ANÁLISIS COMPARATIVO ENTRE BASES DE DATOS NOSQL MONGODB Y CASSANDRA

Autor: Lic. Jonathan Fernando ROMANO

Director: Dr. Waldo HASPERUÉ

"Trabajo Final presentado para obtener el grado de Especialista en Inteligencia de Datos Orientada a Big Data”

"Facultad de Informática - Universidad Nacional de La Plata"

Diciembre – 2024

Índice

Contenido

Capítulo 1: Introducción

Día a día la interacción que existe entre las personas y el mundo de la tecnología, hacen que cada segundo que pase, se generen datos a un ritmo exponencial, por tal motivo, como dijo el Presidente de Telefónica, José María Álvarez-Pallete, en una conferencia en Madrid, ***"Los datos son el petróleo del siglo XXI"*** [14].

Las bases de datos tradicionales han enfrentado desafíos significativos a la hora de procesar y almacenar datos de manera eficiente y escalable, esto llevó a que las bases de datos NoSQL (Not Only SQL), tengan una importancia significativa a la hora de elegir un modelo de datos para dar solución a un problema o al planteo de un nuevo sistema.

Actualmente muchos sistemas necesitan manejar grandes volúmenes de datos de información en tiempo real, por eso es que las bases de datos NoSQL han ganado relevancia y popularidad. En lugar de depender de tablas y relaciones predefinidas, como en las bases de datos relacionales, las bases de datos NoSQL permiten almacenar y recuperar datos no estructurados o semi estructurados. Esto significa que pueden adaptarse fácilmente a diferentes tipos de información, como documentos, grafos, columnas y clave-valor.

La escalabilidad también es un factor crítico en el mundo de los datos. A medida que las empresas generan cada vez más información y las aplicaciones enfrentan una demanda creciente, las bases de datos NoSQL ofrecen la capacidad de escalar horizontalmente agregando más servidores y distribuyendo la carga de trabajo de manera eficiente. Esto permite que las aplicaciones mantengan un alto rendimiento, incluso en situaciones de alta concurrencia donde no solo la demanda y disponibilidad de los datos es relevante, sino también, el tiempo de respuesta, la tasa de transferencia y latencias que existen en los procesos de un sistema o conjuntos de sistemas.

Si bien existen muchas soluciones a la hora de hablar de bases de datos NoSQL (MongoDB, Apache Cassandra, CouchDB, Redis, Neo4j, Apache HBase, AmazonDynamoDB, Apache CouchBase), el presente Trabajo Final Integrador, pretende hacer un análisis comparativo de dos bases de datos muy conocidas en el ambiente del desarrollo de software y utilizadas por grandes empresas de tecnología, estas son, MongoDB [1] y Apache Cassandra [4].

Para la comparativa utilizaremos el software Yahoo Cloud Serving Benchmark (YCSB) [9], ya que el mismo nos permite crear un marco de trabajo estándar, sobre el cual establecemos un conjunto común de cargas de trabajo para evaluar el rendimiento de ambas bases de datos y sus propiedades de servicio "clave-valor".

YCSB, es uno de los más utilizados al momento de comparar mediciones referentes a tiempos de respuesta, latencias, tasa de transferencia, etc., abordaremos las pruebas desde su instalación y configuración, hasta las pruebas de carga con datasets de datos preestablecidos, logrando así tener una visión más didáctica con una guía completa al momento de levantar el ambiente de trabajo.

1.1 Motivación

La amplia gama de bases de datos relacionales y no relacionales que existe hoy en día, hacen que muchas personas se pregunten, ¿Qué tipo de base de datos o modelo de datos se adapta mejor a los requerimientos y objetivos de un sistema a desarrollar?

Cuando se llega a ese punto, se necesita una serie de indicadores que faciliten la elección de una u otra solución. Es allí que, buscamos toda fuente de información y conocimiento que nos permita elegir la mejor herramienta para implementar.

En la web existen numerosos trabajos de análisis y comparativas (informes, papers, tesis, etc.) entre distintas bases de datos (NoSQL Databases: MongoDB vs Cassandra [12], Performance Evolution of Wide Column Store NoSQL Database in non-distributed environment [10], Base de Datos NoSQL: MongoDB vs. Cassandra en operaciones CRUD [21]). Los mismos suelen tener un enfoque cuantitativo con respecto a las mediciones de tiempo y tasa de transferencia de grandes volúmenes de datos. Estos trabajos parten de un ambiente totalmente resuelto, lo cual deja al usuario una explicación prácticamente resuelta.

Por otro lado, las fuentes de información de la herramienta YCSB, parten desde las bases de datos ya instaladas, no permitiendo así, poder entender desde el punto de vista académico, como es su instalación y relación con el sistema YCSB.

Se pudo identificar en la web y en foros, una gran cantidad de dudas y consultas con respecto a, “COMO REALIZAR EL DESPLIEGUE”, y hasta la falta de interpretación en los valores arrojados en los resultados de las cargas de trabajo (WORKLOADS) de YCSB.

Es así que, sumado al mismo tipo de enfoque cuantitativo, se buscará, además, describir de forma metodológica, el proceso para la preparación del ambiente de prueba y análisis (preparación e instalación de las Bases de datos), utilizando la tecnología de contenedores Docker.

1.2 Objetivo

El objetivo general del presente Trabajo Final Integrador es realizar un análisis comparativo entre dos bases de datos No SQL: MongoDB y Cassandra.

El análisis aplicado a las bases de datos mencionadas anteriormente, se centra en, Operaciones y Medidas. Dentro de las Operaciones se evaluarán, la Fase de Carga y la Fase Transaccional. Con respecto a las Medidas, serán considerados el Tiempo de Ejecución, Tasa de Transferencia Efectiva y Latencias.

Para la experimentación del análisis comparativo entre ambas bases, se utilizará un sistema específico para ese trabajo llamado, YCSB (Yahoo! Cloud Serving Benchmark). Este Framework está basado en dos componentes, *Generador de Cargas de Trabajo* y *Escenarios.* Los escenarios de evaluación son llamados Workloads, en cada escenario un Workload genera y almacena datos de prueba dentro de cada base de datos. El set de datos que se utiliza en los distintos test, se genera de manera automática por YCSB, los mismos son representados por un conjunto de

caracteres aleatorios y organizados (10 campos por registro), con un peso aproximado de un Kilobyte por registro. Cada Workload ejecutado combina una serie de operaciones de tipo CRUD (Create, Read, Update, Delete) sobre los datos generados acorde al escenario.

Como objetivo secundario se desplegará el entorno de base de datos en contenedores DOCKER, el cual incluye, motores de bases de datos NoSQL y sus respectivos entornos Cliente para la manipulación de la información.

Entre los temas relevantes del trabajo se abordará, las definiciones de Bases de Datos No SQL, el porqué de su necesidad en el ámbito del Big Data y su impacto en la actualidad, los contenedores Docker como forma de trabajo, despliegue rápido de aplicaciones, creación de archivos de configuración (Docker-Compose) para la interoperabilidad entre contenedores sobre una misma red, y, por último, definición, objetivos y funcionamiento de YCSB.

El aporte que brindará el presente trabajo permitirá contar con una guía básica, pero completa, partiendo desde la creación de un contenedor, pasando por el despliegue de forma rápida y segura de diferentes tipos de bases de datos, hasta poder comprender y visualizar los datos arrojados por la herramienta de Benchmark YCSB.

Capítulo 2: Bases de Datos NoSql

A lo largo de la historia de la tecnología de la información, las bases de datos han experimentado transformaciones muy relevantes. Desde el hito más importante en la historia de las bases de datos modernas, cuando en 1970, Edgar Frank Codd [16] publicó un importante documento para proponer el uso de un modelo de base de datos relacional, organizadas en tablas, con filas y columnas, hasta la actualidad, donde las bases de datos han evolucionado hacia nuevas formas de trabajar con los datos de manera horizontal, y en entornos de la nube o "Cloud".

Es así que, muchas empresas reconocidas, como Netflix, Apple, Instagram, etc., utilizan bases de datos NoSQL, en algunos de sus tantos servicios y/o aplicaciones, por el hecho de cubrir necesidades tales como la escalabilidad, rendimiento y mantenimiento que no encontraban en ninguna solución que existía en el mercado.

2.1 Definición y Concepto de las Bases de Datos NoSQL

Existen diversas interpretaciones del término "NoSQL", por ejemplo, "NO STRUCTURED QUERY LANGUAGE" (No Lenguaje de Consulta Estructurada), que en sus orígenes se entendió como "No SQL", y luego evolucionó para entender el término como "Not Only SQL" (No Solo SQL). Esta última interpretación resalta que éstas bases de datos no buscan reemplazar totalmente a las bases de datos relacionales, sino que ofrecen enfoques alternativos y complementarios.

