MySQL, no es el motor de almacenamiento más usado en general en MySQL (InnoDB) y mucha de la documentación existente no se puede aplicar aquí. Resultaría costoso y complicado aprender en NDB sin haber visto nada previamente.

Asimismo, la base de datos NoSQL MongoDB es más rápida en general que una relacional como MariaDB. Diseñar con un modelo de datos sin estructura es mejor que diseñar con un modelo relacional.

Es por todo esto por lo que finalmente se va a descartar esta posible alternativa y se va a optar por la *replicación* y el *sharding* de MongoDB.

3.1.- Arquitectura propuesta

A continuación, se muestra una parte de la estructura inicial de la base de datos según estaba en MariaDB, en concreto de las tablas tratadas en este trabajo **Capturas** y **Canales**.

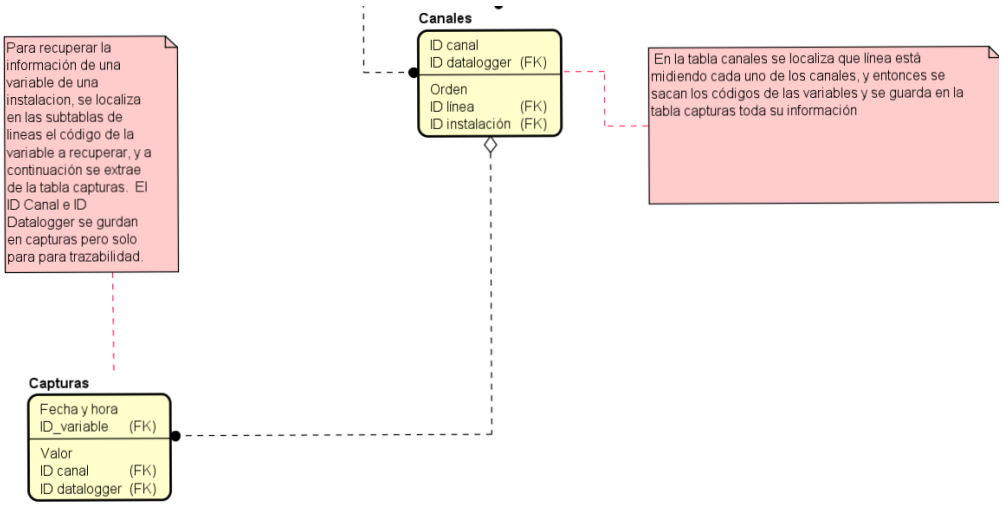

Figura 3.1. – Parte de la estructura inicial MariaDB.

El sistema conformado por esta DB, como ya se describió en la introducción de este trabajo, se trata de una plataforma de *cloud* que permite el procesamiento masivo de datos de múltiples clientes y el procesamiento de estos con técnicas de análisis de datos inteligentes para extraer la información relevante de cara a la toma de decisiones. Todo esto de forma que el sistema capture los datos generados por equipos de medida instalados en las instalaciones de los clientes y los almacene en una plataforma alojada en un servicio de *hosting*.

En concreto, la tabla **Capturas** contiene la mayor parte de la información del sistema, los datos de los sistemas de capturas instalados en las dependencias de los clientes para cada variable energética, canal y datalogger. A continuación, se muestra un ejemplo de estos datos de unos días del mes determinados:

fecha_hora	*id_variable*	*Valor*	*id_canal*	*id_datalogger*
6/8/2017 12:52	T6359356762d3007	21.3125	14	22
6/8/2017 12:52	T1659356762d30b9	224.325	14	22
6/8/2017 12:52	T1359356762d3169	43.358	14	22
6/8/2017 12:52	T7859356762d3219	0.725	14	22

6/8/2017 12:52	T6659356762d32c8	0.713	14	22
6/8/2017 12:52	T1459356762d3377	6938.26	14	22
6/8/2017 12:52	T9859356762d3427	6584.17	14	22
6/8/2017 12:52	T4559356762d34d6	9726.28	14	22
6/8/2017 12:52	T4159356762d35a2	2.09	14	22
6/8/2017 12:52	T9059356762d3655	20.01	14	22
6/8/2017 12:52	T7759356762d3705	223.921	14	22
6/8/2017 12:52	T3359356762d37b4	26.997	14	22
6/8/2017 12:52	T5559356762d3863	0.905	14	22
6/8/2017 12:52	T0059356762d3912	0.869	14	22
6/8/2017 12:52	T2859356762d39c1	5252.65	14	22
6/8/2017 12:52	T9959356762d3a71	2475.14	14	22
6/8/2017 12:52	T2059356762d3b20	6045.19	14	22
6/8/2017 12:52	T6359356762d3bd0	2.34	14	22
6/8/2017 12:52	T4459356762d3c7f	30.43	14	22
6/8/2017 12:52	T1159356762d3d2e	223.331	14	22
6/8/2017 12:52	T0259356762d3dde	45.939	14	22
6/8/2017 12:52	T5059356762d3e8e	0.915	14	22
6/8/2017 12:52	T1959356762d3f3d	0.903	14	22
6/8/2017 12:52	T5659356762d3fec	9261.69	14	22
6/8/2017 12:52	T1459356762d409c	4074.03	14	22
6/8/2017 12:52	T6859356762d414b	10259.6	14	22
6/8/2017 12:52	T1959356762d41fa	2.22	14	22
6/8/2017 12:52	T2759356762d42a9	18.25	14	22
6/8/2017 12:52	T4359356762d4358	223.859	14	22
6/8/2017 12:52	T8759356762d4407	387.732	14	22

6/8/2017 12:52	T2759356762d44b6	116.294	14	22
6/8/2017 12:52	T7459356762d4579	0.824	14	22
6/8/2017 12:52	T8359356762d4632	0.214526	14	22
6/8/2017 12:52	T8359356762d46e8	13133.3	14	22
6/8/2017 12:52	T7959356762d479c	26031.1	14	22
6/8/2017 12:52	T4859356762d4851	49.88	14	22
6/8/2017 12:52	T2059356762d4905	17.792	14	22
6/8/2017 12:52	T8459356762d49ba	233.097	14	22
6/8/2017 12:52	T0459356762d4a70	403.734	14	22
6/8/2017 12:52	T0159356762d4b24	370.732	14	22
6/8/2017 12:52	T9659356762d4bd9	0.995	14	22
6/8/2017 12:52	T0059356762d4c8d	80337.4	14	22
6/8/2017 12:52	T6359356762d4d42	23043.6	14	22
6/8/2017 12:52	T8159356762d4df6	83082.2	14	22
6/8/2017 12:52	T6759356762d4eab	50.07	14	22
6/8/2017 12:52	T8959356762d4f5f	90.1	14	22
6/8/2017 12:53	T6359356762d3007	21.3125	14	22
6/8/2017 12:53	T1659356762d30b9	224.484	14	22
6/8/2017 12:53	T1359356762d3169	43.386	14	22
6/8/2017 12:53	T7859356762d3219	0.721	14	22

Tabla 3.1. – Ejemplo datos tabla Capturas.

Por tanto, al contener esta tabla gran parte de la información masiva procesada por el sistema, es la más propensa para la configuración de sharding. Se tratará de ver si es conveniente sustituir esta tabla **Capturas** actual por otra en MongoDB, utilizando los métodos de *replicación* y *sharding*, o si por el contrario es preferible dejar la configuración actual.

4.- MONGODB

En este apartado se presentará el proceso de configuración de todos los elementos necesarios, empleando la base de datos MongoDB, para la mejora del entorno ya existente. Inicialmente se realizará un estudio en líneas generales de la estructura y el entorno con MariaDB, y se integrará con MongoDB. Después se tratará la propia configuración de este junto con la solución propuesta que permita procesar la cantidad de carga.

Dado que uno de los objetivos de este trabajo es el estudio de las DBs NoSQL, en este apartado se describe en detalle la configuración y el entorno de MongoDB, ya que será la plataforma que emplear.

4.1.- Configuración MongoDB

Primeramente, antes de empezar con las diferentes pruebas, se procede a la instalación de MongoDB basada en [4] con toda la documentación disponible sobre ello.

4.1.1.- Prueba replicación

Para el ejemplo de replicación se va a seguir la configuración explicada a continuación [5].

- Nodo primario: instancia para el nodo que será el que acepte todas las peticiones de escritura:
  - mongod con localhost en el puerto 27017.
- Nodos secundarios: instancias para los nodos donde se replicarán los datos del primario:
  - mongod con localhost en el puerto 27018.
  - mongod con localhost en el puerto 27019.

1) En primer lugar, se han creado 3 carpetas dentro de la ruta de la DB (/*data*/*db*/), para posteriormente lanzar cada una de las instancias en los puertos 27017, 27018 y 27019:

```
sydea@sydeaCloneIII:/data/db$ ls -la | grep repl
drwxrwxrwx  4 root root 4096 mar  2 21:10 repl_1
drwxrwxrwx  4 root root 4096 mar  2 21:10 repl_2
drwxrwxrwx  4 root root 4096 mar  2 21:10 repl_3
```

Figura 4.1. – Carpetas configuración réplica.

2) Se realiza la conexión con los 3 clientes:

```
mongod -replSet rs0 -dbpath /data/db/repl_1 -port 27017
mongod -replSet rs0 -dbpath /data/db/repl_2 -port 27018
mongod -replSet rs0 -dbpath /data/db/repl_3 -port 27019
```

3) Se define en el primer cliente la configuración de la réplica con la variable *rsconf*, y se inicializa el conjunto de réplicas:

```
> rs.initiate(rsconf);
{
        "ok" : 1,
        "operationTime" : Timestamp(1519856317, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1519856317, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rs0:SECONDARY> rs.status();
{
        "set" : "rs0",
        "date" : ISODate("2018-02-28T22:18:53.324Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1519856329, 5),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1519856329, 5),
                        "t" : NumberLong(1)
                },
```

```
            "appliedOpTime" : {
                    "ts" : Timestamp(1519856329, 5),
                    "t" : NumberLong(1)
            },
            "durableOpTime" : {
                    "ts" : Timestamp(1519856329, 5),
                    "t" : NumberLong(1)
            }
    },
    "members" : [
            {
                    "_id" : 0,
                    "name" : "localhost:27017",
                    "health" : 1,
                    "state" : 1,
                    "stateStr" : "PRIMARY",
                    "uptime" : 300,
                    "optime" : {
                            "ts" : Timestamp(1519856329, 5),
                            "t" : NumberLong(1)
                    },
                    "optimeDate" : ISODate("2018-02-28T22:18:49Z"),
                    "infoMessage" : "could not find member to sync from",
                    "electionTime" : Timestamp(1519856327, 1),
                    "electionDate" : ISODate("2018-02-28T22:18:47Z"),
                    "configVersion" : 1,
                    "self" : true
            },
            {
                    "_id" : 1,
                    "name" : "localhost:27018",
                    "health" : 1,
                    "state" : 2,
                    "stateStr" : "SECONDARY",
                    "uptime" : 15,
                    "optime" : {
                            "ts" : Timestamp(1519856329, 5),
                            "t" : NumberLong(1)
                    },
                    "optimeDurable" : {
                            "ts" : Timestamp(1519856329, 5),
                            "t" : NumberLong(1)
                    },
                    "optimeDate" : ISODate("2018-02-28T22:18:49Z"),
```

```
                        "optimeDurableDate" : ISODate("2018-02-28T22:18:49Z"),
                        "lastHeartbeat" : ISODate("2018-02-28T22:18:51.861Z"),
                        "lastHeartbeatRecv" : ISODate("2018-02-28T22:18:49.565Z"
),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "localhost:27017",
                        "configVersion" : 1
                },
                {
                        "_id" : 3,
                        "name" : "localhost:27019",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 15,
                        "optime" : {
                                "ts" : Timestamp(1519856329, 5),
                                "t" : NumberLong(1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1519856329, 5),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2018-02-28T22:18:49Z"),
                        "optimeDurableDate" : ISODate("2018-02-28T22:18:49Z"),
                        "lastHeartbeat" : ISODate("2018-02-28T22:18:51.860Z"),
                        "lastHeartbeatRecv" : ISODate("2018-02-28T22:18:49.567Z"
),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "localhost:27017",
                        "configVersion" : 1
                }
        ],
        "ok" : 1,
        "operationTime" : Timestamp(1519856329, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1519856329, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rs0:PRIMARY>
```

Figura 4.2. – Iniciación réplica y estado actual miembros.

4.1.2.- Prueba sharding

En este ejemplo se van a utilizar los siguientes elementos [6]:

- Base de datos de configuración **config**: en modo replicaSet con nombre configrs y con dos nodos en el grupo de réplica para el almacenamiento de dicha configuración:
  - Mongod en el puerto 26050.

- Mongod en el puerto 26051.

❖ Shards: instancias para los nodos entre los que se realizará el particionado de datos:

- Primer conjunto: mongod en los puertos 27100 y 27101.
- Segundo conjunto: mongod en los puertos 27200 y 27201.

El proceso que se va a seguir para la prueba consta de los siguientes puntos:

1) Al igual que en la prueba de réplica, se crean las carpetas que servirán para almacenar los datos de cada una de las instancias de Mongo replicada.

2) Se inician los servidores de configuración de la réplica y, acto seguido, se activa la replicación:

```
sydea@sydeaCloneI:~$ mongod -configsvr -replSet configrs -dbpath /data/db/cgr1 -port 26050
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten] MongoDB starting : pid=550 port=26050 dbpath=/data/db/cgr1 64-bit host=sydeaCloneI
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten] db version v3.6.2
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten] modules: none
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten] build environment:
2018-03-10T12:44:46.945+0100 I CONTROL  [initandlisten]     distmod: ubuntu1604
2018-03-10T12:44:46.946+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-03-10T12:44:46.946+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-03-10T12:44:46.946+0100 I CONTROL  [initandlisten] options: { net: { port: 26050 }, replication: { replSet: "configrs" }, sharding: { clusterRole: "configsvr" }, s
torage: { dbPath: "/data/db/cgr1" } }
```

```
sydea@sydeaCloneI:~$ mongod -configsvr -replSet configrs -dbpath /data/db/cgr2 -port 26051
2018-03-10T12:04:53.497+0100 I CONTROL  [initandlisten] MongoDB starting : pid=27477 port=26051 dbpath=/data/db/cgr2 64-bit host=sydeaCloneI
2018-03-10T12:04:53.497+0100 I CONTROL  [initandlisten] db version v3.6.2
2018-03-10T12:04:53.497+0100 I CONTROL  [initandlisten] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:04:53.497+0100 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:04:53.497+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-03-10T12:04:53.497+0100 I CONTROL  [initandlisten] modules: none
2018-03-10T12:04:53.498+0100 I CONTROL  [initandlisten] build environment:
2018-03-10T12:04:53.498+0100 I CONTROL  [initandlisten]     distmod: ubuntu1604
2018-03-10T12:04:53.498+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-03-10T12:04:53.498+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-03-10T12:04:53.498+0100 I CONTROL  [initandlisten] options: { net: { port: 26051 }, replication: { replSet: "configrs" }, sharding: { clusterRole: "configsvr" }, s
torage: { dbPath: "/data/db/cgr2" } }
```

Figura 4.3. – Inicio servidores de configuración sharding.

El parámetro "*configsvr*" indica que se trata de instancias de configuración. Asimismo, el parámetro "*replSet configrs*" indica que estas instancias están asociadas al conjunto de replica "*configrs*".

```
sydea@sydeaCloneI:~$ mongo -port 26050
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:26050/
MongoDB server version: 3.6.2
```

Figura 4.4. – Inicio consola mongos puerto 26050.

```
> conf = {
... _id: "configrs",
...
... members: [
... {
... _id: 0,
... host: 'localhost:26050'
... },
... {
... _id: 1,
... host: 'localhost:26051'
... }
... ]
... };
{
        "_id" : "configrs",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:26050"
                },
                {
                        "_id" : 1,
                        "host" : "localhost:26051"
                }
        ]
}
```

Figura 4.5. – Definición variable conf miembros sharding.

```
sydea@sydeaCloneI:~$ cat /etc/hosts
127.0.0.1       localhost
172.16.2.154    sydeaCloneI
172.16.2.254    sydeaCloneII
172.16.2.253    sydeaCloneIII
```

Figura 4.6. – Hosts definidos en cada servidor.

```
> rs.initiate(conf);
{
        "ok" : 1,
        "operationTime" : Timestamp(1520531672, 1),
        "$gleStats" : {
                "lastOpTime" : Timestamp(1520531672, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520531672, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
configrs:SECONDARY>
```

Figura 4.7. – Inicio configuración sharding.

```
configrs:PRIMARY> rs.status();
{
        "set" : "configrs",
        "date" : ISODate("2018-03-08T21:56:46.642Z"),
        "myState" : 1,
        "term" : NumberLong(2),
        "configsvr" : true,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1520546200, 1),
                        "t" : NumberLong(2)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1520546200, 1),
                        "t" : NumberLong(2)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1520546200, 1),
                        "t" : NumberLong(2)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1520546200, 1),
                        "t" : NumberLong(2)
                }
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "localhost:26050",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 50,
                        "optime" : {
                                "ts" : Timestamp(1520546200, 1),
                                "t" : NumberLong(2)
                        },
                        "optimeDate" : ISODate("2018-03-08T21:56:40Z"),
                        "infoMessage" : "could not find member to sync from",
                        "electionTime" : Timestamp(1520546178, 1),
                        "electionDate" : ISODate("2018-03-08T21:56:18Z"),
                        "configVersion" : 1,
                        "self" : true
```

```
                {
                        "_id" : 1,
                        "name" : "localhost:26051",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 34,
                        "optime" : {
                                "ts" : Timestamp(1520546200, 1),
                                "t" : NumberLong(2)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1520546200, 1),
                                "t" : NumberLong(2)
                        },
                        "optimeDate" : ISODate("2018-03-08T21:56:40Z"),
                        "optimeDurableDate" : ISODate("2018-03-08T21:56:40Z"),
                        "lastHeartbeat" : ISODate("2018-03-08T21:56:44.735Z"),
                        "lastHeartbeatRecv" : ISODate("2018-03-08T21:56:45.589Z"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "localhost:26050",
                        "configVersion" : 1
                }
        ],
        "ok" : 1,
        "operationTime" : Timestamp(1520546200, 1),
        "$gleStats" : {
                "lastOpTime" : Timestamp(0, 0),
                "electionId" : ObjectId("7fffffff0000000000000002")
        },
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520546200, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Figura 4.8. – Estado sharding con miembros.

Se puede ver de nuevo cómo se ha iniciado correctamente la replicación con los dos miembros configurados en los puertos 26050 y 26051, primario y secundario, respectivamente.

3) Por otro lado, se inician los shards de los servidores de datos replicados en los puertos 27100, 27101 (primer conjunto), 27200 y 27201 (segundo conjunto):

```
sydea@sydeaCloneI:~$ mongod -shardsvr -replSet p1 -dbpath /data/db/p1_1 -port 27100
2018-03-10T12:53:26.770+0100 I CONTROL  [initandlisten] MongoDB starting : pid=1519 port=27100 dbpath=/data/db/p1_1 64-bit host=sydeaCloneI
2018-03-10T12:53:26.770+0100 I CONTROL  [initandlisten] db version v3.6.2
2018-03-10T12:53:26.770+0100 I CONTROL  [initandlisten] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:53:26.770+0100 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:53:26.770+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-03-10T12:53:26.771+0100 I CONTROL  [initandlisten] modules: none
2018-03-10T12:53:26.771+0100 I CONTROL  [initandlisten] build environment:
2018-03-10T12:53:26.771+0100 I CONTROL  [initandlisten]     distmod: ubuntu1604
2018-03-10T12:53:26.771+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-03-10T12:53:26.771+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-03-10T12:53:26.771+0100 I CONTROL  [initandlisten] options: { net: { port: 27100 }, replication: { replSet: "p1" }, sharding: { clusterRole: "shardsvr" }, storage:
 { dbPath: "/data/db/p1_1" } }
```

Figura 4.9. – Inicio shard puerto 27100.

```
sydea@sydeaCloneI:~$ mongod -shardsvr -replSet p1 -dbpath /data/db/p1_2 -port 27101
2018-03-10T12:54:21.791+0100 I CONTROL  [initandlisten] MongoDB starting : pid=1628 port=27101 dbpath=/data/db/p1_2 64-bit host=sydeaCloneI
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten] db version v3.6.2
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten] modules: none
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten] build environment:
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten]     distmod: ubuntu1604
2018-03-10T12:54:21.792+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-03-10T12:54:21.793+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-03-10T12:54:21.793+0100 I CONTROL  [initandlisten] options: { net: { port: 27101 }, replication: { replSet: "p1" }, sharding: { clusterRole: "shardsvr" }, storage:
 { dbPath: "/data/db/p1_2" } }
```

Figura 4.10. – Inicio shard puerto 27101.

```
sydea@sydeaCloneI:~$ mongod -shardsvr -replSet p2 -dbpath /data/db/p2_1 -port 27200
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] MongoDB starting : pid=28619 port=27200 dbpath=/data/db/p2_1 64-bit host=sydeaCloneI
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] db version v3.6.2
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] modules: none
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] build environment:
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten]     distmod: ubuntu1604
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-03-10T12:09:57.418+0100 I CONTROL  [initandlisten] options: { net: { port: 27200 }, replication: { replSet: "p2" }, sharding: { clusterRole: "shardsvr" }, storage:
 { dbPath: "/data/db/p2_1" } }
```

Figura 4.11. – Inicio shard puerto 27200.

```
sydea@sydeaCloneI:~$ mongod -shardsvr -replSet p2 -dbpath /data/db/p2_2 -port 27201
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] MongoDB starting : pid=1762 port=27201 dbpath=/data/db/p2_2 64-bit host=sydeaCloneI
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] db version v3.6.2
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] modules: none
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] build environment:
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten]     distmod: ubuntu1604
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-03-10T12:55:30.568+0100 I CONTROL  [initandlisten] options: { net: { port: 27201 }, replication: { replSet: "p2" }, sharding: { clusterRole: "shardsvr" }, storage:
 { dbPath: "/data/db/p2_2" } }
```

Figura 4.12. – Inicio shard puerto 27201.

4) Se activan los conjuntos de réplica:

```
sydea@sydeaCloneI:~$ mongo -port 27100
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27100/
MongoDB server version: 3.6.2
```

Figura 4.13. – Inicio consola mongos puerto 27100.

```
> var config = {
... _id: "pl",
...
... members: [
... {
... _id: 0,
... host: 'localhost:27100'
... },
... {
... _id: 1,
... host: 'localhost:27101'
... }
... ]
... };
> rs.initiate(config);
{ "ok" : 1 }
```

Figura 4.14. – Definición variable conf sharding.

```
p1:SECONDARY> rs.status();
{
        "set" : "p1",
        "date" : ISODate("2018-03-08T18:03:26.724Z"),
        "myState" : 2,
        "term" : NumberLong(0),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(0, 0),
                        "t" : NumberLong(-1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1520532199, 1),
                        "t" : NumberLong(-1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1520532199, 1),
                        "t" : NumberLong(-1)
                }
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "localhost:27100",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 154,
                        "optime" : {
                                "ts" : Timestamp(1520532199, 1),
                                "t" : NumberLong(-1)
                        },
                        "optimeDate" : ISODate("2018-03-08T18:03:19Z"),
                        "infoMessage" : "could not find member to sync from",
                        "configVersion" : 1,
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "localhost:27101",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
```

```
                        "uptime" : 7,
                        "optime" : {
                                "ts" : Timestamp(1520532199, 1),
                                "t" : NumberLong(-1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1520532199, 1),
                                "t" : NumberLong(-1)
                        },
                        "optimeDate" : ISODate("2018-03-08T18:03:19Z"),
                        "optimeDurableDate" : ISODate("2018-03-08T18:03:19Z"),
                        "lastHeartbeat" : ISODate("2018-03-08T18:03:24.402Z"),
                        "lastHeartbeatRecv" : ISODate("2018-03-08T18:03:26.498Z"),
                        "pingMs" : NumberLong(0),
                        "configVersion" : 1
                }
        ],
        "ok" : 1
}
```

Figura 4.15. – Estado sharding tras activación.

```
sydea@sydeaCloneI:~$ mongo -port 27200
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27200/
MongoDB server version: 3.6.2
```

Figura 4.16. – Inicio consola mongos puerto 27200.

```
> config = {
... _id: "p2",
...
... members: [
... {
... _id: 0,
... host: 'localhost:27200'
... },
... {
... _id: 1,
... host: 'localhost:27201'
... }
... ]
... };
{
        "_id" : "p2",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:27200"
                },
                {
                        "_id" : 1,
                        "host" : "localhost:27201"
                }
        ]
}
> rs.initiate(config);
{ "ok" : 1 }
```

Figura 4.17. – Definición variable conf sharing puerto 27200.

5) Se crean los procesos mongos, que son los que hacen de balanceadores para distribuir las queries entre el conjunto.

```
sydea@sydeaCloneI:~$ mongos -configdb configrs/127.0.0.1:26050,127.0.0.1:26051
2018-03-08T19:09:40.466+0100 W SHARDING [main] Running a sharded cluster with fewer than 3 config servers should only be done for testing purposes and is not recommende
d for production.
```

```
2018-03-08T19:09:40.474+0100 I SHARDING [mongosMain] mongos version v3.6.2
2018-03-08T19:09:40.474+0100 I CONTROL  [mongosMain] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-08T19:09:40.474+0100 I CONTROL  [mongosMain] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-08T19:09:40.474+0100 I CONTROL  [mongosMain] allocator: tcmalloc
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] modules: none
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] build environment:
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain]     distmod: ubuntu1604
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain]     distarch: x86_64
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain]     target_arch: x86_64
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] db version v3.6.2
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] allocator: tcmalloc
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] modules: none
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] build environment:
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain]     distmod: ubuntu1604
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain]     distarch: x86_64
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain]     target_arch: x86_64
2018-03-08T19:09:40.475+0100 I CONTROL  [mongosMain] options: { sharding: { configDB: "configrs/127.0.0.1:26050,127.0.0.1:26051" } }
2018-03-08T19:09:40.477+0100 I NETWORK  [mongosMain] Starting new replica set monitor for configrs/127.0.0.1:26050,127.0.0.1:26051
2018-03-08T19:09:40.477+0100 I SHARDING [thread1] creating distributed lock ping thread for process sydeaCloneI:27017:1520532580:7393906634938292677 (sleeping for 30000
ms)
```

Figura 4.18. – Inicio proceso mongos balanceadores.

```
sydea@sydeaCloneI:~$ mongos -configdb configrs/127.0.0.1:26050,127.0.0.1:26051 -port 26601
2018-03-10T12:12:34.488+0100 W SHARDING [main] Running a sharded cluster with fewer than 3 config servers should only be done for testing purposes and is not recommende
d for production.
2018-03-10T12:12:34.495+0100 I CONTROL  [main]
2018-03-10T12:12:34.495+0100 I CONTROL  [main] ** WARNING: Access control is not enabled for the database.
2018-03-10T12:12:34.495+0100 I CONTROL  [main] **          Read and write access to data and configuration is unrestricted.
2018-03-10T12:12:34.495+0100 I CONTROL  [main]
2018-03-10T12:12:34.495+0100 I CONTROL  [main] ** WARNING: This server is bound to localhost.
2018-03-10T12:12:34.495+0100 I CONTROL  [main] **          Remote systems will be unable to connect to this server.
2018-03-10T12:12:34.495+0100 I CONTROL  [main] **          Start the server with --bind_ip <address> to specify which IP
2018-03-10T12:12:34.495+0100 I CONTROL  [main] **          addresses it should serve responses from, or with --bind_ip_all to
2018-03-10T12:12:34.495+0100 I CONTROL  [main] **          bind to all interfaces. If this behavior is desired, start the
2018-03-10T12:12:34.495+0100 I CONTROL  [main] **          server with --bind_ip 127.0.0.1 to disable this warning.
2018-03-10T12:12:34.495+0100 I CONTROL  [main]
2018-03-10T12:12:34.495+0100 I SHARDING [mongosMain] mongos version v3.6.2
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] allocator: tcmalloc
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] modules: none
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] build environment:
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain]     distmod: ubuntu1604
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain]     distarch: x86_64
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain]     target_arch: x86_64
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] db version v3.6.2
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] git version: 489d177dbd0f0420a8ca04d39fd78d0a2c539420
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] allocator: tcmalloc
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] modules: none
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] build environment:
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain]     distmod: ubuntu1604
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain]     distarch: x86_64
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain]     target_arch: x86_64
2018-03-10T12:12:34.496+0100 I CONTROL  [mongosMain] options: { net: { port: 26601 }, sharding: { configDB: "configrs/127.0.0.1:26050,127.0.0.1:26051" } }
2018-03-10T12:12:34.498+0100 I NETWORK  [mongosMain] Starting new replica set monitor for configrs/127.0.0.1:26050,127.0.0.1:26051
2018-03-10T12:12:34.498+0100 I SHARDING [thread1] creating distributed lock ping thread for process sydeaCloneI:26601:1520680354:747151021379981901 8 (sleeping for 30000
ms)
```

Figura 4.19. – Inicio proceso mongos balanceadores puerto 26601.

Con el parámetro "*configdb*" se indican los servidores de configuración, seguido de las direcciones y puertos de estos.

6) Se ponen en marcha las particiones y se indican los maestros de cada conjunto de réplica de las instancias de datos:

```
sydea@sydeaCloneI:~$ mongo
MongoDB shell version v3.6.2
connecting to: mongodb://127.0.0.1:27017
```

Figura 4.20. – Inicio mongo.

```
mongos> sh.addShard("p1/"+'localhost:27100');
{
        "shardAdded" : "p1",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520532739, 6),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520532739, 6)
}
mongos> sh.addShard("p2/"+'localhost:27200');
{
        "shardAdded" : "p2",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520532752, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520532752, 5)
}
```

Figura 4.21. – Indicar maestros de cada conjunto de réplicas.

```
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5aa178e4239bef0168a07573")
  }
  shards:
        {  "_id" : "p1",  "host" : "p1/localhost:27100,localhost:27101",  "state" : 1 }
        {  "_id" : "p2",  "host" : "p2/localhost:27200,localhost:27201",  "state" : 1 }
  active mongoses:
        "3.6.2" : 2
  autosplit:
        Currently enabled: yes
  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" : "config",  "primary" : "config",  "partitioned" : true }
```

Figura 4.22. – Estado ya con los shards.

7) Una vez ya está todo inicializado, se prueba a habilitar el sharding para una determinada base de datos (DB). Para ello, se crea la DB *shipsdb* y se habilita para sharding.

```
mongos> use shipsdb
switched to db shipsdb
mongos> sh.enableSharding("shipsdb")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520532799, 9),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520532799, 9)
}
```

Figura 4.23. – Habilitar sharding para shipsdb.