Se entiende como Base de Datos NoSQL (Not Only SQL, No Solo SQL) a los sistemas de gestión de bases de datos que proporcionan un enfoque alternativo para el almacenamiento y acceso a datos en comparación con las bases de datos relacionales tradicionales. Estos sistemas están diseñados para abordar las limitaciones de las bases de datos relacionales en términos de escalabilidad horizontal, manejo de datos no estructurados y flexibilidad en el esquema de datos. En lugar de depender del modelo de tablas y relaciones, las bases de datos NoSQL utilizan modelos de datos como documentos, grafos, clave-valor o columnas para adaptarse a una variedad de casos de uso y necesidades de desarrollo [17].

Las bases de datos NoSQL presentan modelos de datos específicos con esquemas flexibles diseñados para satisfacer las demandas de las aplicaciones modernas. Están diferenciadas por un conjunto diverso de características y múltiples modelos, pueden considerarse como una categoría particular dentro del panorama de bases de datos. Más adelante se detallará los modelos de datos y características NoSQL.

A continuación, se muestran algunas diferencias entre las bases de datos SQL y NoSQL, las cuales ayudan a comprender más el concepto al que apunta cada una de ellas:

	Bases de datos relacionales	Bases de datos NoSQL
Cargas de trabajo óptimas	Las bases de datos relacionales están diseñadas para aplicaciones de procesamiento de transacciones online (OLTP) altamente coherentes y transaccionales, y son buenas para el procesamiento analítico online (OLAP).	Las bases de datos NoSQL están diseñadas para varios patrones de acceso a datos que incluyen aplicaciones de baja latencia. Las bases de datos de búsqueda NoSQL están diseñadas para hacer análisis sobre datos semiestructurados.
Modelo de datos	El modelo relacional normaliza los datos en tablas conformadas por filas y columnas. Un esquema define estrictamente las tablas, las filas, las columnas, los índices, las relaciones entre las tablas y otros elementos de las bases de datos. La base de datos impone la integridad referencial en las relaciones entre tablas.	Las bases de datos NoSQL proporcionan una variedad de modelos de datos, como clave-valor, documentos y gráficos, que están optimizados para el rendimiento y la escala.
Propiedades ACID	Las bases de datos relacionales ofrecen propiedades de atomicidad, coherencia, aislamiento y durabilidad (ACID): • La atomicidad requiere que una transacción se ejecute por completo o no se ejecute en absoluto. • La coherencia requiere que una vez confirmada una transacción, los datos deban acoplarse al esquema de la base de datos. • El aislamiento requiere que las transacciones simultáneas se ejecuten por separado. • La durabilidad requiere la capacidad de recuperarse de un error inesperado del sistema o de un corte de energía y volver al último estado conocido.	Las bases de datos NoSQL a menudo hacen concesiones al flexibilizar algunas de las propiedades ACID de las bases de datos relacionales para un modelo de datos más flexible que puede escalar horizontalmente. Esto hace que las bases de datos NoSQL sean una excelente opción para casos de uso de baja latencia y alto rendimiento que necesitan escalar horizontalmente más allá de las limitaciones de una sola instancia.
Rendimiento	Normalmente, el rendimiento depende del subsistema de disco. Se necesita la optimización de consultas, índices y estructura de	El rendimiento es, por lo general, depende del tamaño del clúster de hardware subyacente, la latencia de red y la aplicación que efectúa la llamada.

	tabla para lograr el máximo rendimiento.	
Escalado	Las bases de datos relacionales generalmente escalan en forma ascendente las capacidades de computación del hardware o la ampliación mediante la adición de réplicas para cargas de trabajo de solo lectura.	Las bases de datos NoSQL normalmente se pueden particionar porque los patrones de acceso son escalables mediante el uso de arquitectura distribuida para aumentar el rendimiento que proporciona un rendimiento constante a una escala casi ilimitada.
API	Solicita almacenar y recuperar datos que están comunicados mediante consultas que se ajustan a un lenguaje de consulta estructurado (SQL). Estas consultas son analizadas y ejecutadas por la base de datos relacional.	Las API basadas en objetos permiten a los desarrolladores almacenar y recuperar fácilmente estructuras de datos. Las claves de partición permiten que las aplicaciones busquen pares de clave-valor, conjuntos de columnas o documentos semiestructurados que contengan atributos y objetos de aplicación serializados.

Tabla 1: SQL (relacional) en comparación con NoSQL (no relacional) [18]

2.2 Características de las Bases de Datos NoSQL

Si bien existen diferentes tipos de bases de datos NoSQL, cada uno posee características propias que resuelven problemas en diferentes situaciones.

En primer lugar, presentan una estructura distribuida, cuando hablamos de bases de datos distribuidas hablamos de un conjunto de bases que se encuentran lógicamente relacionadas, pero se encuentran en sitios diferentes, interconectadas por una red. La información, por lo tanto, se encuentra repartida entre todas esas bases o nodos. De manera lógica podemos verlas como un solo sistema, pero de manera física se pueden ver como varios sistemas. Así, el usuario podrá acceder a la información de cualquiera de esos nodos como si todos los datos se encontraran en el mismo lugar.

Como segundo lugar hablamos de la escalabilidad horizontal. Podemos decir que pretende afrontar el crecimiento sin perder calidad en los servicios que ofrece, y podemos dividirla en dos tipos:

1. Escalabilidad Horizontal: Se dice cuando un sistema al agregar más nodos mejora el rendimiento.

2. Escalabilidad Vertical: Se dice cuando se agregan más recursos a un nodo del sistema y el sistema en conjunto mejora.

Otra de las principales características que encontramos en estos sistemas, es que no cuentan con una estructura de datos definida, si no que cada tipo NoSQL emplea la propia, al contrario que SQL, el cual estructuraba sus datos en tablas, lo que hace que las primeras sean más flexibles a la hora de almacenar datos.

Cuando hablamos de características de este tipo de bases de datos, nos encontramos el teorema CAP o también conocido como Teorema de Brewer[23], debido que es un teorema relacionado con los sistemas distribuidos. Este teorema nos dice que en sistemas distribuidos es imposible garantizar simultáneamente las siguientes tres propiedades:

1. Consistencia (Consistency): la cual consiste en hacer que todos los nodos del sistema vean la misma información al mismo tiempo, es lo que conocemos como integridad de la información.

2. Disponibilidad (Availability): consiste en garantizar que el sistema se podrá utilizar a pesar de que uno de los nodos se haya caído.

3. Tolerancia al particionamiento (Partition tolerance): también conocido como tolerancia a fallos. Es la capacidad que tiene el sistema de seguir funcionando a pesar de que existan fallos parciales que dividan dicho sistema.

Para conocer mejor este concepto se presenta la siguiente ilustración, en la que aparecen los siguientes términos:

1. AP: Este tipo garantiza tanto la disponibilidad como la tolerancia a particiones, pero no la consistencia.
2. CP: Indica que garantiza la consistencia y la tolerancia.
3. CA: Encontramos aquí problemas con la tolerancia a particiones, ya que garantizan la consistencia y la disponibilidad.

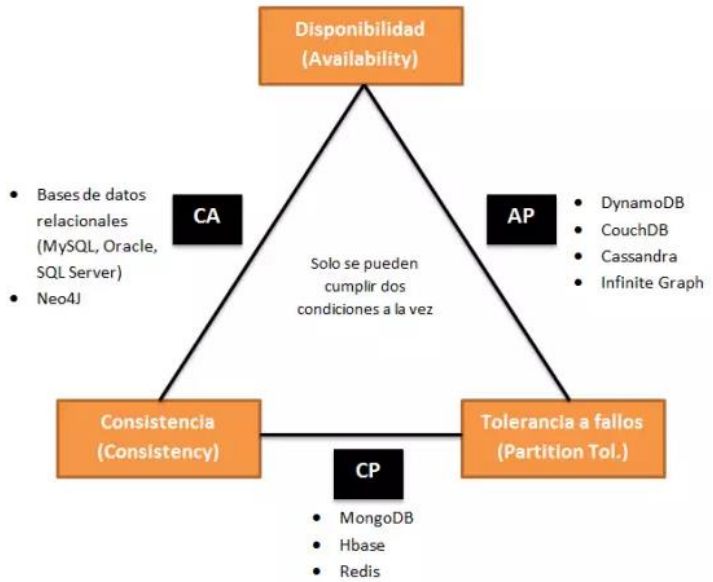

Ilustración 1: Teorema CAP (Consistency, Availability, Partition Tolerance)

Todas estas características nos llevan por lo tanto a obtener una serie de ventajas e inconvenientes en este tipo de sistemas. En cuanto a las ventajas, dado que son sistemas distribuidos, cuentan con la alta velocidad de la que estos disponen, y de una alta capacidad de almacenamiento.