```
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5aa178e4239bef0168a07573")
  }
  shards:
        {  "_id" : "p1",  "host" : "p1/localhost:27100,localhost:27101",  "state" : 1 }
        {  "_id" : "p2",  "host" : "p2/localhost:27200,localhost:27201",  "state" : 1 }
  active mongoses:
        "3.6.2" : 2
  autosplit:
        Currently enabled: yes
  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" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "shipsdb",  "primary" : "p2",  "partitioned" : true }
```

Figura 4.24. – Estado tras activar sharding.

Ahora el sharding ya está activado (*balancer: Currently enabled: yes*).

8) Se prueba el sharding para una determinada colección, creando una determinada colección e indicando que puede partir los valores utilizando la clave "*_id*". Esto se consigue con la función *shardCollection()* especificando la colección "*ships*" y la clave mencionada.

```
mongos> sh.shardCollection("shipsdb.ships",{_id:1},true)
{
        "collectionsharded" : "shipsdb.ships",
        "collectionUUID" : UUID("c49b1028-82ce-45ff-8740-6af7c9e1f8a5"),
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520532930, 12),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520532930, 12)
}
```

Figura 4.25. – Clave sharding para colección ships.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5aa178e4239bef0168a07573")
  }
  shards:
        {  "_id" : "p1",  "host" : "p1/localhost:27100,localhost:27101",  "state" : 1 }
        {  "_id" : "p2",  "host" : "p2/localhost:27200,localhost:27201",  "state" : 1 }
  active mongoses:
        "3.6.2" : 2
  autosplit:
        Currently enabled: yes
  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" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "shipsdb",  "primary" : "p2",  "partitioned" : true }
                shipsdb.ships
                        shard key: { "_id" : 1 }
                        unique: true
                        balancing: true
                        chunks:
                                p2      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
```

Figura 4.26. – Estado sharding tras añadir shipsdb.

Si se vuelve a comprobar el estado del grupo de particionado, se puede ver que el "*p2*" está en modo particionado (*partitioned: true*) pero el *balancer* (que es quien moverá los datos) no se ha llegado a ejecutar. Para conseguir que comience a ejecutarse se utiliza la función *setBalancerState(boolean).*

Se insertan valores en la DB:

```
mongos> db.ships.insert({name:'USS Enterprise-D',operator:'Starfleet',type:'Explorer',class:'Galaxy',crew:750,codes:[10,11,12]});
WriteResult({ "nInserted" : 1 })
mongos> db.ships.insert({name:'USS Prometheus',operator:'Starfleet',class:'Prometheus',crew:4,codes:[1,14,17]});
WriteResult({ "nInserted" : 1 })
mongos> db.ships.insert({name:'USS Defiant',operator:'Starfleet',class:'Defiant',crew:50,codes:[10,17,19]});
WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.ships.insert({name:'IKS Buruk',operator:' Klingon Empire',class:'Warship',crew:40,codes:[100,110,120]});
WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.ships.insert({name:'IKS Somraw',operator:' Klingon Empire',class:'Raptor',crew:50,codes:[101,111,120]});
WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.ships.insert({name:'Scimitar',operator:'Romulan Star Empire',type:'Warbird',class:'Warbird',crew:25,codes:[201,211,220]});
WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.ships.insert({name:'Narada',operator:'Romulan Star Empire',type:'Warbird',class:'Warbird',crew:65,codes:[251,251,220]});
WriteResult({ "nInserted" : 1 })
```

Figura 4.27. – Inserción de valores en la DB.

9) Se repite lo mismo, pero con otro conjunto de datos, de forma que se puede comprobar cómo se distribuyen los datos entre los shards:

```
mongos> use piedras
switched to db piedras
mongos> sh.enableSharding("piedras")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520533013, 6),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520533013, 6)
}
```

Figura 4.28. – Activar sharding para la colección piedras.

```
mongos> sh.shardCollection("piedras.minerales",{_id:1},true)
{
        "collectionsharded" : "piedras.minerales",
        "collectionUUID" : UUID("dc2ebd36-ca5a-448e-9bfd-7e4ebbda1fbc"),
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520533035, 18),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520533035, 18)
}
```

Figura 4.29. – Clave sharding base de datos minerales.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5aa178e4239bef0168a07573")
  }
  shards:
        {  "_id" : "p1",  "host" : "p1/localhost:27100,localhost:27101",  "state" : 1 }
        {  "_id" : "p2",  "host" : "p2/localhost:27200,localhost:27201",  "state" : 1 }
  active mongoses:
        "3.6.2" : 2
  autosplit:
        Currently enabled: yes
  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" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "piedras",  "primary" : "p1",  "partitioned" : true }
                piedras.minerales
                        shard key: { "_id" : 1 }
                        unique: true
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "shipsdb",  "primary" : "p2",  "partitioned" : true }
                shipsdb.ships
                        shard key: { "_id" : 1 }
                        unique: true
                        balancing: true
                        chunks:
```

```
                                p2      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
        {  "_id" : "shipsdb4",  "primary" : "p1",  "partitioned" : false }
```

Figura 4.30. – Estado tras añadir DB piedras.

```
mongos> db.minerales.insert({_id:"Calcita", color:"Jaspeada", brillo:"opaco", dureza:3, textura:"Arcillosa"})
WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.minerales.insert({_id:"Plata", color:"Plateado",brillo:"Metálico", dureza:2.7, textura:"Lisa"})
WriteResult({ "nInserted" : 1 })
mongos> db.minerales.insert({_id:"Cuarzo", color:"Incoloro",brillo:"Vítreo", dureza:7, textura:"Lisa"})
db.minerales.insert({_id:"Grafito",color:"Gris",brillo:"Submetálico", dureza:1, textura:"Granulosa"})WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.minerales.insert({_id:"Yeso", color:"Jaspeado", brillo:"Vítreo",dureza:[1.5,2],textura:"Arcillosa"})
WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.minerales.insert({_id:"Halita", color:"Blanco",brillo:"Vítreo", dureza:2.5, textura:"Granulosa"})
WriteResult({ "nInserted" : 1 })
mongos>
mongos> db.minerales.insert({_id:"Grafito",color:"Gris",brillo:"Submetálico", dureza:1, textura:"Granulosa"})
WriteResult({ "nInserted" : 1 })
```

Figura 4.31. – Inserción datos DB piedras.

Una vez realizado todo esto, se puede comprobar el número de chunks y cómo se han distribuido los datos entre los dos shards.

Todo esto teniendo en cuenta que la distribución se hace en función del tamaño de los chunks, 64 MB por defecto.

Ahora se va a probar a modificar este tamaño de los chunks haciendo inserciones de volúmenes grandes de datos. Para ello, se aprovecha del poder de MongoDB de ejecutar scripts Javascrip.

10) Se habilita el sharding para la DB *shardTestDB* y la colección *users* dentro de ella. En este caso se va a utilizar como índice para el sharding el username.

```
mongos> sh.enableSharding("shardTestDB")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520546534, 6),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520546534, 6)
}
```

Figura 4.32. – Activar sharding para shardTestDB.

```
mongos> use shardTestDB
switched to db shardTestDB
mongos> db.users.getIndexes()
[ ]
mongos> db.users.ensureIndex({username:1})
{
        "raw" : {
                "p2/localhost:27200,localhost:27201" : {
                        "createdCollectionAutomatically" : true,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "ok" : 1
                }
        },
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520546572, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520546572, 1)
}
mongos> sh.shardCollection("shardTestDB.users",{username:1})
{
        "collectionsharded" : "shardTestDB.users",
        "collectionUUID" : UUID("578eca88-ffa0-4504-a012-f2dffcbbbe2d"),
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520546687, 13),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520546687, 13)
}
```

Figura 4.33. – Clave sharding shardTestDB.

11) Se cambia el tamaño del chunk a 1 MB para ver así si se consigue una distribución simétrica entre los nodos.

```
mongos> use config
switched to db config
mongos> db.settings.find( { "_id" : "chunksize" } )
mongos> db.settings.save({ "_id" : "chunksize", value : 1 })
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : "chunksize" })
```

Figura 4.34. – Cambio tamaño chunk.

12) Una vez configurado el tamaño del chunk, se comprueba si los datos se distribuyen sin que se hayan balanceado inicialmente. Para ello, primero se

para el balanceador, luego se insertan los datos y ya se inicia de nuevo el balanceador.

```
mongos> db.settings.find({"_id" : "chunksize"})
{ "_id" : "chunksize", "value" : 1 }
mongos> sh.stopBalancer()
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520546990, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520546990, 1)
}
mongos> sh.getBalancerState()
false
```

Figura 4.35. – Parar el balanceador.

```
mongos> use shardTestDB2
switched to db shardTestDB2
mongos> for (var i=0; i<100000; i++) {db.users.insert({"username" : "user"+i, "created at" : new Date()});}

WriteResult({ "nInserted" : 1 })
```

Figura 4.36. – Inserción datos shardTestDB.

```
mongos> sh.status({verbose : 1})
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5aa178e4239bef0168a07573")
  }
  shards:
        {  "_id" : "p1",  "host" : "p1/localhost:27100,localhost:27101",  "state" : 1 }
        {  "_id" : "p2",  "host" : "p2/localhost:27200,localhost:27201",  "state" : 1 }
  active mongoses:
        {  "_id" : "sydeaCloneI:26601",  "mongoVersion" : "3.6.2",  "ping" : ISODate("2018-03-08T22:16:28.698Z"),  "up" : NumberLong(1020),  "waiting" : true }
        {  "_id" : "sydeaCloneI:27017",  "mongoVersion" : "3.6.2",  "ping" : ISODate("2018-03-08T22:16:28.543Z"),  "up" : NumberLong(1030),  "waiting" : true }
  autosplit:
        Currently enabled: yes
```

```
  balancer:
        Currently enabled:  no
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  2
        Last reported error:  Could not find host matching read preference { mode: "primary" } for set p1
        Time of Reported error:  Thu Mar 08 2018 22:56:50 GMT+0100 (CET)
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "piedras",  "primary" : "p1",  "partitioned" : true }
                piedras.minerales
                        shard key: { "_id" : 1 }
                        unique: true
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "shardTestDB",  "primary" : "p2",  "partitioned" : true }
                shardTestDB.users
                        shard key: { "username" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p2      1
                        { "username" : { "$minKey" : 1 } } -->> { "username" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
        {  "_id" : "shardTestDB2",  "primary" : "p1",  "partitioned" : false }
        {  "_id" : "shipsdb",  "primary" : "p2",  "partitioned" : true }
                shipsdb.ships
                        shard key: { "_id" : 1 }
                        unique: true
                        balancing: true
                        chunks:
                                p2      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
        {  "_id" : "shipsdb4",  "primary" : "p1",  "partitioned" : false }
        {  "_id" : "test",  "primary" : "p2",  "partitioned" : false }
```

Figura 4.37. – Estado tras la inserción.

```
mongos> sh.startBalancer()
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520547468, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1520547468, 1)
}
```

Figura 4.38. – Arrancar el balanceador.

```
mongos> sh.status({ verbose : 1 })
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5aa178e4239bef0168a07573")
  }
  shards:
        {  "_id" : "p1",  "host" : "p1/localhost:27100,localhost:27101",  "state" : 1 }
        {  "_id" : "p2",  "host" : "p2/localhost:27200,localhost:27201",  "state" : 1 }
  active mongoses:
        {  "_id" : "sydeaCloneI:26601",  "mongoVersion" : "3.6.2",  "ping" : ISODate("2018-03-08T22:17:48.749Z"),  "up" : NumberLong(1100),  "waiting" : true }
        {  "_id" : "sydeaCloneI:27017",  "mongoVersion" : "3.6.2",  "ping" : ISODate("2018-03-08T22:17:48.601Z"),  "up" : NumberLong(1110),  "waiting" : true }
  autosplit:
        Currently enabled: yes
```

```
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  2
        Last reported error:  Could not find host matching read preference { mode: "primary" } for set p1
        Time of Reported error:  Thu Mar 08 2018 22:56:50 GMT+0100 (CET)
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "piedras",  "primary" : "p1",  "partitioned" : true }
                piedras.minerales
                        shard key: { "_id" : 1 }
                        unique: true
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
        {  "_id" : "shardTestDB",  "primary" : "p2",  "partitioned" : true }
                shardTestDB.users
                        shard key: { "username" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p2      1
                        { "username" : { "$minKey" : 1 } } -->> { "username" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
        {  "_id" : "shardTestDB2",  "primary" : "p1",  "partitioned" : false }
        {  "_id" : "shipsdb",  "primary" : "p2",  "partitioned" : true }
                shipsdb.ships
                        shard key: { "_id" : 1 }
                        unique: true
                        balancing: true
                        chunks:
                                p2      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
        {  "_id" : "shipsdb4",  "primary" : "p1",  "partitioned" : false }
        {  "_id" : "test",  "primary" : "p2",  "partitioned" : false }
```

Figura 4.39. – Estado activado balanceador.

13) Finalmente, se prueba la inserción de datos con el balanceador ya activo.

```
mongos> for (var i=100000; i<110000; i++) {db.users.insert({"username" : "user"+i, "created at" : new Date()});}
WriteResult({ "nInserted" : 1 })
```

Figura 4.40. – Insercción datos shardTestDB.

```
mongos> sh.status({ verbose : 1 })
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5aa178e4239bef0168a07573")
  }
  shards:
        {  "_id" : "p1",  "host" : "p1/localhost:27100,localhost:27101",  "state" : 1 }
        {  "_id" : "p2",  "host" : "p2/localhost:27200,localhost:27201",  "state" : 1 }
  active mongoses:
        {  "_id" : "sydeaCloneI:26601",  "mongoVersion" : "3.6.2",  "ping" : ISODate("2018-03-31T09:56:50.256Z"),  "up" : NumberLong(1411),  "waiting" : true }
        {  "_id" : "sydeaCloneI:27017",  "mongoVersion" : "3.6.2",  "ping" : ISODate("2018-03-31T09:56:45.240Z"),  "up" : NumberLong(1411),  "waiting" : true }
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  1
        Last reported error:  Could not find host matching read preference { mode: "primary" } for set p2
        Time of Reported error:  Sat Mar 31 2018 11:30:10 GMT+0200 (CEST)
        Migration Results for the last 24 hours:
                4 : Success
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                p1      1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
```

```
  {  "_id" : "piedras",  "primary" : "p1",  "partitioned" : true }
        piedras.minerales
                shard key: { "_id" : 1 }
                unique: true
                balancing: true
                chunks:
                        p1      1
                { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 0)
  {  "_id" : "shardTestDB",  "primary" : "p2",  "partitioned" : true }
        shardTestDB.users
                shard key: { "username" : 1 }
                unique: false
                balancing: true
                chunks:
                        p2      1
                { "username" : { "$minKey" : 1 } } -->> { "username" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
  {  "_id" : "shardTestDB2",  "primary" : "p1",  "partitioned" : true }
        shardTestDB2.users
                shard key: { "username" : 1 }
                unique: false
                balancing: true
                chunks:
                        p1      10
                        p2      4
                { "username" : { "$minKey" : 1 } } -->> { "username" : "user107255" } on : p2 Timestamp(2, 0)
                { "username" : "user107255" } -->> { "username" : "user15515" } on : p2 Timestamp(3, 0)
                { "username" : "user15515" } -->> { "username" : "user22775" } on : p2 Timestamp(4, 0)
                { "username" : "user22775" } -->> { "username" : "user30033" } on : p2 Timestamp(5, 0)
                { "username" : "user30033" } -->> { "username" : "user37294" } on : p1 Timestamp(5, 1)
                { "username" : "user37294" } -->> { "username" : "user44553" } on : p1 Timestamp(1, 5)
                { "username" : "user44553" } -->> { "username" : "user51812" } on : p1 Timestamp(1, 6)
                { "username" : "user51812" } -->> { "username" : "user59072" } on : p1 Timestamp(1, 7)
                { "username" : "user59072" } -->> { "username" : "user66331" } on : p1 Timestamp(1, 8)
                { "username" : "user66331" } -->> { "username" : "user73591" } on : p1 Timestamp(1, 9)
                { "username" : "user73591" } -->> { "username" : "user80850" } on : p1 Timestamp(1, 10)
                { "username" : "user80850" } -->> { "username" : "user8811" } on : p1 Timestamp(1, 11)
                { "username" : "user8811" } -->> { "username" : "user9537" } on : p1 Timestamp(1, 12)
                { "username" : "user9537" } -->> { "username" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 13)
{  "_id" : "shipsdb",  "primary" : "p2",  "partitioned" : true }
        shipsdb.ships
                shard key: { "_id" : 1 }
                unique: true
                balancing: true
                chunks:
                        p2      1
                { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p2 Timestamp(1, 0)
{  "_id" : "shipsdb4",  "primary" : "p1",  "partitioned" : false }
{  "_id" : "test",  "primary" : "p2",  "partitioned" : false }
```

Figura 4.41. – Estado tras inserccción datos.

```
mongos> db.users.getShardDistribution()

Shard p1 at p1/localhost:27100,localhost:27101
 data : 6.72MiB docs : 107078 chunks : 7
 estimated data per chunk : 984KiB
 estimated docs per chunk : 15296

Shard p2 at p2/localhost:27200,localhost:27201
 data : 7.11MiB docs : 112922 chunks : 7
 estimated data per chunk : 1.01MiB
 estimated docs per chunk : 16131

Totals
 data : 13.84MiB docs : 220000 chunks : 14
 Shard p1 contains 48.59% data, 48.67% docs in cluster, avg obj size on shard : 65B
 Shard p2 contains 51.4% data, 51.32% docs in cluster, avg obj size on shard : 66B
```

Figura 4.42. – Distribución datos sharding.

Se espera un tiempo y se vuelve a comprobar el estado:

```
{  "_id" : "shardTestDB2",  "primary" : "p1",  "partitioned" : true }
        shardTestDB2.users
                shard key: { "username" : 1 }
                unique: false
                balancing: true
                chunks:
                        p1      7
                        p2      7
                { "username" : { "$minKey" : 1 } } -->> { "username" : "user107255" } on : p2 Timestamp(2, 0)
                { "username" : "user107255" } -->> { "username" : "user15515" } on : p2 Timestamp(3, 0)
                { "username" : "user15515" } -->> { "username" : "user22775" } on : p2 Timestamp(4, 0)
                { "username" : "user22775" } -->> { "username" : "user30033" } on : p2 Timestamp(5, 0)
                { "username" : "user30033" } -->> { "username" : "user37294" } on : p2 Timestamp(6, 0)
                { "username" : "user37294" } -->> { "username" : "user44553" } on : p2 Timestamp(7, 0)
                { "username" : "user44553" } -->> { "username" : "user51812" } on : p2 Timestamp(8, 0)
                { "username" : "user51812" } -->> { "username" : "user59072" } on : p1 Timestamp(8, 1)
                { "username" : "user59072" } -->> { "username" : "user66331" } on : p1 Timestamp(1, 8)
                { "username" : "user66331" } -->> { "username" : "user73591" } on : p1 Timestamp(1, 9)
                { "username" : "user73591" } -->> { "username" : "user80850" } on : p1 Timestamp(1, 10)
                { "username" : "user80850" } -->> { "username" : "user8811" } on : p1 Timestamp(1, 11)
                { "username" : "user8811" } -->> { "username" : "user9537" } on : p1 Timestamp(1, 12)
                { "username" : "user9537" } -->> { "username" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 13)
```

Figura 4.43. – Estado tras un tiempo inserccción datos.

Como se puede observar en las últimas imágenes, los datos insertados se distribuyen correctamente aun con el balanceador parado inicialmente.

4.2.- Comportamiento shards

Una vez realizadas la configuración en MongoDB y viendo cómo se comporta con replicación y sharding, se lleva a cabo otra prueba más. En este caso ya con la configuración final, se prueba a insertar datos en unas colecciones determinadas para, posteriormente, eliminar uno de los shards configurados y ver cómo se comporta [7].

```
mongos> sh.getBalancerState()
true
mongos> db.adminCommand( { listShards: 1 } )
{
        "shards" : [
                {
                        "_id" : "p1",
                        "host" : "p1/sydeaCloneII:27100,sydeaCloneII:27101",
                        "state" : 1
                },
                {
                        "_id" : "p2",
                        "host" : "p2/sydeaCloneIII:27200,sydeaCloneIII:27201",
                        "state" : 1
                }
        ],
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1524084226, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1524084226, 1)
}
```

Figura 4.44. – Lista shards.

```
mongos> db.adminCommand( { removeShard: "p2" } )
{
        "msg" : "draining started successfully",
        "state" : "started",
        "shard" : "p2",
        "note" : "you need to drop or movePrimary these databases",
        "dbsToMove" : [
                "shipsdb",
                "shardTestDB2"
        ],
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1524084306, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1524084306, 2)
}
```

Figura 4.45. – Eliminación sharding.

```
mongos> db.adminCommand( { removeShard: "p2" } )
{
        "msg" : "draining ongoing",
        "state" : "ongoing",
        "remaining" : {
                "chunks" : NumberLong(0),
                "dbs" : NumberLong(2)
        },
        "note" : "you need to drop or movePrimary these databases",
        "dbsToMove" : [
                "shipsdb",
                "shardTestDB2"
        ],
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1524084382, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1524084382, 1)
}
```

Figura 4.46. – Estado eliminación shard.

```
    {  "_id" : "piedras",  "primary" : "p1",  "partitioned" : true }
            piedras.minerales
                    shard key: { "_id" : 1 }
                    unique: true
                    balancing: true
                    chunks:
                            p1      13
                    { "_id" : { "$minKey" : 1 } } -->> { "_id" : "Cuarzo" } on : p1 Timestamp(8, 0)
                    { "_id" : "Cuarzo" } -->> { "_id" : ObjectId("5ad50d4769661a3e88d33e59") } on : p1 Timestamp(9, 0)
                    { "_id" : ObjectId("5ad50d4769661a3e88d33e59") } -->> { "_id" : ObjectId("5ad50d5369661a3e88d35c77") } on : p1 Timestamp(10, 0)
                    { "_id" : ObjectId("5ad50d5369661a3e88d35c77") } -->> { "_id" : ObjectId("5ad50d6269661a3e88d382f8") } on : p1 Timestamp(11, 0)
                    { "_id" : ObjectId("5ad50d6269661a3e88d382f8") } -->> { "_id" : ObjectId("5ad50d6e69661a3e88d3a116") } on : p1 Timestamp(12, 0)
                    { "_id" : ObjectId("5ad50d6e69661a3e88d3a116") } -->> { "_id" : ObjectId("5ad50d7a69661a3e88d3c318") } on : p1 Timestamp(13, 0)
                    { "_id" : ObjectId("5ad50d7a69661a3e88d3c318") } -->> { "_id" : ObjectId("5ad50d8669661a3e88d3e136") } on : p1 Timestamp(7, 1)
                    { "_id" : ObjectId("5ad50d8669661a3e88d3e136") } -->> { "_id" : ObjectId("5ad50d9369661a3e88d40338") } on : p1 Timestamp(1, 11)
                    { "_id" : ObjectId("5ad50d9369661a3e88d40338") } -->> { "_id" : ObjectId("5ad50d9e69661a3e88d42156") } on : p1 Timestamp(1, 13)
                    { "_id" : ObjectId("5ad50d9e69661a3e88d42156") } -->> { "_id" : ObjectId("5ad50dac69661a3e88d44358") } on : p1 Timestamp(1, 14)
                    { "_id" : ObjectId("5ad50dac69661a3e88d44358") } -->> { "_id" : ObjectId("5ad50db769661a3e88d46176") } on : p1 Timestamp(1, 16)
                    { "_id" : ObjectId("5ad50db769661a3e88d46176") } -->> { "_id" : ObjectId("5ad50dc469661a3e88d48378") } on : p1 Timestamp(1, 17)
                    { "_id" : ObjectId("5ad50dc469661a3e88d48378") } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 18)
    {  "_id" : "shardTestDB2",  "primary" : "p2",  "partitioned" : true }
            shardTestDB2.users
                    shard key: { "username" : 1 }
                    unique: false
                    balancing: true
                    chunks:
                            p1      13
                    { "username" : { "$minKey" : 1 } } -->> { "username" : "user1" } on : p1 Timestamp(2, 0)
                    { "username" : "user1" } -->> { "username" : "user107146" } on : p1 Timestamp(6, 2)
                    { "username" : "user107146" } -->> { "username" : "user15844" } on : p1 Timestamp(6, 3)
                    { "username" : "user15844" } -->> { "username" : "user17257" } on : p1 Timestamp(6, 4)
                    { "username" : "user17257" } -->> { "username" : "user24515" } on : p1 Timestamp(4, 0)
                    { "username" : "user24515" } -->> { "username" : "user31774" } on : p1 Timestamp(5, 0)
                    { "username" : "user31774" } -->> { "username" : "user39033" } on : p1 Timestamp(6, 0)
                    { "username" : "user39033" } -->> { "username" : "user474" } on : p1 Timestamp(7, 0)
                    { "username" : "user474" } -->> { "username" : "user6061" } on : p1 Timestamp(8, 0)
                    { "username" : "user6061" } -->> { "username" : "user6787" } on : p1 Timestamp(9, 0)
                    { "username" : "user6787" } -->> { "username" : "user75128" } on : p1 Timestamp(10, 0)
                    { "username" : "user75128" } -->> { "username" : "user8921" } on : p1 Timestamp(11, 0)
                    { "username" : "user8921" } -->> { "username" : { "$maxKey" : 1 } } on : p1 Timestamp(12, 0)
```

```
  {  "_id" : "shipsdb",  "primary" : "p2",  "partitioned" : true }
          shipsdb.ships
                  shard key: { "_id" : 1 }
                  unique: true
                  balancing: true
                  chunks:
                          p1      1
                  { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(2, 0)
```

Figura 4.47. – Estado tras eliminación shard.

```
mongos> db.runCommand( { movePrimary: "shipsdb", to: "p1" })
{
        "ok" : 0,
        "errmsg" : "movePrimary may only be run against the admin database.",
        "code" : 13,
        "codeName" : "Unauthorized",
        "$clusterTime" : {
                "clusterTime" : Timestamp(1524084840, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1524084840, 2)
```

Figura 4.48. – Mover shipsdb a shard p1.

Para parar el proceso de "draining" definitivamente y que vuelva a balancear, es necesario el siguiente comando:

```
mongos> use config
switched to db config
mongos> db.shards.update({},{$unset:{draining:true}}, false, true)
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 1 })
```