Además, debido al bajo coste computacional que suponen, en comparación con sistemas de tipo SQL, hace que sea también más baratas económicamente, ya que necesitan de menos recursos para llevar a cabo su trabajo. Otra de sus ventajas es que responde muy bien tanto a la activación como desactivación de servicios, por lo que se puede adecuar a las necesidades que se tengan en el momento.

Cuando se habla de inconvenientes, a veces los sistemas NoSQL pueden sufrir pérdidas de datos e inconsistencias.

2.2 Modelos de Datos en Bases de Datos NoSQL

Un modelo de base de datos es la estructura lógica que adopta la base de datos, incluyendo las relaciones y limitaciones que determinan cómo se almacenan y organizan y cómo se accede a los datos. Así mismo, un modelo de base de datos también define qué tipo de operaciones se pueden realizar con los datos, es decir, que también determina cómo se manipulan los mismos, proporcionando también la base sobre la que se diseña el lenguaje de consultas.

En los siguientes puntos, se muestran algunos de los modelos de datos NoSQL más usados.

2.2.1 Bases de Datos Tipo Clave-Valor

Las bases de datos del tipo Clave-Valor (también llamadas a veces como tablas hash) se estructuran almacenando la información como si fuesen un diccionario. El diccionario, en este caso, es un tipo de datos que contienen tuplas clave-valor. Las personas añaden y solicitan valores a partir de una clave asociada que conocen de antemano.

Según Martin Kleppmann [19], "En una base de datos clave-valor, los datos se almacenan en un conjunto de pares clave-valor, donde cada clave es un identificador único y cada valor es una estructura de datos. Los valores no se interpretan: la base de datos no entiende la estructura interna de los datos. Los sistemas de bases de datos clave-valor suelen ofrecer únicamente dos operaciones sobre los valores: almacenar un valor con una clave dada y recuperar el valor almacenado previamente para una clave dada."

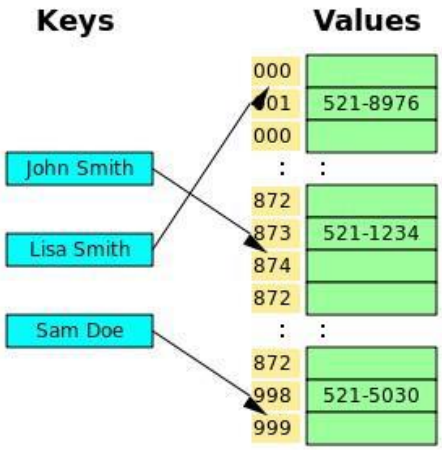

Ilustración 2: Ejemplo de almacenamiento de información en una base de datos tipo clave-valor

Los sistemas modernos de almacenamiento clave-valor se caracterizan por tener una elevada escalabilidad horizontal y un rendimiento muy bueno para volúmenes de datos muy grandes. Su estructura es muy sencilla de utilizar y comprender, aunque su implementación puede llegar a ser compleja.

Cassandra es una de las bases de datos clave-valor más importantes, está desarrollada en Java por Apache. Ofrece ciertas funcionalidades como la consistencia inmediata, la verificación de la integridad de datos o las referencias externas.

2.2.2 Bases de Datos Orientadas a Documentos

Las Bases de Datos Orientadas a Documentos, son otro tipo de base de datos NoSQL y es uno de los modelos más utilizados, con un grado de complejidad y flexibilidad superior a las bases de datos clave – valor.

Una base de datos de documentos es una base de datos diseñada para manejar, almacenar y consultar documentos semiestructurados. Los documentos en una base de datos de documentos son similares a los documentos JSON [formato de notación de objetos JavaScript], pero pueden incluir tipos de datos adicionales, como fechas y binarios. Las bases de datos de documentos son especialmente útiles para el desarrollo de aplicaciones web, donde los datos almacenados pueden ser altamente variados en estructura y contenido [20].

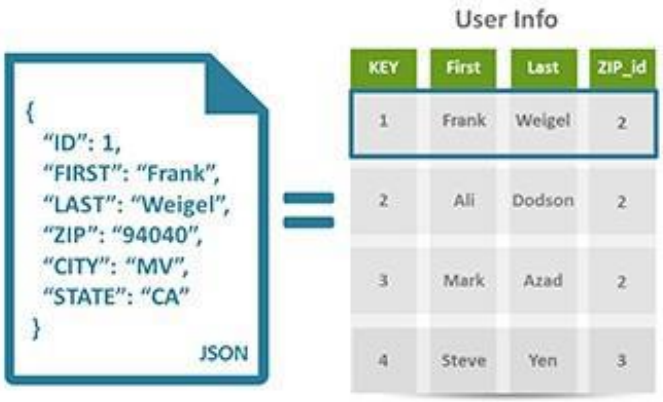

Ilustración 3: Ejemplo de almacenamiento de información en una base de datos documental

Las codificaciones más habituales de estos documentos suelen ser XML, YAML o JSON, pero también se pueden almacenar en formato Word o PDF.

Habitualmente, los documentos se estructuran dentro de una serie de contenedores llamadas colecciones (collections) que son proporcionados por el gestor de base de datos utilizado. Los mismos son almacenados dentro de la base con clave única, por la cual son también recuperados posteriormente con la misma clave.

Se pueden generar índices que afecten a otros campos, pero hay que evaluar bien su construcción, ya que aceleran y mejoran los tiempos de carga por ese campo, pero a cambio tienen necesidades de espacio y de mantenimiento a tener en cuenta.

2.2.3 Bases de Datos Orientadas a Grafos

Las bases de datos orientadas a grafos como su propio nombre indica, almacenan la información en forma de nodos y relaciones entre ellos. Los nodos representan las entidades, y las relaciones vienen representadas como aristas entre dichos nodos, además, dichas relaciones tienen la capacidad de poseer atributos. De este modo, se puede emplear la teoría de grafos para recorrer la base de datos y así gestionar y procesar la información.

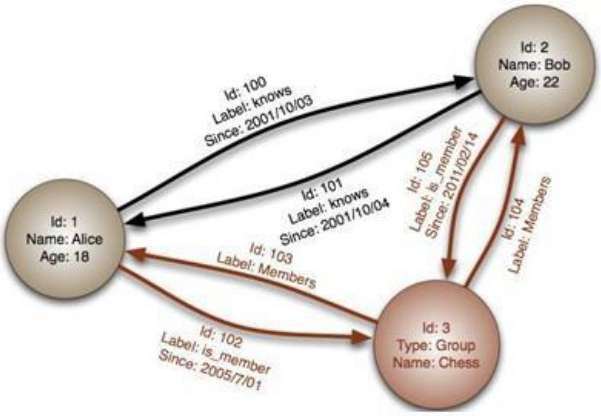

Ilustración 4: Ejemplo de almacenamiento de información con un esquema orientado a grafos

Una base de datos orientada a grafos, de forma generalizada, es cualquier sistema de información donde cada elemento tiene un puntero directo hacia sus elementos adyacentes, es decir, no sería necesario realizar consultas mediante índices.

A diferencia de la mayoría de bases de datos, el rendimiento de una base de datos orientada a grafos no se deteriora con el crecimiento de la base de datos, ni con consultas o procesamientos muy intensivos de los datos. Esto es debido a que el rendimiento será siempre proporcional al tamaño y el rendimiento que tenga el recorrer la parte del grafo que esté implicada, y no el tamaño global del grafo.

El modelo de grafos proporciona una elevada flexibilidad para manipular el conjunto de datos, ya que le permite conectar los datos de forma que posteriormente sea sencillo realizar consultas o actualizaciones desde un sistema/aplicación. También nos permiten realizar un mantenimiento progresivo y ágil de los sistemas, a medida que la aplicación va creciendo, de forma que se puede adaptar a las nuevas necesidades del negocio.

2.2.4 Bases de Datos Orientadas a Columnas

Las bases de datos orientadas a columnas son otro caso particular de la enorme familia de las bases de datos NoSQL. En este tipo de almacenamiento, de forma contraria al modelo relacional, los datos se almacenan en columnas, en lugar de almacenarse en filas.

Los datos están compuestos por una o más familias de columnas que se agrupan de forma lógica en determinadas columnas en la base de datos. Una clave se utiliza para identificar y señalar a un número de columnas en la base de datos. Cada columna contiene las filas de nombres o tuplas, valores, ordenados y separados por comas.
Las filas en cada familia de columnas no requieren tener las mismas columnas, por lo que las columnas pueden ser agregadas a una sola fila o a todas las filas de la familia de columnas.

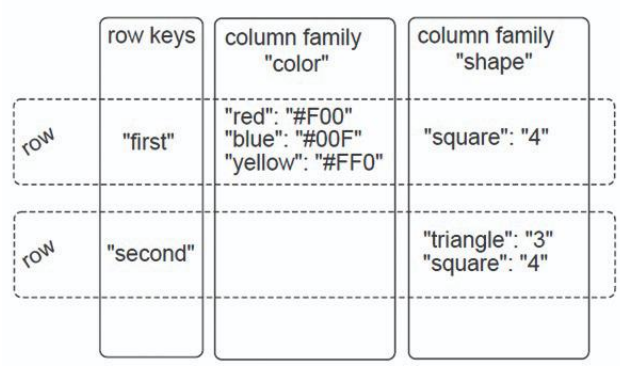

Ilustración 5: Ejemplo de almacenamiento de información con una base de datos orientada a columnas

Las bases de datos orientadas a columnas tienen acceso rápido de lectura y escritura a los datos almacenados. Las filas que corresponden a una sola columna se almacenan como una sola entrada de disco, lo cual facilita el acceso durante las operaciones de lectura y escritura. Dicho de otra forma, una base de datos en columnas está optimizada para lograr una recuperación rápida de columnas de datos, normalmente en aplicaciones analíticas.

Las comparativas de rendimiento entre los sistemas de gestión por filas (especialmente RDBMS) y los basados en columna, vienen principalmente de la mano de la eficiencia de los accesos que realizan a disco. Los accesos a posiciones consecutivas se producen en una cantidad de tiempo sensiblemente menor que los accesos que se producen a posiciones aleatorias a disco.

Los sistemas de almacenamiento basados en columnas proporcionan un mejor rendimiento cuando es necesario realizar una transformación de datos que involucra a todas o gran parte de las filas, pero solo a una o una pequeña parte de las columnas.

Una desventaja de almacenar datos en columnas es que los datos de una entidad se fragmentan entre varias columnas; por lo tanto, la inserción y la actualización o lectura del contenido completo de una entidad puede ser más lenta y compleja que en una base de datos relacional [22].

2.3 Base de datos MongoDb

2.3.1 ¿Qué es MongoDb?

MongoDB es un sistema de gestión de bases de datos (DBMS) NoSQL, está orientada a documentos flexibles en lugar de tablas y filas para procesar y almacenar diversas formas de datos.

Es de código abierto y fue desarrollado en C++, en lugar de guardar los datos en tablas lo hace en estructuras de datos BSON (similar a JSON) con un esquema dinámico.

```
{

    Nombre: "Jonathan",
    Apellidos: "Romano",
    Edad: 37,
    Aficiones: ["Música","Baile","Viaje"],
    Amigos: [
        {
          Nombre:"Alberto",
          Edad:38    },
        {
          Nombre:"Sergio",
          Edad:37
        }
    ]
}
```

Ilustración 6: Ejemplo del formato BSON similar a JSON para el almacenamiento de datos

2.3.2 Características de MongoDb

a) <u>Flexibilidad:</u> Almacena los datos en documentos JSON y estos ofrecen un modelo de datos que se asigna a la perfección a los tipos de lenguaje de programación nativa y el esquema

dinámico que hace que sea más fácil para evolucionar su modelo de datos a comparación con un sistema de esquemas forzados como lo son los manejadores de bases de datos relacionales (RDBMS).

b) Replicación: Del mismo modo, la replicación es un proceso básico en la gestión de bases de datos. MongoDB soporta el tipo de replicación primario-secundario. De este modo, mientras podemos realizar consultas con el primario, el secundario actúa como réplica de datos en solo lectura a modo copia de seguridad con la particularidad de que los nodos secundarios tienen la habilidad de poder elegir un nuevo primario en caso de que el primario actual deje de responder.

c) Indexación: El concepto de índices en MongoDB es similar al de bases de datos relacionales, con la diferencia de que cualquier campo documentado puede ser indexado y añadir múltiples índices secundarios.

d) Balanceo de carga: Puede ejecutarse en varios servidores, equilibrando la carga y/o la duplicación de datos para mantener el sistema en funcionamiento en caso de fallo de hardware. La configuración automática es fácil de implementar, y las nuevas máquinas se puede agregar a una base de datos en ejecución.

e) Almacenamiento de archivos: Aprovechando la capacidad de MongoDB para el balanceo de carga y la replicación de datos, MongoDB puede ser utilizado también como un sistema de archivos. Esta funcionalidad, llamada GridFS e incluida en la distribución oficial, permite manipular archivos y contenido.

f) Ejecución de JavaScript del lado del servidor: MongoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

g) Consultas ad hoc: Soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Además, estas consultas pueden devolver un campo específico del documento, pero también puede ser una función JavaScript definida por el usuario.

2.3.3 Arquitectura de MongoDb

El manejador de bases de datos MongoDB proporciona tecnología de fragmentación como solución para escalar horizontalmente [24]. La tecnología de fragmentación es un sistema de clúster de bases de datos que se utiliza para expandir horizontalmente datos masivos, y los datos se dividen y almacenan en diferentes nodos de datos para manejar mayores cargas de datos. Esto permite reducir la presión sobre el rendimiento de una sola máquina causada por un gran volumen de datos y aplicaciones de alto rendimiento, y mejorar el rendimiento del acceso aleatorio en un gran volumen de datos. Al almacenar datos masivos, es posible que un servidor no sea suficiente para almacenar los datos y proporcionar un rendimiento de lectura y escritura aceptable. Al establecer un clúster, los datos se pueden dividir y almacenar en varias máquinas, de modo que el sistema de base de datos pueda almacenar y procesar más datos para satisfacer las necesidades de procesamiento del gran crecimiento de datos.

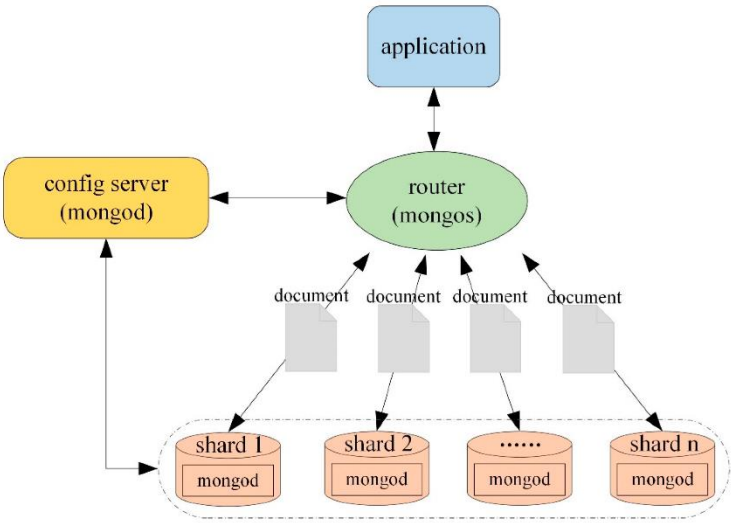

Ilustración 7: Arquitectura de Base de datos MongoDb

La arquitectura del clúster MongoDB incluye principalmente tres partes: el servidor de fragmentos (Shared N), el servidor de enrutamiento (Router Mongos) y el servidor de configuración (Config Server). Los datos reales se almacenan en el servidor de fragmentos, que puede ser un conjunto de réplicas o un servidor. El servidor enrutador (router mongo) es la entrada que los clientes solicitan en la base de datos y se utiliza para abordar y localizar las solicitudes de estos clientes. Todas estas solicitudes deben ser manejadas por los router mongos y reenviadas al servidor de fragmentos correspondiente como se muestra en la ilustración 7. El servidor de configuración se utiliza para almacenar la información de configuración del enrutador y el fragmento, que se configura al principio y no necesita mucho espacio ni recursos.

2.4 Base de datos Cassandra

2.4.1 ¿Qué es Cassandra?

Apache Cassandra es un sistema de gestión de bases de datos (DBMS) distribuida, pertenece a la familia de bases de datos NoSQL y basada en el modelo de almacenamiento de clave-valor. Está desarrollada en Java, y se destaca por su enfoque en la escalabilidad horizontal y la tolerancia a fallos. Ha sido diseñado para abordar los desafíos asociados con el manejo de grandes volúmenes de datos distribuidos en entornos donde la disponibilidad y el rendimiento son críticos.