```
{  "_id" : "piedras",  "primary" : "p1",  "partitioned" : true }
        piedras.minerales
                shard key: { "_id" : 1 }
                unique: true
                balancing: true
                chunks:
                        p1      7
                        p2      6
                { "_id" : { "$minKey" : 1 } } -->> { "_id" : "Cuarzo" } on : p2 Timestamp(14, 0)
                { "_id" : "Cuarzo" } -->> { "_id" : ObjectId("5ad50d476966la3e88d33e59") } on : p2 Timestamp(15, 0)
                { "_id" : ObjectId("5ad50d476966la3e88d33e59") } -->> { "_id" : ObjectId("5ad50d536966la3e88d35c77") } on : p2 Timestamp(16, 0)
                { "_id" : ObjectId("5ad50d536966la3e88d35c77") } -->> { "_id" : ObjectId("5ad50d626966la3e88d382f8") } on : p2 Timestamp(17, 0)
                { "_id" : ObjectId("5ad50d626966la3e88d382f8") } -->> { "_id" : ObjectId("5ad50d6e66966la3e88d3a116") } on : p2 Timestamp(18, 0)
                { "_id" : ObjectId("5ad50d6e66966la3e88d3a116") } -->> { "_id" : ObjectId("5ad50d7a66966la3e88d3c318") } on : p2 Timestamp(19, 0)
                { "_id" : ObjectId("5ad50d7a66966la3e88d3c318") } -->> { "_id" : ObjectId("5ad50d866966la3e88d3e136") } on : p1 Timestamp(19, 1)
                { "_id" : ObjectId("5ad50d866966la3e88d3e136") } -->> { "_id" : ObjectId("5ad50d936966la3e88d40338") } on : p1 Timestamp(1, 11)
                { "_id" : ObjectId("5ad50d936966la3e88d40338") } -->> { "_id" : ObjectId("5ad50d9e66966la3e88d42156") } on : p1 Timestamp(1, 13)
                { "_id" : ObjectId("5ad50d9e66966la3e88d42156") } -->> { "_id" : ObjectId("5ad50dac66966la3e88d44358") } on : p1 Timestamp(1, 14)
                { "_id" : ObjectId("5ad50dac66966la3e88d44358") } -->> { "_id" : ObjectId("5ad50db76966la3e88d46176") } on : p1 Timestamp(1, 16)
                { "_id" : ObjectId("5ad50db76966la3e88d46176") } -->> { "_id" : ObjectId("5ad50dc46966la3e88d48378") } on : p1 Timestamp(1, 17)
                { "_id" : ObjectId("5ad50dc46966la3e88d48378") } -->> { "_id" : { "$maxKey" : 1 } } on : p1 Timestamp(1, 18)
{  "_id" : "shardTestDB2",  "primary" : "p2",  "partitioned" : true }
        shardTestDB2.users
                shard key: { "username" : 1 }
                unique: false
                balancing: true
                chunks:
                        p1      7
                        p2      6
                { "username" : { "$minKey" : 1 } } -->> { "username" : "user1" } on : p2 Timestamp(13, 0)
                { "username" : "user1" } -->> { "username" : "user107146" } on : p2 Timestamp(14, 0)
                { "username" : "user107146" } -->> { "username" : "user15844" } on : p2 Timestamp(15, 0)
                { "username" : "user15844" } -->> { "username" : "user17257" } on : p2 Timestamp(16, 0)
                { "username" : "user17257" } -->> { "username" : "user24515" } on : p2 Timestamp(17, 0)
                { "username" : "user24515" } -->> { "username" : "user31774" } on : p2 Timestamp(18, 0)
                { "username" : "user31774" } -->> { "username" : "user39033" } on : p1 Timestamp(18, 1)
                { "username" : "user39033" } -->> { "username" : "user474" } on : p1 Timestamp(7, 0)
                { "username" : "user474" } -->> { "username" : "user6061" } on : p1 Timestamp(8, 0)
                { "username" : "user6061" } -->> { "username" : "user6787" } on : p1 Timestamp(9, 0)
                { "username" : "user6787" } -->> { "username" : "user75128" } on : p1 Timestamp(10, 0)
                { "username" : "user75128" } -->> { "username" : "user8921" } on : p1 Timestamp(11, 0)
                { "username" : "user8921" } -->> { "username" : { "$maxKey" : 1 } } on : p1 Timestamp(12, 0)
```

5.- PRUEBAS

En este capítulo se va a describir el entorno tecnológico sobre el que se van a llevar a cabo las pruebas. Además de detallarse las diferentes pruebas de carga realizadas con los datos de consumo eléctrico de la tabla Capturas enviados por el último datalogger real, así como los correspondientes resultados de las medidas comparándolos con los obtenidos previamente con el entorno inicial con MariaDB.

5.1.- Características tecnológicas

En las máquinas utilizadas está corriendo un Intel Xeon E5530 2.4 GHz con 8 GB de memoria y 3 núcleos de procesamiento. Además, el sistema ha sido desarrollado para ejecutarse sobre servidores Linux con Apache Web Server (2.4.18), PHP (7.0.8) y MariaDB (10.0.28), MongoDB (16.04) como bases de datos. El gestor de bases de datos seleccionado dispone de funcionalidades que permiten trabajar con el paradigma *Big Data*, manejando grandes cantidades de información.

El sistema va a poder integrarse en un entorno distribuido de tratamiento masivo de información mediante MongoDB. Se trata de un módulo que permite configurar clústeres de bases de datos escalables, permitiendo el balance de carga, la gestión de failovers y el método denominado *sharding*. Esta tecnología, como ya se ha mencionado, permite distribuir los datos entre distintos *shards* (conjuntos de servidores que almacenan parte de los datos), para que la carga a la hora de realizar consultas e insercciones se reparta.

5.2.- Simulaciones de arquitecturas

A lo largo de este apartado se va a estudiar el comportamiento de diversas simulaciones mediante operaciones de inserción, borrado y actualizado de unos determinados volúmenes de datos. Todo ello tanto con MariaDB como con MongoDB, dividiendo en este último caso entre *replicación* y *sharding*, con el fin de elegir la que mejor se adecue al sistema.