Lo que distingue a Cassandra es su capacidad para manejar cargas de trabajo intensivas en escritura y lectura en entornos distribuidos y dinámicos. Su modelo de datos basado en columnas facilita la recuperación eficiente de conjuntos de datos específicos y su enfoque en la escalabilidad horizontal permite agregar más nodos al sistema para aumentar su capacidad de manejo de datos.

Al no requerir un esquema fijo, Cassandra ofrece flexibilidad en la forma en que los datos pueden ser almacenados, lo que es especialmente útil en aplicaciones donde la estructura de los datos puede cambiar con el tiempo. Además, su lenguaje de consulta CQL (Contextual Query

Language) proporciona una interfaz similar a SQL (Structured Query Language), simplificando la interacción con la base de datos.

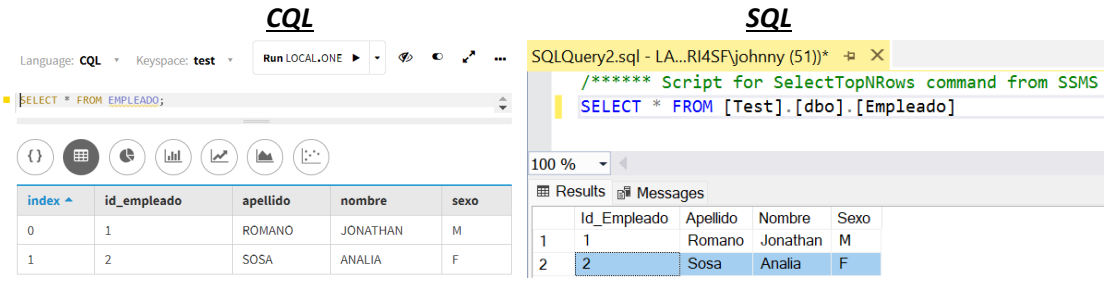

Ilustración 8: Similitud de consulta CQL (Cassandra) y SQL (SQL SERVER)

La tolerancia a fallos es fundamental en los entornos distribuidos, y Cassandra aborda este aspecto mediante la replicación de datos en varios nodos. Esto significa que incluso si un nodo falla, los datos están disponibles en otros nodos, garantizando la disponibilidad continua del sistema.

2.4.2 Características de Cassandra

a) Distribuido: Cassandra distribuye los datos dinámicamente entre los nodos del clúster, evitando la sobrecarga en nodos individuales y garantizando un rendimiento equitativo, siendo éste transparente para el usuario. La distribución equitativa de carga facilita el balanceo a medida que se agregan o eliminan nodos del clúster. No se requieren redistribuciones manuales complejas.

b) Descentralizado: Es un sistema que ha sido pensado de forma que no exista ningún punto de fallo único. Todos los nodos en Cassandra desempeñan la misma funcionalidad, no existe nodo maestro, ni nodos que coordinen las actividades que desempeñan los demás. La autonomía de los nodos significa que cada nodo puede funcionar de manera independiente. Si un nodo experimenta problemas, los demás pueden seguir trabajando sin depender críticamente de ese nodo específico. Por tanto, la descentralización proporciona principalmente dos beneficios: funcionalmente es más sencillo que los sistemas maestro – esclavo, ayudando a evitar pérdidas en el servicio.

c) Escalable: La escalabilidad podría ser definida como la capacidad de un sistema para, ante un incremento considerable de la demanda en un corto espacio de tiempo, continuar ofreciendo el servicio sin degradar la calidad del mismo.
Cassandra permite escalar horizontalmente al agregar nuevos nodos al clúster. Esto es crucial para manejar el crecimiento en volumen de datos y la carga de trabajo. La adición de nodos es relativamente sencilla y no requiere cambios significativos en la arquitectura existente.
Al distribuir la carga entre varios nodos, evita la saturación de recursos en nodos individuales, esto garantiza un rendimiento equitativo y eficiente, incluso a medida que el sistema crece.

d) Alta Disponibilidad: Cassandra almacena múltiples copias de los datos en varios nodos. Esto proporciona redundancia, de modo que, si un nodo falla, los datos todavía están disponibles en otros nodos. La probabilidad de pérdida de datos debido a la falla de un solo nodo se reduce significativamente. La arquitectura distribuida garantiza que incluso en presencia de fallas, el sistema pueda continuar operando sin interrupciones. Los clientes pueden acceder a los datos desde nodos alternativos.

e) Tolerante a fallos: Si en algún nodo surge un error de comunicación, el resto puede seguir funcionando sin problemas, y en caso de que sea necesario, puede agregar un nuevo nodo en lugar del que está fallando. Si el error es de hardware, el nodo puede ser apartado hasta que se solucione el problema. Llegado al caso, puede incluso ser totalmente excluido del clúster y sustituido por otro nuevo.
Al replicar datos en varios nodos, Cassandra asegura que la pérdida de un nodo no resulte en la pérdida de datos. La consistencia configurable permite ajustar la tolerancia a fallos según las necesidades de la aplicación.

2.4.3 Arquitectura de Cassandra

La arquitectura de Cassandra permite la distribución transparente de datos a los nodos. Esto significa que puede determinar la ubicación de sus datos en el clúster en función de los datos. Cualquier nodo puede aceptar cualquier solicitud ya que no hay maestros ni esclavos. Si un nodo tiene los datos, los devolverá. De lo contrario, enviará la solicitud al nodo que tiene los datos.

Puede especificar el número de réplicas de los datos para lograr el nivel requerido de redundancia. Por ejemplo, si los datos son muy críticos, es posible que desee especificar un factor de replicación de 4 o 5.

Si los datos no son críticos, puede especificar solo dos. También proporciona coherencia ajustable, es decir, el nivel de coherencia se puede especificar como una compensación con el rendimiento. Las transacciones siempre se escriben en un registro de confirmación en el disco para que sean duraderas.

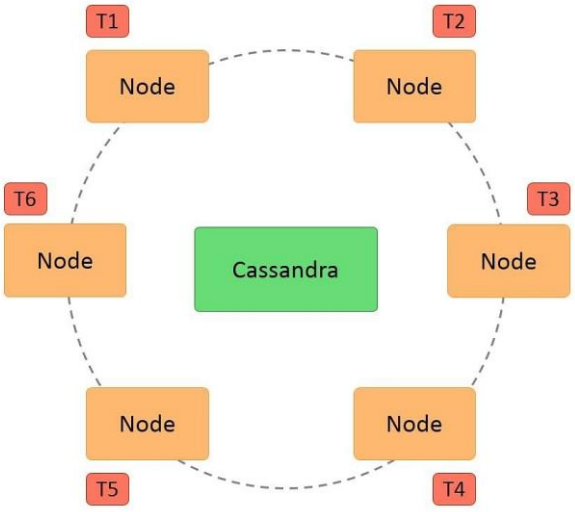

Ilustración 9: Arquitectura de Base de datos Cassandra

Capítulo 3: YCSB

3.1 ¿Qué es YCSB?

YCSB[9] (Yahoo Cloud System Benchmark) es un software de código abierto para medir el rendimiento y escalabilidad de diferentes sistemas gestores de bases de datos.

Fue desarrollado por Brian F. Cooper en Yahoo! Research, con el objetivo de evaluar y comparar el rendimiento de diferentes sistemas de almacenamiento distribuido.

El objetivo principal de YCSB es proporcionar un estándar para la evaluación del rendimiento de sistemas de almacenamiento distribuido y bases de datos NoSQL. El conjunto de herramientas incluye una serie de cargas de trabajo (workloads) que simulan sistemas reales y operaciones habituales, como la lectura y escritura de datos.

Algunas de las bases de datos y sistemas que YCSB puede evaluar incluyen Apache Cassandra, Apache HBase, MongoDB, Redis, Couchbase, entre otros. Al ejecutar las pruebas de YCSB en estos sistemas, los desarrolladores y administradores pueden obtener métricas objetivas y comparativas sobre el rendimiento de diferentes soluciones en diversas configuraciones.