Los volúmenes de datos utilizados son datos obtenidos de la tabla **Capturas** de unas medidas concretas realizadas en el año 2017 con la plataforma inicial, de forma que las medidas de tiempo sean lo más reales posibles. Y se prueba con 50, 500, 2500, 5000, 25000 y 50000 elementos con el fin de poder ver cómo evoluciona el tiempo que tarda a medida que el número de elementos aumenta.

Cabe destacar que para la realización de todas las pruebas que se muestran a continuación se realizan unos scripts [8] que se adjuntan en el Anexo I de este trabajo. Cada uno de estos scripts se ejecuta 20 veces y se calcula el valor medio del tiempo de todas ellas.

5.2.1.- MariaDB

1) Insert

Los resultados que se obtienen del tiempo que se tarda en insertar varias cantidades de datos en MariaDB son los siguientes:

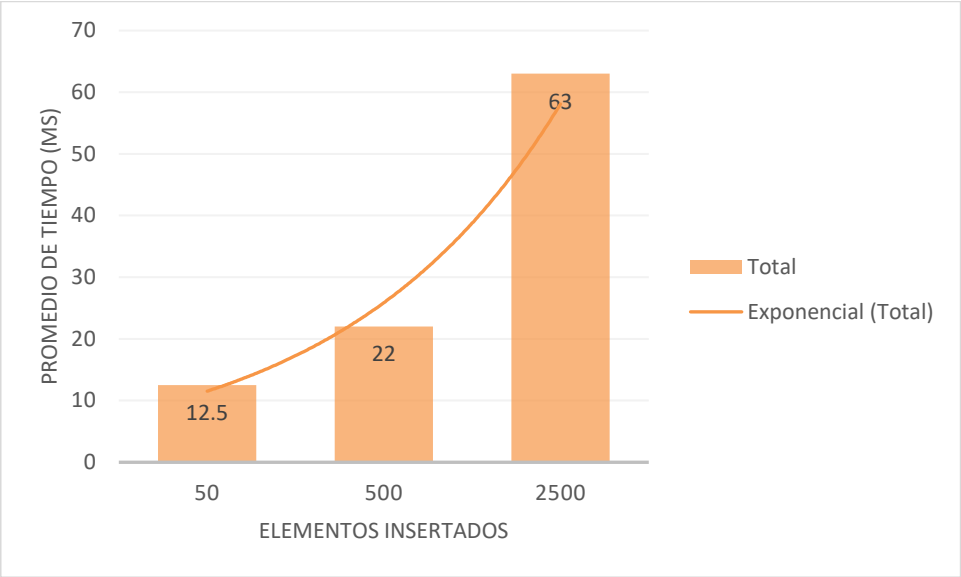

Figura 5.1. – Resultados inserción MariaDB pocos datos.

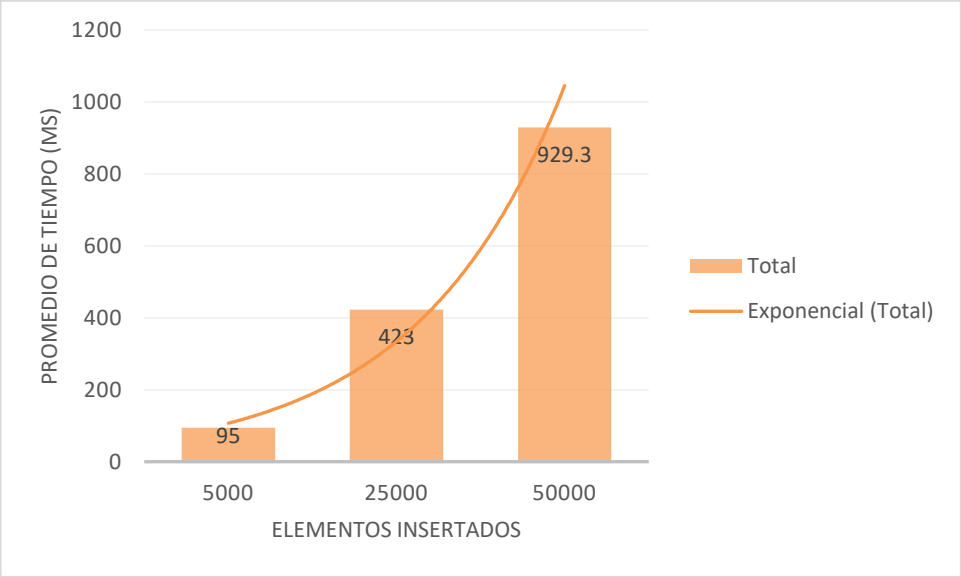

Figura 5.2. – Resultados inserción MariaDB muchos datos.

Observando las figuras obtenidas, es posible ver que el tiempo que tarda en realizar las inserciones aumenta considerablemente conforme lo hace el número de elementos, como era de esperar.

Estos resultados dependen mucho del tamaño de la base de datos en el momento de la inserción, ya que MariaDB tiende a un peor comportamiento cuando la DB se encuentra más congestionada.

2) Delete

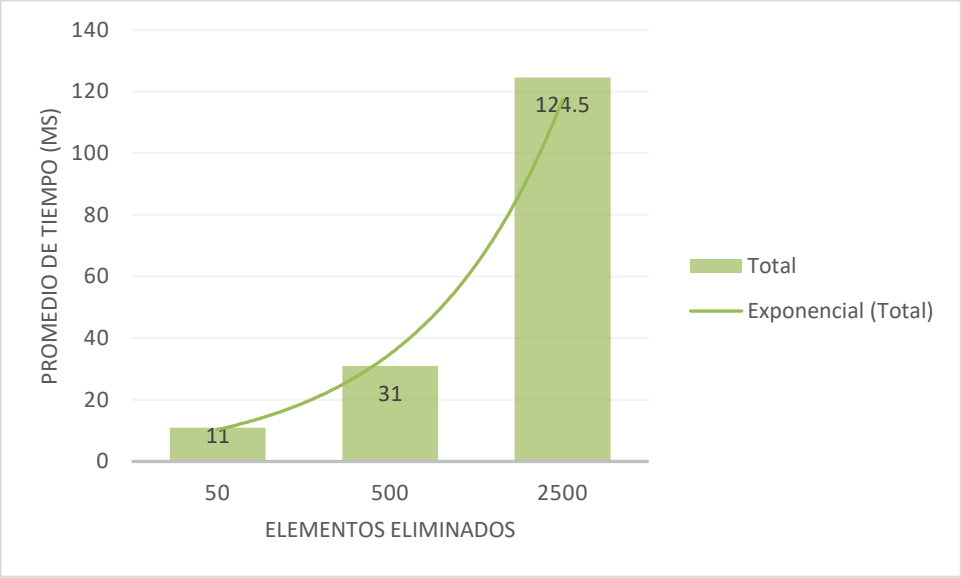

Figura 5.3. – Resultados eliminación MariaDB pocos datos.

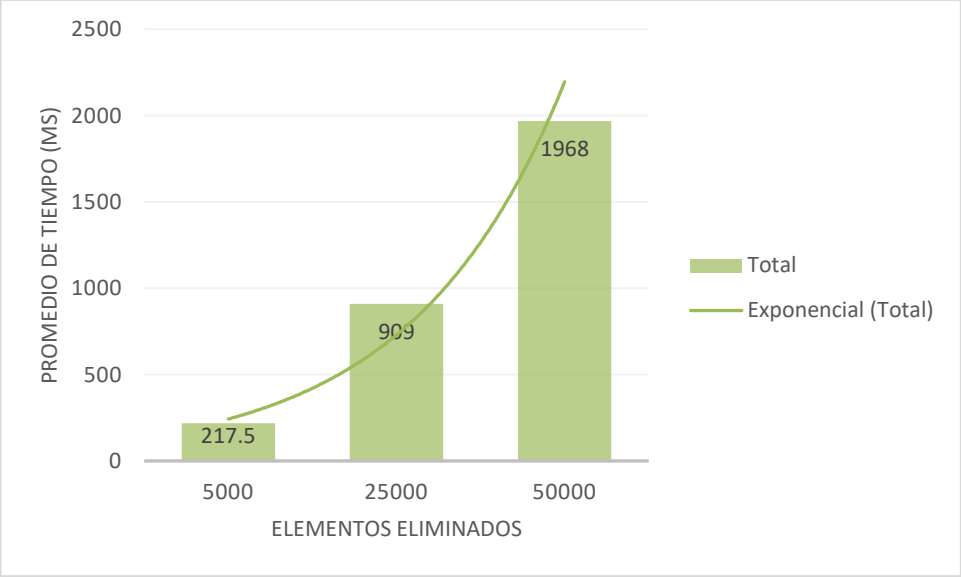

Figura 5.4. – Resultados eliminación MariaDB muchos datos.

3) Update

Para las pruebas de "*update*" se ha tratado de actualizar el parámetro *id_canal* con el mismo valor con la condición de que se haga para cada uno de los elementos (*id_variable*) de la tabla **Capturas**.

Primeramente, se insertan el número de elementos necesarios para ya posteriormente proceder a la actualización de los mismos con la condición mencionada.

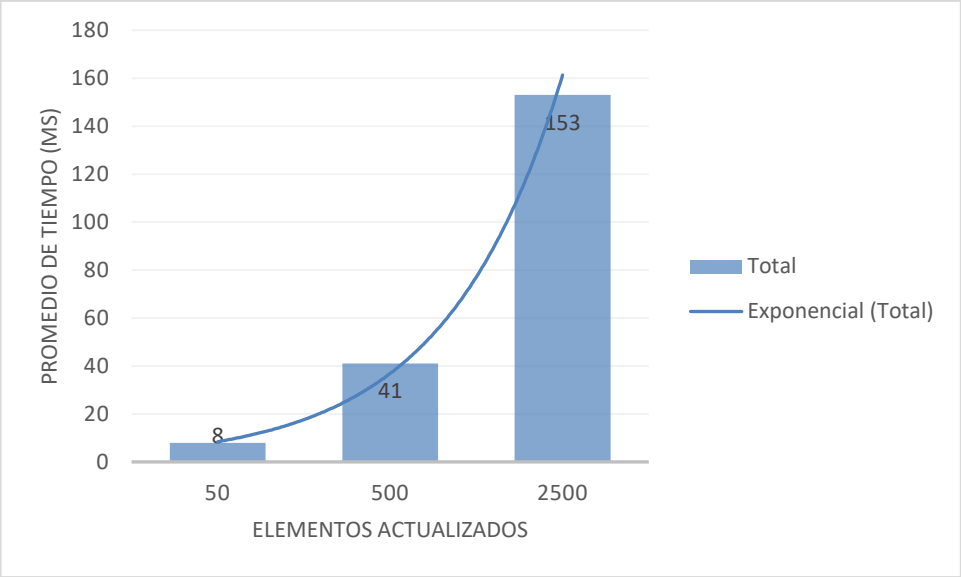

Figura 5.5. – Resultados actualización MariaDB pocos datos.

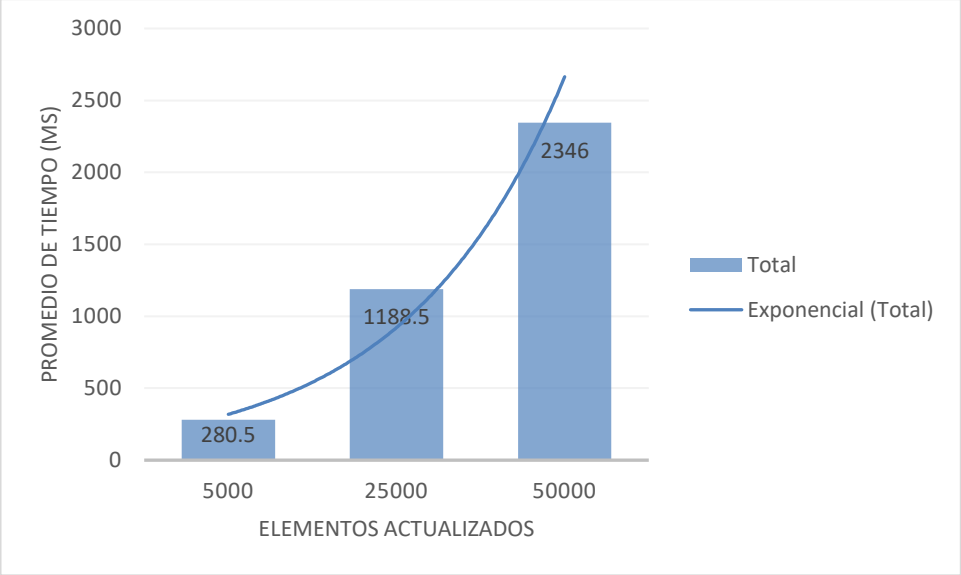

Figura 5.6. – Resultados actualización MariaDB muchos datos.

Según los resultados obtenidos de las diferentes pruebas en MariaDB, se puede observar como varía el tiempo que tarda en realizar las diferentes operaciones respecto al tamaño de datos, lo que se conoce como orden de complejidad. En este caso, esta evolución es de tipo normal, o lo que es lo mismo, que con el doble de datos tarda

aproximadamente el doble de tiempo. Lo peor sería que evolucionara de forma exponencial, que es que con el doble de datos tarda X veces más.

También se puede observar que, con volúmenes grandes de datos, la operación que más tarda en realizar es la de "*update*", llegando incluso a tardar hasta más del doble de tiempo que en el caso de "*insert*". A pesar de que, en esta operación, la evolución de tiempo en cuanto al volúmen de datos es algo mejor al tardar un poco menos del doble de tiempo con el doble de datos.

En el siguiente apartado se va a comprobar si utilizando MongoDB mejoran estos resultados y se consigue una mejor evolución.

5.2.2.- MongoDB sin sharding

Una vez se han visto las pruebas del entorno con MariaDB, se llevan a cabo las pruebas suponiendo que la *replicación* está ya activada. Estas pruebas se basan en configuración de replicación siguiente:

- Nodo primario: instancia para el nodo SideaCloneI que será el que acepte todas las peticiones de escritura:
  - SideaCloneI, mongod en el puerto 27017.
- Nodos secundarios: instancias para los nodos SideaCloneII y SideaCloneII donde se replicarán los datos del primario:
  - SideaCloneII, mongod en el puerto 27018.
  - SideaCloneIII, mongod en el puerto 27019.

Tras la configuración, hay que conectarse con los 3 clientes a cada uno de los nodos y con sus puertos correspondientes para que todo funcione correctamente.

1) Insert

En este caso, se van a realizar dos pruebas de inserción, la primera sin tener en cuenta cuántos elementos hay en la DB y, la segunda, eliminando todo el contenido de la DB tras cada inserción. De esta forma, se va a poder comprobar si el hecho de que la DB esté más o menos liberada influye en los resultados.

Los resultados obtenidos para la primera prueba de inserción son los siguientes:

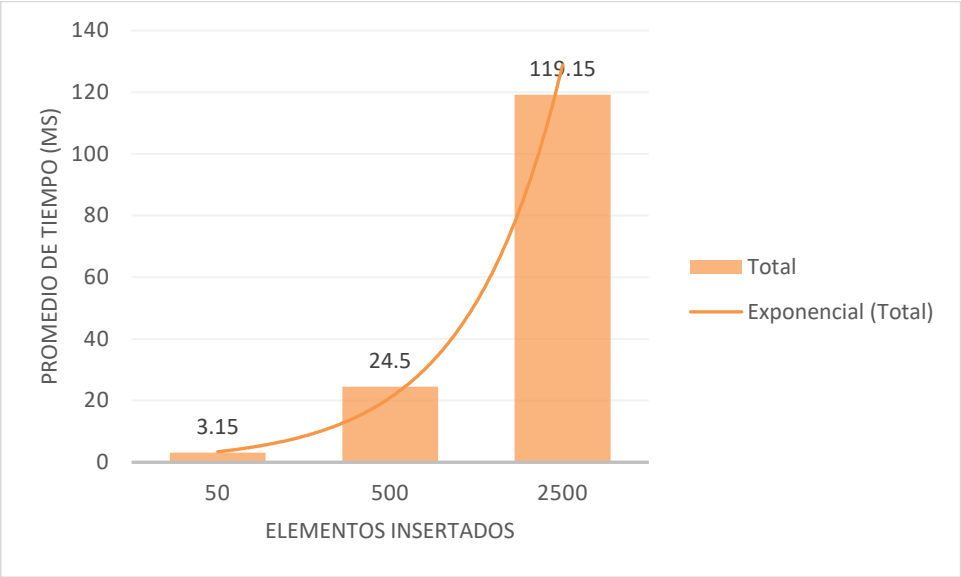

Figura 5.7. – Resultados inserción MongoDB replicación pocos datos.

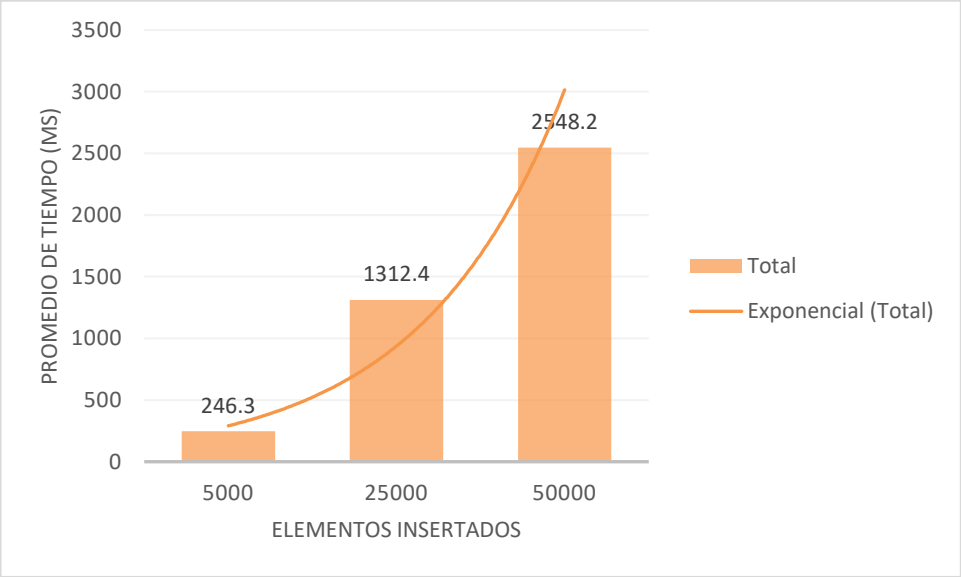

Figura 5.8. - Resultados inserción MongoDB replicación muchos datos.

Y realizando las inserciones cuando la base de datos está vacía se obtiene:

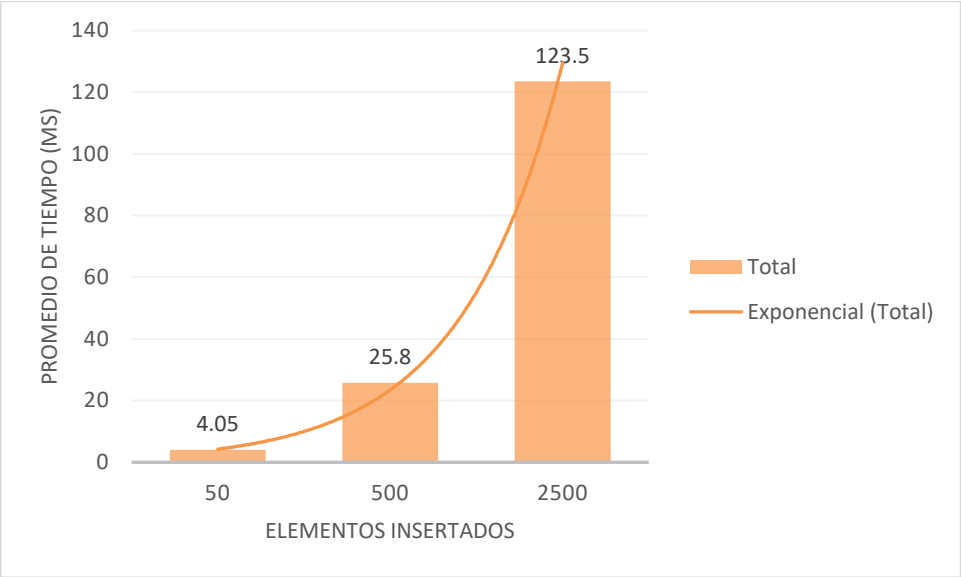

Figura 5.9. - Resultados inserción MongoDB replicación pocos datos y DB vacía.

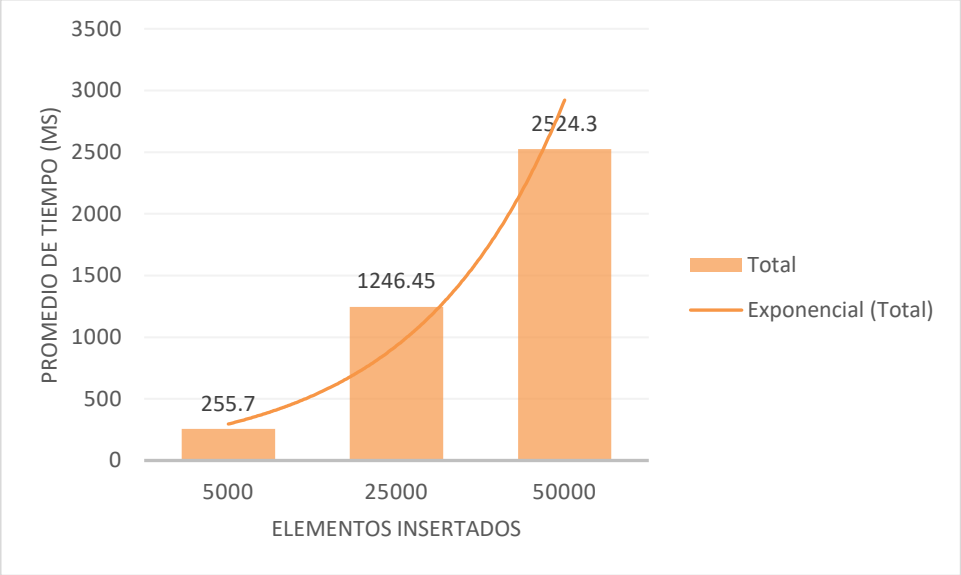

Figura 5.10. - Resultados inserción MongoDB replicación muchos datos y DB vacía.

Según los resultados obtenidos a lo largo de este apartado, el tiempo de inserción en colecciones vacías en Mongo es más bajo que cuando ésta tiene datos. Lo cual se aprecia en mayor medida cuando la cantidad de datos a insertar es mayor y también, cuando la base de datos está más llena.

Por tanto, si es posible al inicio del proceso, lo mejor sería borrar completamente la colección y volver a generarla de nuevo con los datos que se va a insertar.