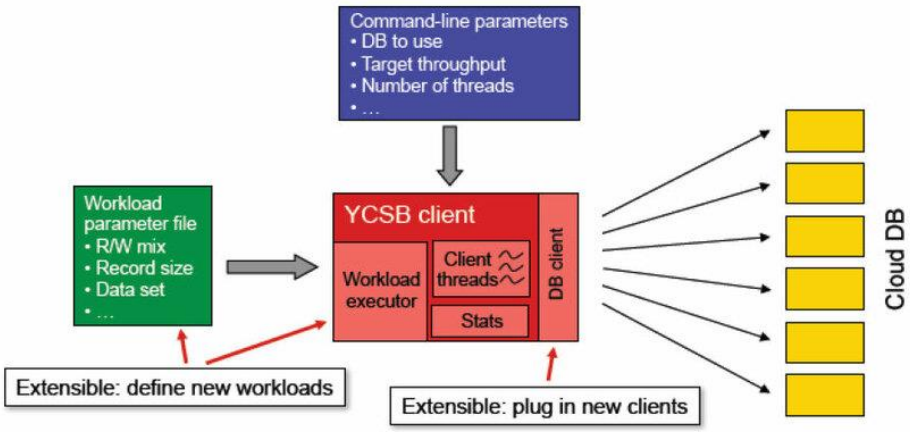

Ilustración 9: Arquitectura extensible del Yahoo Cloud Serving Benchmark (YCSB)

3.2 Características principales de YCSB

YCSB es un sistema diseñado para generar una determinada carga de trabajo a un sistema de gestión de bases de datos, de manera que se pueda medir su rendimiento (latencia y throughput).

Este sistema nos permite trabajar con diferentes bases de datos como, por ejemplo: Cassandra, MongoDb, HBase etc. Funciona generando una carga de trabajo a un número de operaciones por segundo establecido y reportando al final datos sobre la ejecución. Estos datos son:

Tiempo de Ejecución, Número de operaciones ejecutadas, Throughput (Tasa de Transferencia Efectiva) y Latencias mínima, máxima, media y percentiles 90, 95, 99 y 99,9 de cada tipo de operación y número de operaciones completadas con éxito.

3.3 Cargas de trabajo "Workload"

Las pruebas de rendimiento se realizan en base a lo que se conoce como workloads (carga de trabajo). Los diferentes workloads están conformados por una serie de parámetros configurables que determinan el tipo y la distribución de las operaciones de lectura, escritura, actualización y eliminación que se ejecutarán durante las pruebas de rendimiento simulando un trabajo real en la base de datos seleccionada.

- **Workload A**: Consiste en una mezcla equilibrada de operaciones de lectura (50%) y actualización (50%). Representa un caso de uso típico donde las operaciones de lectura y actualización son igualmente importantes.
- **Workload B**: La mayoría de las operaciones son de lectura (95%) con un pequeño porcentaje de operaciones de actualización (5%). Este workload simula un escenario donde la lectura de datos es mucho más común que la actualización.
- **Workload C**: Todas las operaciones son de lectura (100%). Simula un escenario donde las operaciones de actualización están ausentes.
- **Workload D**: Consiste en una combinación de lectura (95%), actualización (0%) e inserción (5%). Simula un caso de uso donde las operaciones de lectura son predominantes, pero ocasionalmente se realizan actualizaciones en los datos.
- **Workload E**: En esta carga de trabajo, se consultan rangos de registros, en lugar de datos individuales registros, escaneo (95%), inserción (5%).
- **Workload F**: En esta carga de trabajo, el cliente leerá un registro, lo modificará y volverá a escribir, lectura (50%) y lectura-actualización-escritura (50%).

Dentro de este contexto, utilizamos dos comandos esenciales para realizar las pruebas de rendimiento, ellos son, **LOAD** y **RUN**. Estos comandos están relacionados a los diferentes workloads, los cuales determinan el tipo y la distribución de las operaciones que se ejecutarán durante las pruebas.

Cuando ejecutamos el comando **LOAD**, estamos cargando datos en la base de datos seleccionada, esto es independiente al tipo de workload que le pasemos por parámetro, ya que solo realizará un proceso de carga de registros.

Finalizada la carga de datos, ejecutamos el comando RUN para realizar las pruebas de rendimiento. A diferencia del comando LOAD, para este comando debemos especificar que workload deseamos utilizar para la prueba. Por ejemplo, podríamos elegir el workload B para simular un escenario donde la lectura de datos es mucho más común que la actualización.

A su vez los workloads se pueden configurar para realizar las cuatro principales operaciones en bases de datos: lectura, escritura, actualización y lectura por rangos de clave (scan). Además, para la elección de las filas con las que interactuar se necesita definir la distribución de probabilidad que debe emplear el sistema. YCSB dispone de las siguientes distribuciones de probabilidad:

Uniform (Uniforme):

Descripción: Se elige una fila con probabilidad uniforme. Esto significa que todas las filas tienen la misma probabilidad de ser seleccionadas.

Impacto en YCSB: En este caso, cada operación tiene la misma probabilidad de seleccionar cualquier fila de datos. Es útil para simular escenarios en los que todas las filas son igualmente propensas a ser accedidas.

Zipfian (Zipfiana):

Descripción: Se elige una fila según la distribución zipfiana. En esta distribución, algunas filas son muy populares (cabeza de la distribución) mientras que la mayoría no son tan populares (cola de la distribución).

Impacto en YCSB: Se utiliza para simular situaciones en las que algunas filas son más frecuentemente accedidas que otras. Es común en escenarios del mundo real donde hay elementos populares y menos populares.

Latest (Más Recientes):

Descripción: Funciona como la distribución zipfiana, pero en este caso, las filas más recientes son las que se encuentran en la cabeza de la distribución.

Impacto en YCSB: Útil para simular casos en los que las filas más recientes son más propensas a ser accedidas. Puede ser relevante en situaciones donde la temporalidad de los datos es un factor importante.

Multinomial (Multinomial):

Descripción: La probabilidad puede ser especificada para cada elemento. Por ejemplo, se pueden asignar diferentes probabilidades a operaciones específicas, como lecturas, actualizaciones, exploraciones y escrituras.

Impacto en YCSB: Ofrece flexibilidad al permitir la asignación de probabilidades específicas a diferentes tipos de operaciones. Esto puede ser útil para modelar escenarios donde ciertas operaciones son más comunes que otras.

Además de los tipos de operación que se van a realizar y la distribución de probabilidad, se pueden especificar el número de hilos cliente que van a generar trabajo contra la base de datos, el throughput objetivo de la prueba, el número de operaciones y la duración máxima, entre otros.

Capítulo 4: Experimentación

El presente Trabajo Final Integrador desarrolla la comparativa entre dos bases de datos No SQL, MongoDB y Cassandra, describiendo en los primeros capítulos la parte teórica de los temas tratados.

En este capítulo se despliegan las herramientas necesarias para poder realizar la comparativa, todo sobre ambientes virtualizados basados en contenedores. Para ello, se describirá que tipo de software se utiliza en la presente experimentación.

4.1 Virtualización basada en contenedores

Los contenedores son una forma de virtualización del sistema operativo. Un solo contenedor se puede usar para ejecutar cualquier cosa, desde un microservicio o un proceso de software a una aplicación de mayor tamaño. Dentro de un contenedor se encuentran todos los ejecutables, las bibliotecas y los archivos de configuración necesarios. Sin embargo, en comparación con las máquinas o servidores virtuales, los contenedores no contienen imágenes del sistema operativo. Esto los hace más ligeros y portátiles, con una sobrecarga significativamente menor.

La mayor ventaja de este enfoque consiste en el ahorro de costos, que se obtiene al ejecutar múltiples instancias virtuales en el mismo hardware, permitiendo una mejora en la utilización de sus recursos. Como se menciona en el párrafo anterior, otra ventaja es el aislamiento que ofrece, las aplicaciones pueden requerir diferentes versiones de sus dependencias y bibliotecas, para lo cual, cada contenedor tendría sus propios recursos para que la aplicación funcione sin entrar en conflicto con el mismo host.

En implementaciones de aplicaciones de mayor tamaño, se pueden poner en marcha varios contenedores, y así mismo esos contenedores pueden estar en uno o varios clústeres de contenedores. En ese punto y debido a su amplio despliegue, se recomienda usar un orquestador de contenedores como, por ejemplo, Kubernetes, herramienta que podría ponerse a pruebas en investigaciones futuras.

A continuación, se muestra dos tipos básicos de virtualización, por un lado, contenedor, y por otro hipervisor.

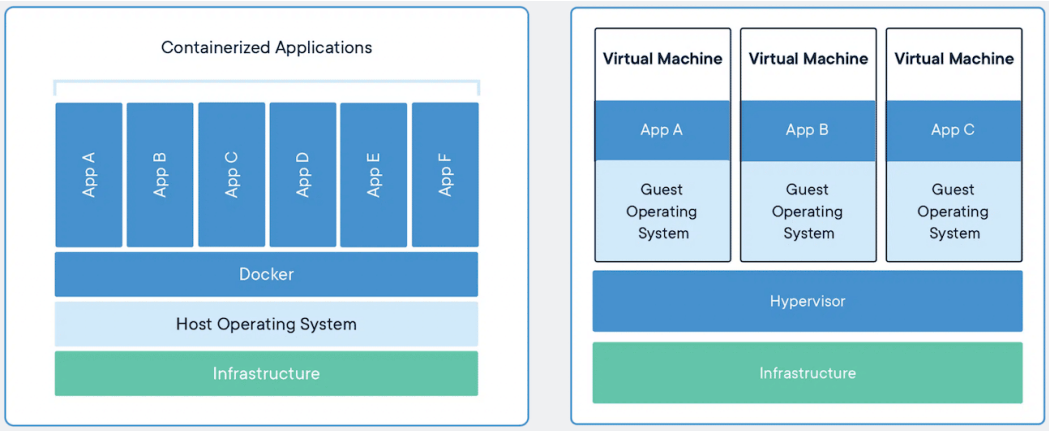

Ilustración 10: Arquitectura de virtualización basada en Contenedores e Hipervisores.

4.2 ¿Qué es Docker y cómo funciona?

Docker es un software de código abierto que nos permite crear, probar e implementar aplicaciones rápidamente. Empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que nuestra aplicación se ejecute.

Implementa y ajusta la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

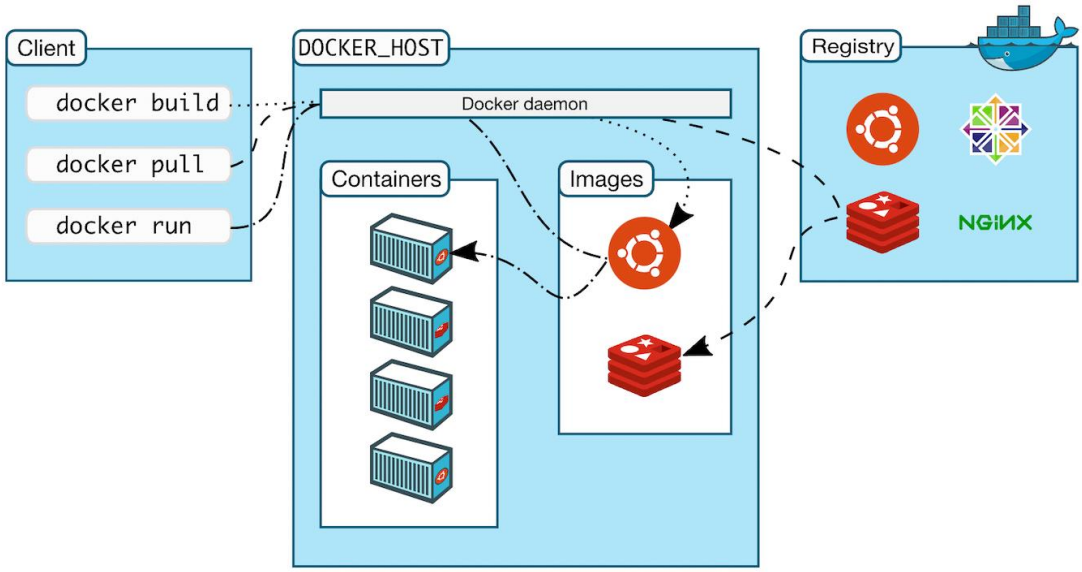

Ilustración 11: Diagrama Docker Engine

Docker Engine es la tecnología cliente-servidor para construir y contenerizar aplicaciones en Docker [6]. Esencialmente, soporta todas las tareas relacionadas con la ejecución de una aplicación basada en contenedores:

Estos son los principales componentes del motor Docker:

- **Docker Daemon:** Gestiona las imágenes Docker, los contenedores, las redes y los volúmenes. También escucha las peticiones de la API de Docker y las procesa.
- **API REST del motor Docker**: Una API desarrollada por Docker que interactúa con el demonio.
- **Docker CLI (Client)**: La interfaz de línea de comandos para comunicarse con el demonio Docker.

4.2.1 Dockerfile

Es un archivo de texto plano que contiene una lista de instrucciones que Docker utiliza para construir una imagen Docker. Estas instrucciones incluyen comandos para instalar software, configurar el entorno, copiar archivos y configurar el contenedor. Básicamente, el Dockerfile especifica cómo se debe construir la imagen Docker.

4.2.2 Imágen Docker

Una imagen Docker es una plantilla de solo lectura con instrucciones para crear un contenedor Docker. A menudo, una imagen se basa en otra imagen, con alguna personalización adicional. Por ejemplo, si necesitamos construir una imagen de un servidor web, podemos incluir primero Ubuntu Linux, luego, poner en capas la codificación de Apache y PHP, así como los detalles de configuración necesarios para ejecutar la aplicación.

4.2.3 Contenedor

Cuando hablamos de contenedores nos referimos a un paquete ligero, independiente y ejecutable de una pieza de software que contiene todas las bibliotecas, archivos de configuración, dependencias y otras partes necesarias para ejecutar la aplicación.

Los contenedores Docker son las instancias en ejecución creadas e implementadas a partir de imágenes de Docker y los recursos del sistema asignados. Cabe recordar que se pueden lanzar varios contenedores en el mismo host, y todos están aislados entre sí y actúan por separado.

4.2.4 Docker-compose

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones Docker multicontenedor de forma más sencilla. Permite definir las configuraciones de varios contenedores en un archivo YAML (denominado "docker-compose.yml") en lugar de tener que definirlos y ejecutarlos uno por uno con comandos individuales de Docker [25].

Con Docker Compose, puedes especificar los servicios, las redes y los volúmenes que componen tu aplicación, así como configurar sus relaciones y dependencias. Además, Docker Compose simplifica la administración de las aplicaciones, permitiendo la creación, el inicio y el apagado de todos los contenedores de una aplicación con un solo comando.

4.3 Hardware y software de prueba

Para las pruebas se usó:

- Un equipo host con procesador 11th Gen Intel(R) Core(TM) i5-1135G7 2.40GHz, con 16 GB de memoria RAM y con un sistema operativo Windows 11 de 64 bits.
- Software Oracle VM VirtualBox 7.0.
    - Máquina virtual, con procesador 11th Gen Intel(R) Core(TM) i5-1135G7 2.40GHz, con 6 GB de memoria RAM y con un sistema operativo Ubuntu 22.04.4 LTS de 64 bits.
- Docker, versión “***20.10.21”***.
- Docker-Compose, versión ***“1.29.2”.***
- YCSB (Yahoo! Cloud Serving Benchmark), versión ***“0.17.0”.***
- Base de datos MongoDB, versión ***“4.4.18”***. Versión del gestor MongoDb-Compass ***“1.26.1”.***

- Base de datos Cassandra, versión ***"6.8.35".*** Versión del gestor Datastax/Dse-Studio ***"6.8.26"***.

El diagrama con el hardware y software propuesto para la experimentación se muestra en la ilustración 12.

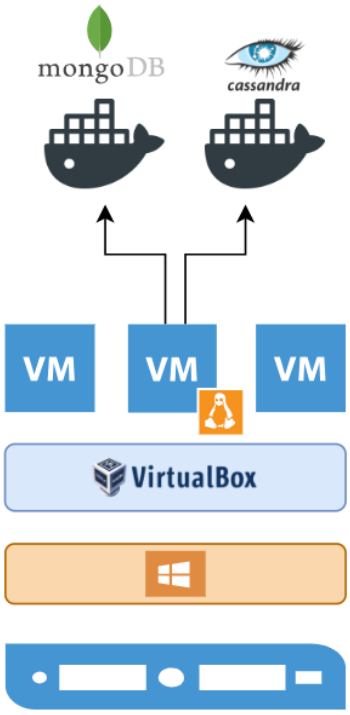

Ilustración 12: Diagrama de la experimentación propuesta

4.4 Dataset de prueba

Como se mencionó en el punto 1.2 Objetivo, el dataset se obtiene opcionalmente del sistema YCSB, la cual genera una cantidad N de registros que le podemos pasar por parámetro. Cada registro está compuesto por una clave primaria y una serie de campos que contienen valores aleatorios.

Estructura del dataset:

- Clave primaria: cada registro se identifica mediante una clave única generada aleatoriamente, una cadena como "user2837293432". Esta clave será utilizada para las operaciones de lectura, escritura y actualización de datos.

- Campos: cada registro contiene 10 campos denominados secuencialmente como "field1", "field2", …, "field10", etc. Los valores de cada campo son caracteres aleatorios y cada campo tiene una longitud fija de 100 bytes, lo que resulta en un tamaño de 1 kb por registro.