2) Delete

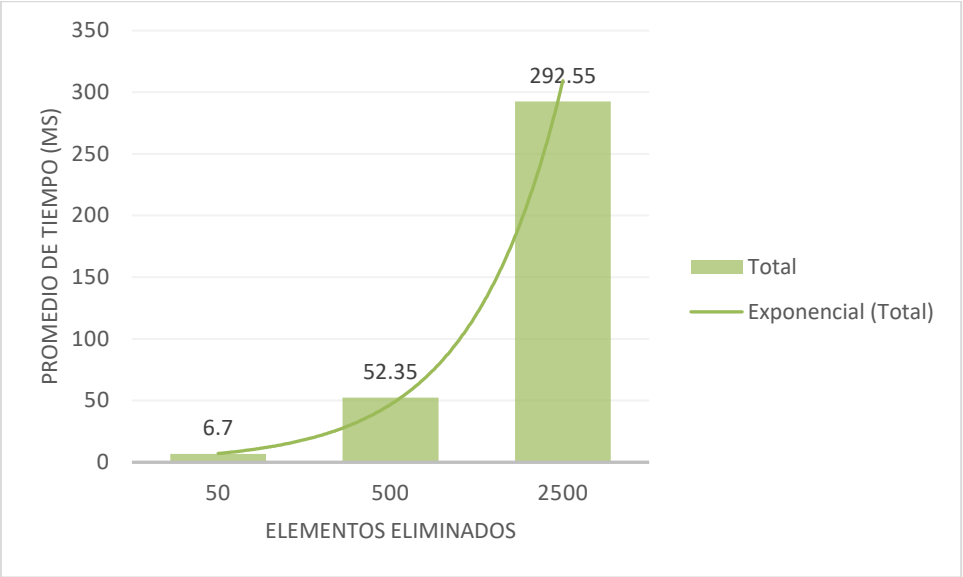

Figura 5.11. - Resultados eliminación MongoDB replicación pocos datos.

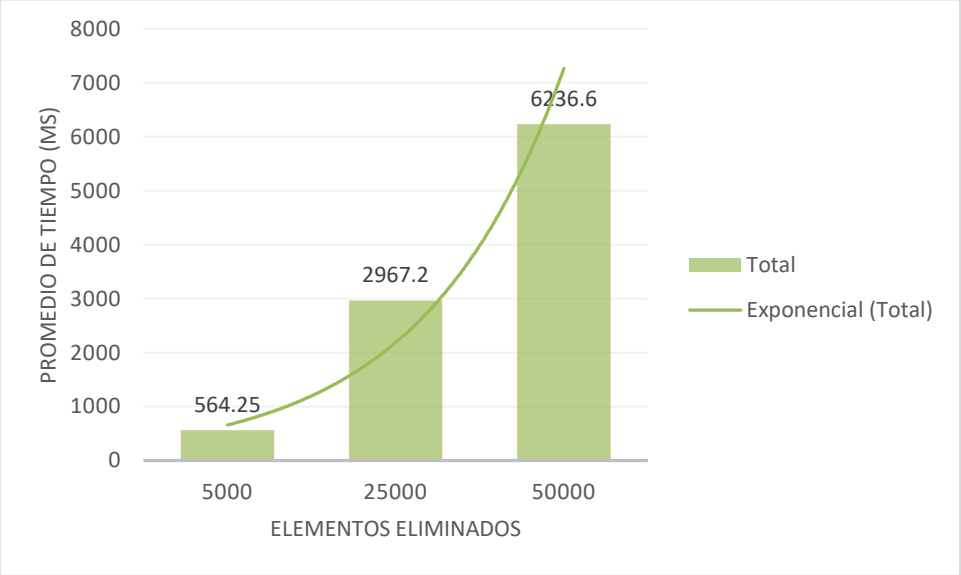

Figura 5.12. - Resultados inserción MongoDB replicación muchos datos.

3) Update

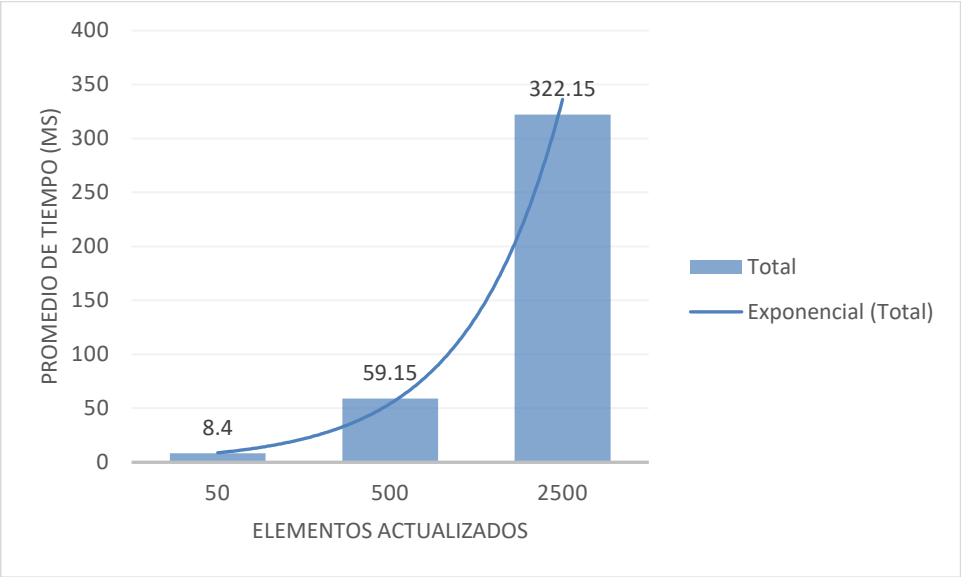

Figura 5.13. - Resultados actualización MongoDB replicación pocos datos.

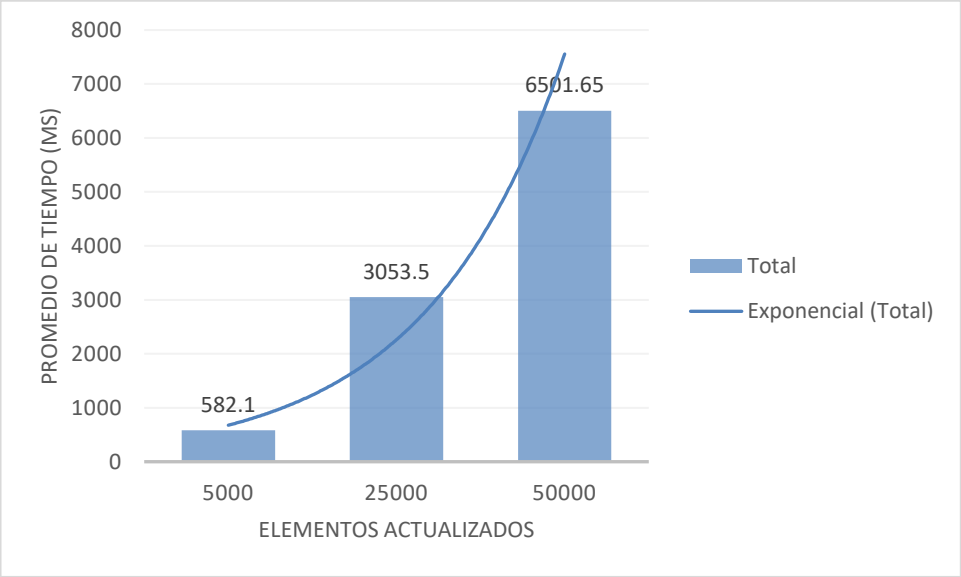

Figura 5.14. - Resultados actualización MongoDB replicación muchos datos.

Aunque a priori se pueda pensar que en este caso no debería influir el hecho de que haya una cantidad elevada de datos en la base de datos al ejecutar el "*update*", ya que se limita a realizar una búsqueda de unos elementos concretos y a actualizar unos parámetros de esos elementos. Pero, en base a los resultados obtenidos, se ha podido ver que sí que influye y que realiza la operación ligeramente más rápido que en el otro caso.

Al igual que en el primer apartado con MariaDB, se observa el orden de complejidad con MongoDB + *replicación*, obteniendo la evolución tipo logarítmica deseada para el caso de pocos datos. Este tipo es que con el doble de datos tarda menos del doble de tiempo. Pero, para el caso de muchos datos, la evolución sigue siendo de tipo normal tardando aproximadamente el doble de tiempo.

También se ve que la operación que más tarda en realizar sigue siendo la de "*update*", siendo más de doble de tiempo que la de "*insert*". Pero, en este caso la operación de "*delete*" tarda casi el mismo tiempo que la de "*update*". Esto puede ser debido a que, al estar replicados todos los datos entre los diferentes nodos, tarda más en encontrarlos y en terminar de borrarlos.

5.2.3.- MongoDB con sharding

De igual forma, se procede con las pruebas de MongoDB con *sharding* y suponiendo que la *replicación* y el balanceo están también activados. La configuración en este caso se basa en lo siguiente:

- Base de datos de configuración: Primer clon SideaCloneI y con dos nodos en el grupo de réplica para el almacenamiento de la configuración:
    - Mongod en el puerto 26050.
    - Mongod en el puerto 26051.
- Shards: instancias para los nodos SideaCloneII y SideaCloneIII entre los que se realizará el particionado de datos:
    - Primer conjunto (SideaCloneII): mongod en los puertos 27100 y 27101.
    - Segundo conjunto (SideaCloneIII): mongod en los puertos 27200 y 27201.

Al igual que en la replicación, una vez configurado todo, hay que conectarse a mongo e indicarle los shards mencionados.

1) Insert

A continuación, se pueden ver los resultados obtenidos para el caso de la inserción, tanto para el caso de insertar pocos datos como para una gran cantidad de estos.

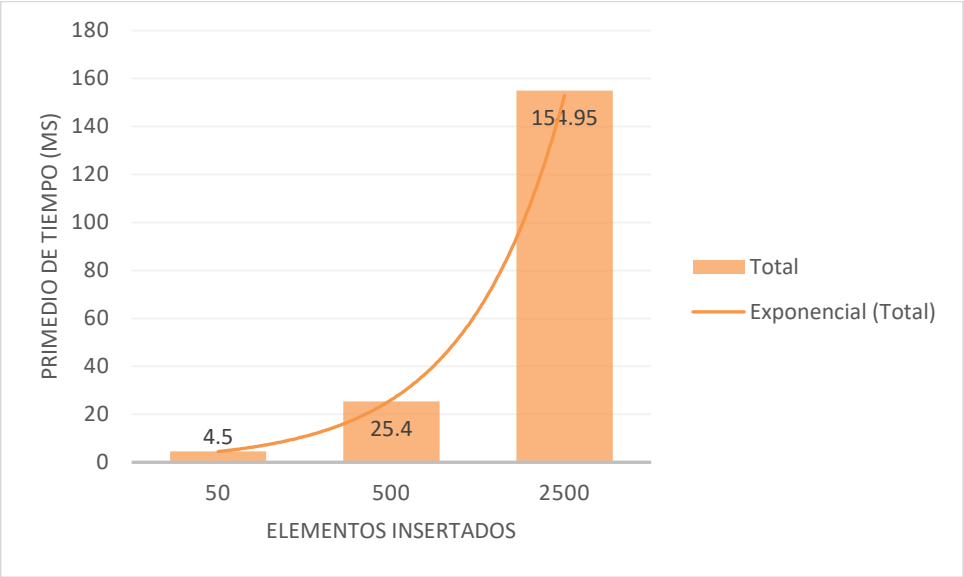

Figura 5.15. - Resultados inserción MongoDB sharding pocos datos.

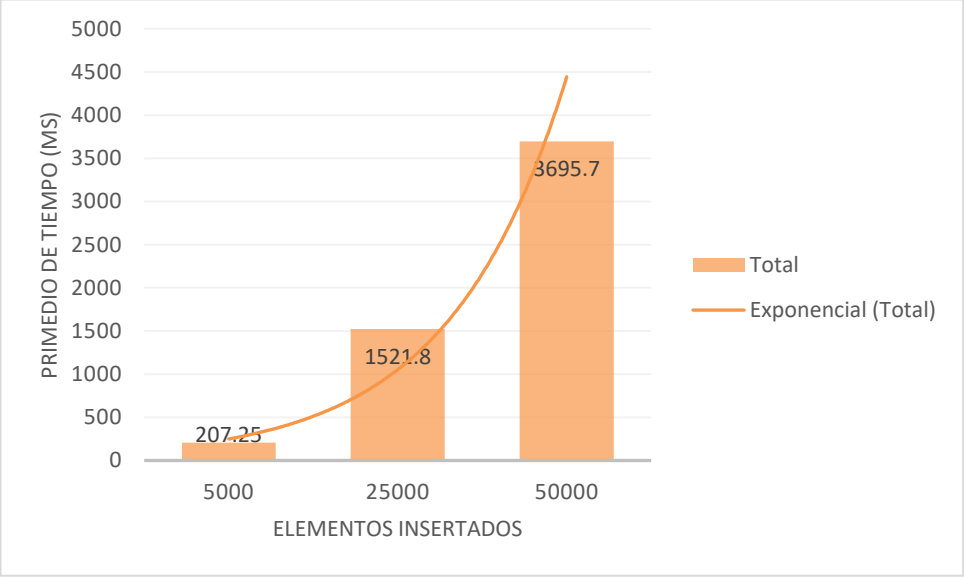

Figura 5.16. - Resultados inserción MongoDB sharding muchos datos.

2) Delete

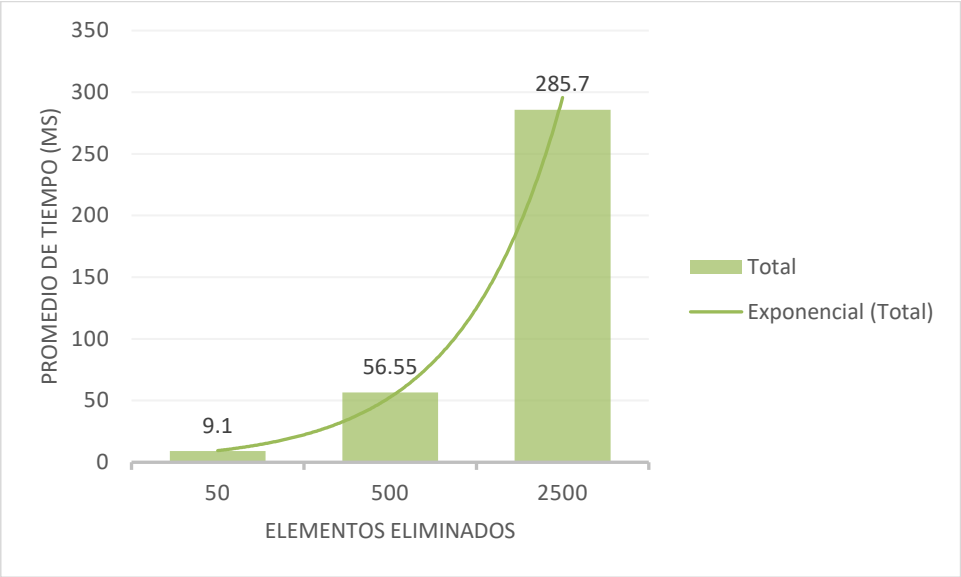

Figura 5.17. - Resultados eliminación MongoDB sharding pocos datos.

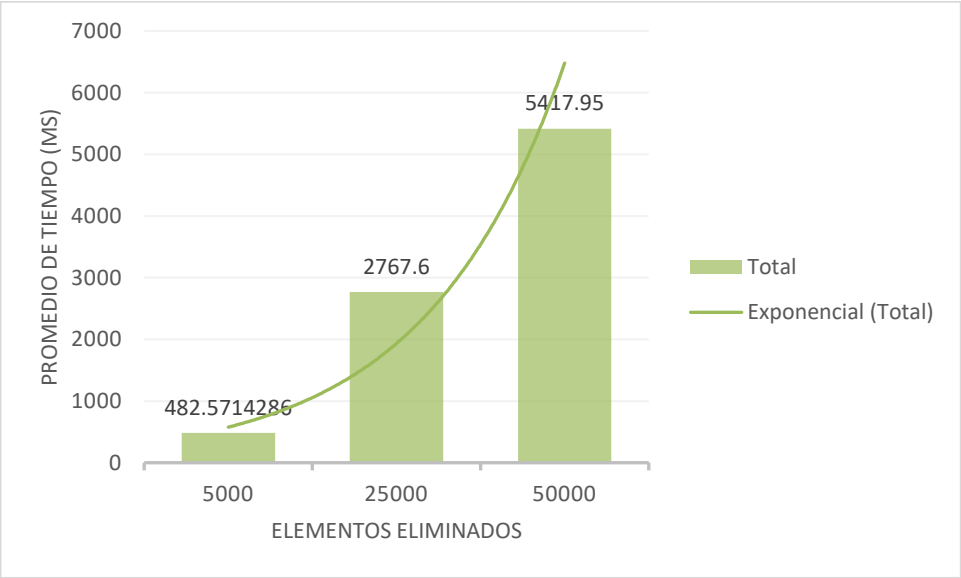

Figura 5.18. - Resultados eliminación MongoDB sharding muchos datos.

3) Update

Las gráficas que resumen el tiempo que tarda en realizar "*update*" de los diferentes volúmenes de datos considerados son:

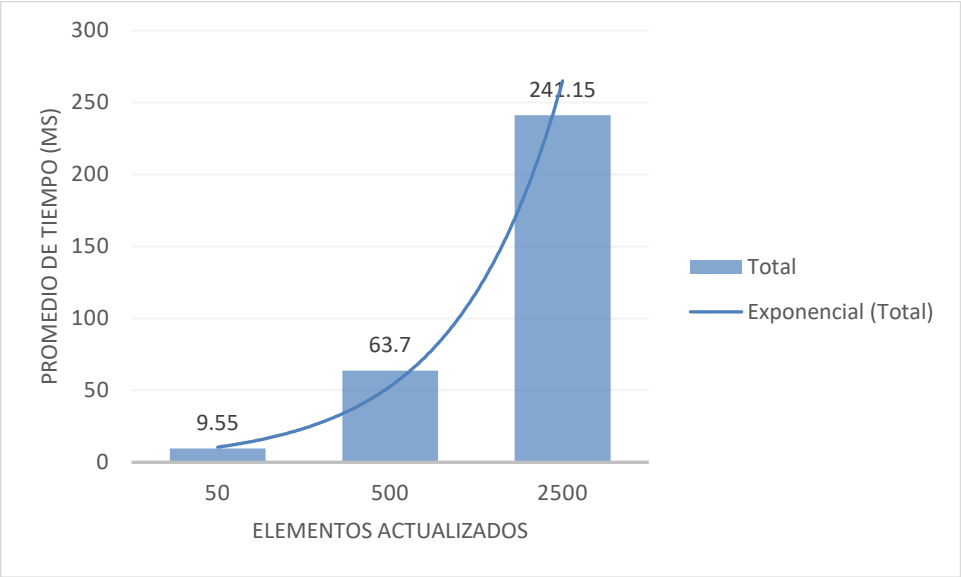

Figura 5.19. - Resultados actualización MongoDB sharding pocos datos.

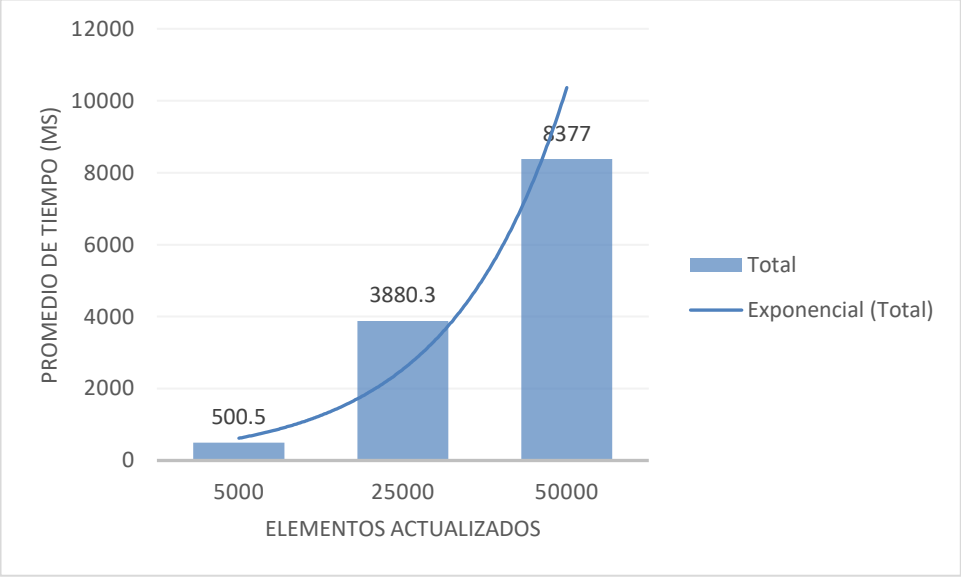

Figura 5.20. - Resultados actualización MongoDB sharding muchos datos.

Analizando de nuevo el orden de complejidad en MongoDB con *sharding*, se obtienen unos resultados similares al caso anterior con *replicación*, con evolución de tipo logarítmica para pocos datos y normal para muchos datos. Además, la operación que más tarda en realizar sigue siendo la de "*update*", pero se reduce considerablemente el tiempo en el caso de "*delete*".

La configuración que ha sido la más lenta en cuanto a resultados es la de MongoDB con *replicación* y sin *sharding*, siendo totalmente lógico que sea peor que con *sharding*.

Teniendo en cuenta todas las pruebas realizadas con los seis volúmenes de datos y las diferentes configuraciones se pudo ver que, a pesar de lo que se podía pensar en un primer momento, en este caso MariaDB tarda menos tiempo en realizar las operaciones. Incluso, en muchos de los casos este tiempo llega a ser más de la mitad que con replicación o sharding.

5.2.4.- Análisis de resultados

En este apartado se muestran los resultados en conjunto obtenidos de las tres configuraciones realizadas con el propósito de poder ver y analizar con mayor claridad cual de las opciones es la mejor para el sistema, tras haber analizado individualmente cada uno de ellos.

1) Insert

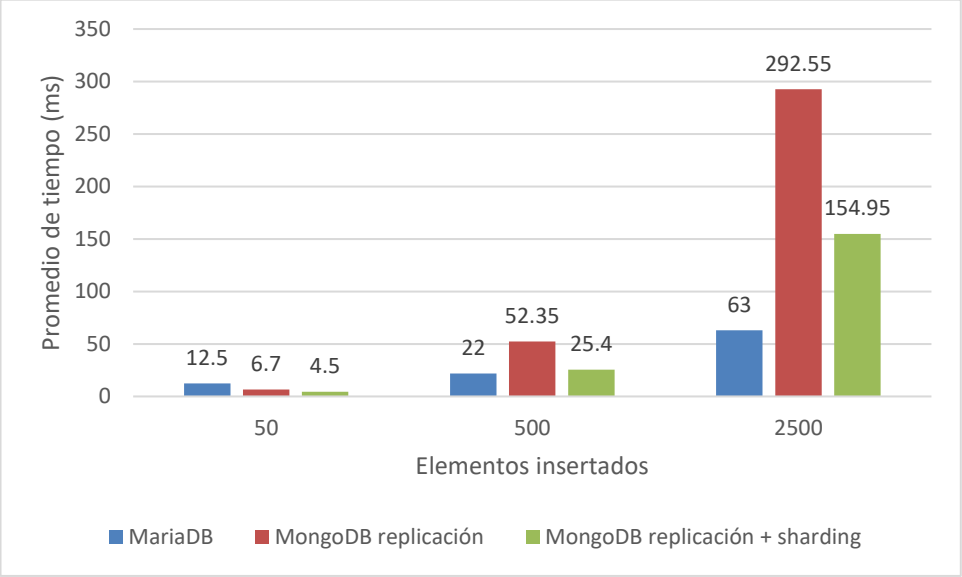

Figura 5.21. – Resultados conjuntos inserción pocos datos.

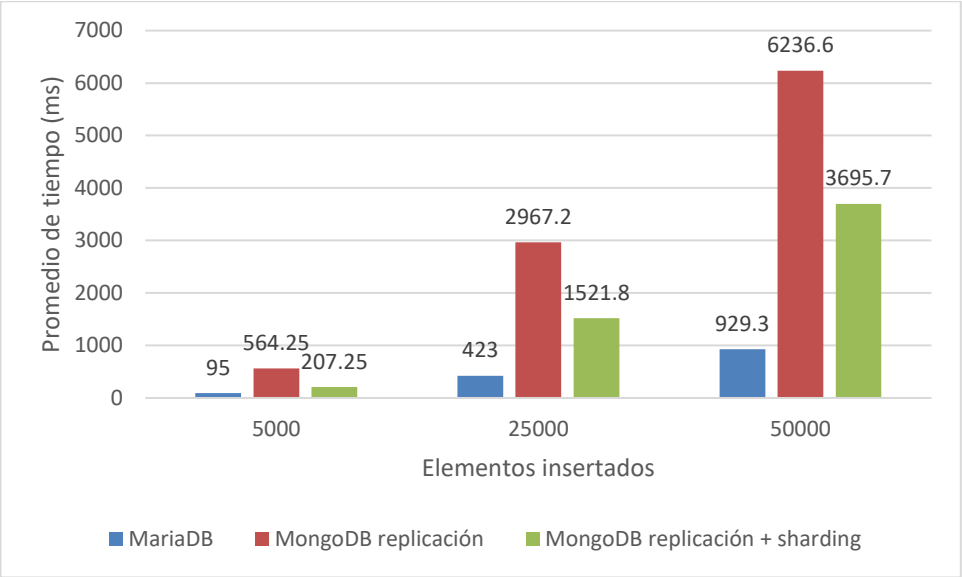

Figura 5.22. – Resultados conjuntos inserción muchos datos.

2) Delete

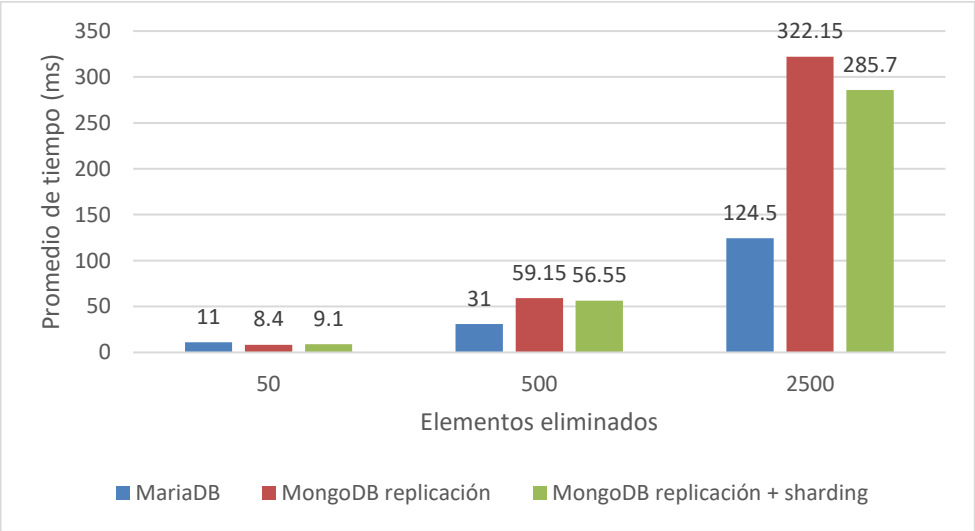

Figura 5.23. – Resultados conjuntos eliminación pocos datos.

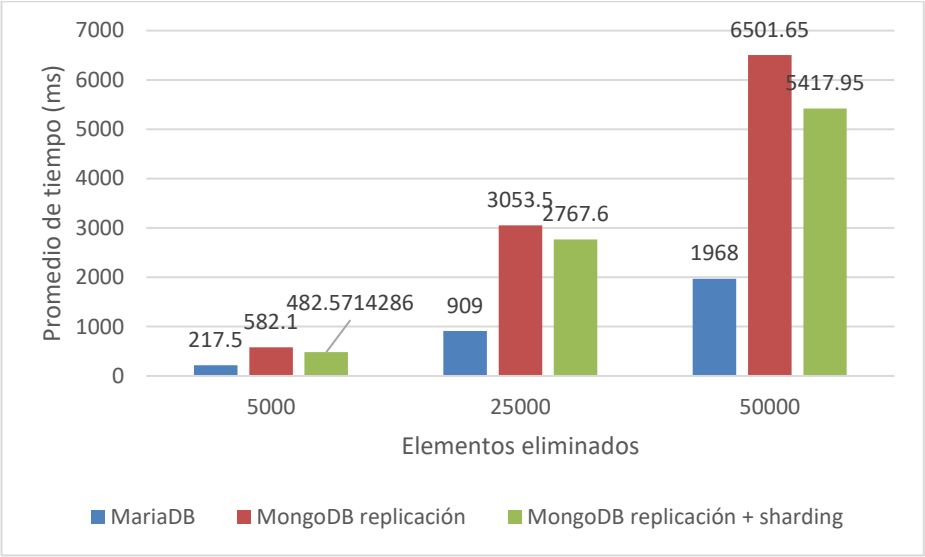

Figura 5.24. – Resultados conjuntos eliminación muchos datos.

3) Update

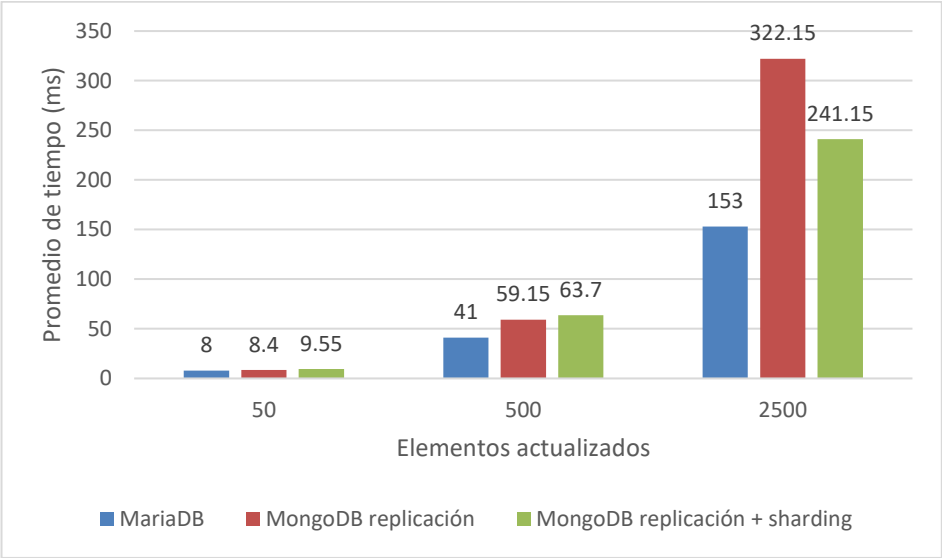

Figura 5.25. – Resultados conjuntos actualización pocos datos.

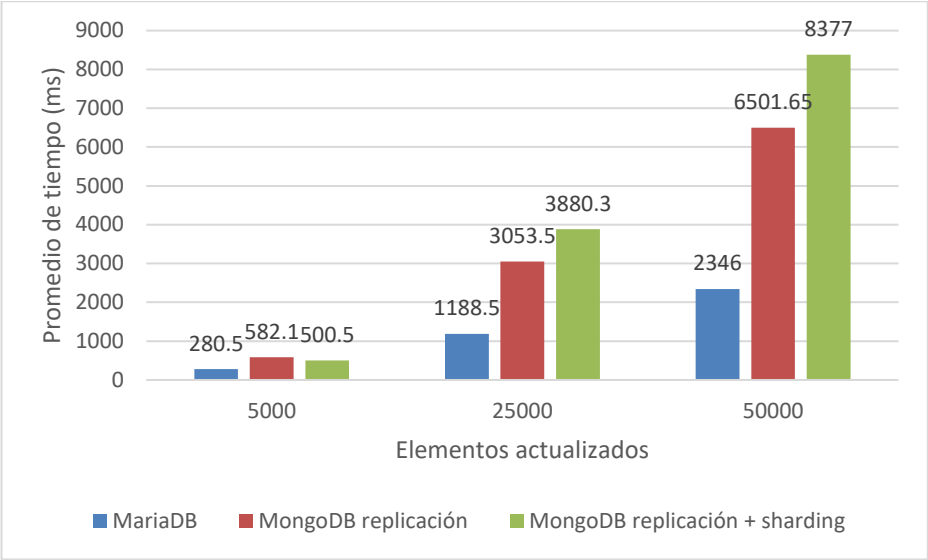

Figura 5.26. – Resultados conjuntos actualización muchos datos.

Se puede concluir que sustituyendo la tabla **Capturas** de la base de datos con MongoDB no se gana en tiempo con respecto a la configuración inicial con MariaDB, para las operaciones analizadas de "*insert*", "*update*" y "*delete*". Lo cual se podía esperar porque las bases de datos SQL están muy preparadas para dichas operaciones en MariaDB y la DB estaba muy descargada en el momento de las pruebas. Todo ello a pesar de que, en cuanto al orden de complejidad, MongoDB tiene una mejor evolución al tardar en algunos casos menos de doble de tiempo con el doble de datos.

Por otro lado, gracias a MongoDB y a su estructura, se aumenta la escalabilidad del sistema inicial ya que con MariaDB el servidor puede llegar a congestionarse en el momento en el que se guarde un número elevado de datos.

Por último, si se comparan todos los resultados con los obtenidos en otros artículos como [9] [10], se ve que en este caso no existe una mejora clara de la base NoSQL con respecto a la relacional. Y es que los tiempos que tarda en realizar las operaciones son bastante más altos en MongoDB, siendo MongoDB con replicación el peor caso. Pero, estos tiempos podrían llegar a variar mucho en función de si las operaciones de "*insert*", "*delete*" y "*update*" se realizan con un número elevado de condiciones, comportándose en ese caso mejor MongoDB.

Cabe destacar que para un análisis completo habría que realizar varias configuraciones de *replicación* y *sharding*, con diferente número de nodos, realizando las operaciones de

manera concurrente, etc. Esto permitiría ver si existe la posibilidad de reducir los tiempos con una configuración determinada en MongoDB.

6.- CONCLUSIONES Y TRABAJOS FUTUROS

6.1.- Conclusiones generales

En este documento se presenta una mejora de la plataforma cloud *Energy Analytics* diseñada con el sistema de bases de datos MariaDB, mediante otro sistema de bases de datos MongoDB y empleando la tecnología *Cloud Computing*. La plataforma final está formada por un cluster de 3 nodos configurado mediante las técnicas de *replicación* y *sharding*.

El proceso de mejora de la plataforma descrita comienza por una primera toma de contacto con la base de datos MongoDB, su instalación y su configuración. En primer lugar, se ha visto el correcto funcionamiento en la prueba llevada a cabo de replicación, comportándose uno de los nodos del clúster como primario y replicándose los datos insertados a los secundarios. Por otro lado, al realizar la prueba de *sharding*, se comprobó también como los datos insertados en una DB de prueba se balanceaban correctamente entre los shards configurados.

Seguidamente, se realizaron las pruebas de carga con MariaDB. Y se obtuvieron unos gráficos comparativos de tiempo para las distintas operaciones de "*insert*", "*update*" y "*delete*"; y los distintos volúmenes de datos. Se vio como el orden de complejidad en general es de tipo normal, tardando en realizar las operaciones aproximadamente el doble de tiempo con el doble de datos.

Tras las pruebas de carga con MariaDB, se llevaron a cabo también las mismas con el sistema MongoDB elegido, primero sólo configurando la *replicación* y, después con ésta y *sharding*. En este caso se obtuvieron unos resultados algo mejores al conseguir una evolución de tipo logarítmica para pocos datos, tardando menos del doble de tiempo en realizar cada una de las operaciones estudiadas.

Finalmente, comparando los resultados de los promedios de tiempo obtenidos en las diferentes simulaciones de arquitecturas realizadas, se vio que en general MariaDB tarda menos tiempo en realizar las operaciones que MongoDB sin o con *sharding*. En conclusión, no ha sido posible obtener una reducción del tiempo con respecto a la plataforma inicial con

MariaDB, pero si se ha aumentado la escalabilidad del sistema gracias a su estructura y se ha mejorado la evolución del tiempo que tarda en realizarlas conforme al volumen de datos.

6.2.- Trabajos futuros

Se plantean como posibles líneas futuras de trabajo:

- Tener en cuenta diferentes configuraciones para las pruebas de carga.
- Realizar pruebas con otro tamaño de chunk del sharding.

REFERENCIAS

[1] *MariaDB*. (2018). Obtenido de https://mariadb.com/kb/es/about-mariadb/

[2] *MongoDB*. (2018). Obtenido de https://docs.mongodb.com/manual

[3] Karwin, B. (2014). Quora. Obtenido de https://www.quora.com/Why-isnt-MySQL-Cluster-more-popular-than-either-NoSQL-solutions-or-hand-sharded-MySQL-configurations

[4] *MongoDB*. (2018). Obtenido de https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

[5] Netmind. (2018). *bit*. Obtenido de Computer Training: https://www.bit.es/knowledge-center/mongodb-para-big-data-replicacion-y-sharding-i/

[6] Netmind. (2018). *bit*. Obtenido de Computer training: https://www.bit.es/knowledge-center/mongodb-para-big-data-replicacion-y-sharding-iii/

[7] Alonso, J. D. (2013). *Happy Minds Software*. Obtenido de http://www.happyminds.es/eliminar-datos-en-mongodb/#sthash.YN3SW0ce.2opKLdv7.dpbs

[8] *MongoDB*. (2018). Obtenido de https://docs.mongodb.com/manual/tutorial/write-scripts-for-the-mongo-shell/

[9] Wittawat Puangsaijai, S. P. (2017). A Comparative Study of Relational Database and Key-Value Database for Big Data Applications. 2-4.

[10] Min-Gyue Jung, S.-A. Y.-L. (2015). A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment. 3-4.