- Colecciones: En nuestro caso, utilizamos una única colección (o tabla, dependiendo del motor de almacenamiento utilizado) para almacenar los datos generados por YCSB.
- Índices:
  Para mejorar el rendimiento en las operaciones de búsqueda y actualización, se realizaron pruebas utilizando tanto una colección indizada como una no indizada.
    - El índice se construye sobre la clave primaria (userXXXXXX), lo que permite una recuperación rápida de registros basados en esta clave.
    - En el caso de la colección no indizada, las búsquedas y actualizaciones requieren un escaneo completo de los registros, lo que afecta el tiempo de ejecución.

Dado que el cliente y el servidor están alojados en el mismo nodo, la latencia no formará parte de este estudio. YCSB proporciona configuración de subprocesos y un conjunto de números de operación por subproceso. Durante las pruebas iniciales observamos que, al usar subprocesos, la cantidad de operaciones por segundo en realidad se reducía. Esto se debe al hecho de que las pruebas se ejecutan en una máquina virtual con recursos incluso menores que un host.

4.5 Resultados

4.5.1 Test de Carga (Load)

El comando load utiliza para cargar datos en la base de datos antes de ejecutar un workload. Independientemente del tipo de workload que se utilice, el comando load solo inserta un conjunto de registros predefinidos en el sistema de base de datos, preparándolo para las pruebas de rendimiento. Durante este proceso, YCSB crea un número determinado de registros, con una estructura específica (por ejemplo, clave-valor), que simula los datos que serán utilizados en las operaciones de lectura y escritura de los workloads posteriores.

Como primer paso se realizaron las cargas de datos con el comando LOAD, y el workload A (es indistinto el workload), dichas cargas fueron de 100.000, 200.000, 300.000, 700.000 y 1.000.000 de registros cada una, tal como se muestra en la siguiente ilustración.

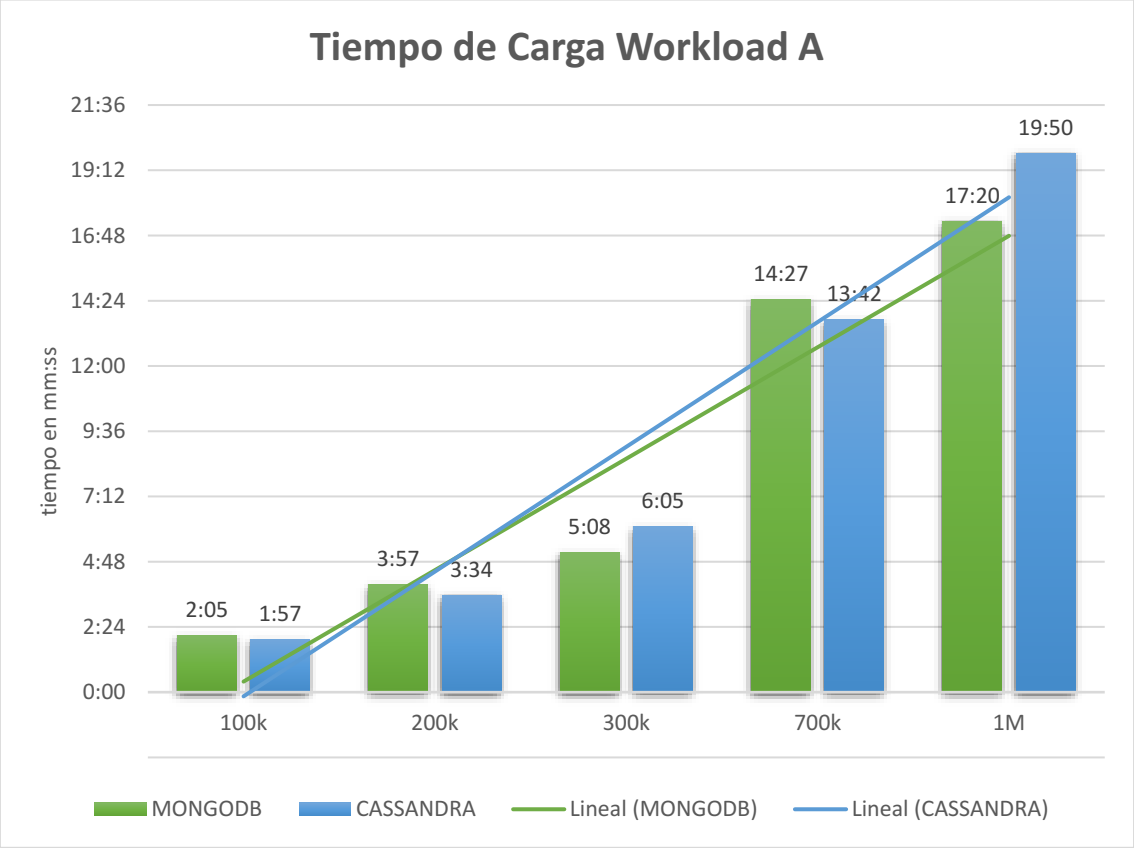

Ilustración 13: Cantidad de registros y tiempo de carga en minutos y segundos Workload A.

Al comparar los tiempos de carga entre MongoDB y CassandraDB para diferentes volúmenes de datos, se observan tendencias que destacan los puntos fuertes y débiles de cada base de datos según la cantidad de registros.

Para cargas pequeñas, CassandraDB muestra un rendimiento ligeramente superior, completando la carga de 100.000 registros en 1:57 minutos frente a los 2:05 minutos de

MongoDB. Una tendencia similar se observa con 200.000 registros, donde CassandraDB logra una ventaja leve con un tiempo de 3:34 frente a 3:57 de MongoDB. Esto refleja la eficiencia de CassandraDB en escrituras rápidas y su capacidad de manejar datos distribuidos con baja latencia.

Para el rango de los 300k y 700k, el rendimiento varía entre las dos tecnologías. Con 300.000 registros, MongoDB supera a CassandraDB, cargando los datos en 5:08 minutos frente a los 6:05 minutos de CassandraDB. Sin embargo, al aumentar a 700.000 registros, CassandraDB se adelanta nuevamente, registrando un tiempo de carga de 13:42, ligeramente mejor que los 14:27 de MongoDB. Este comportamiento sugiere que MongoDB maneja mejor volúmenes moderados de datos, mientras que CassandraDB mantiene una ventaja en cargas distribuidas más sostenidas.

Cuando el volumen de datos alcanza un millón de registros, MongoDB demuestra un rendimiento más consistente, completando la carga en 17:20 minutos, mientras que CassandraDB toma 19:50 minutos. Esto puede atribuirse a la capacidad de MongoDB para optimizar sus operaciones de escritura a medida que el tamaño de los datos crece, mientras que CassandraDB podría experimentar un aumento en la latencia debido a la replicación y sincronización entre nodos en su arquitectura distribuida. Esto indica que, bajo las condiciones de prueba, MongoDB es marginalmente más rápido que Cassandra al gestionar volúmenes de datos crecientes en esta escala específica.

4.5.2 Test de Ejecución (Run)

En estos siguientes test, vamos a enfocarnos en las ejecuciones de los workloads A, B, C y D, ya que estos cubren las operaciones esenciales que utilizamos comúnmente (lecturas, escrituras y lecturas-escrituras mixtas) y proporcionan información representativa del rendimiento para la mayoría de los sistemas.

A diferencia de los workloads E y F, que están diseñados para escenarios específicos que no forman parte de este trabajo final integrador. En particular, el workload E se centra en consultas analíticas complejas y en patrones de acceso a datos menos frecuentes en nuestras aplicaciones típicas, mientras que el workload F simula patrones de acceso de usuario altamente variables, útiles en entornos con cambios dinámicos y heterogéneos en la carga, algo que tampoco es relevante para nuestras necesidades actuales.

4.5.2.1 Workload A

Workload A simula una carga de trabajo de mezcla equilibrada entre operaciones de lectura y escritura, típica de aplicaciones transaccionales. Este perfil realiza un 50% de operaciones de lectura y un 50% de escrituras, lo que lo convierte en un escenario representativo para aplicaciones que requieren tanto consultar datos existentes como actualizarlos frecuentemente, como sistemas de gestión de usuarios o catálogos en línea. Al ejecutar el comando run en YCSB con el workload A, se realizan estas operaciones de manera concurrente y aleatoria sobre los datos previamente cargados, midiendo métricas clave como latencia y rendimiento bajo la configuración establecida. Esto permite evaluar cómo la base de datos maneja el acceso simultáneo a los datos y los costos asociados con cada tipo de operación.

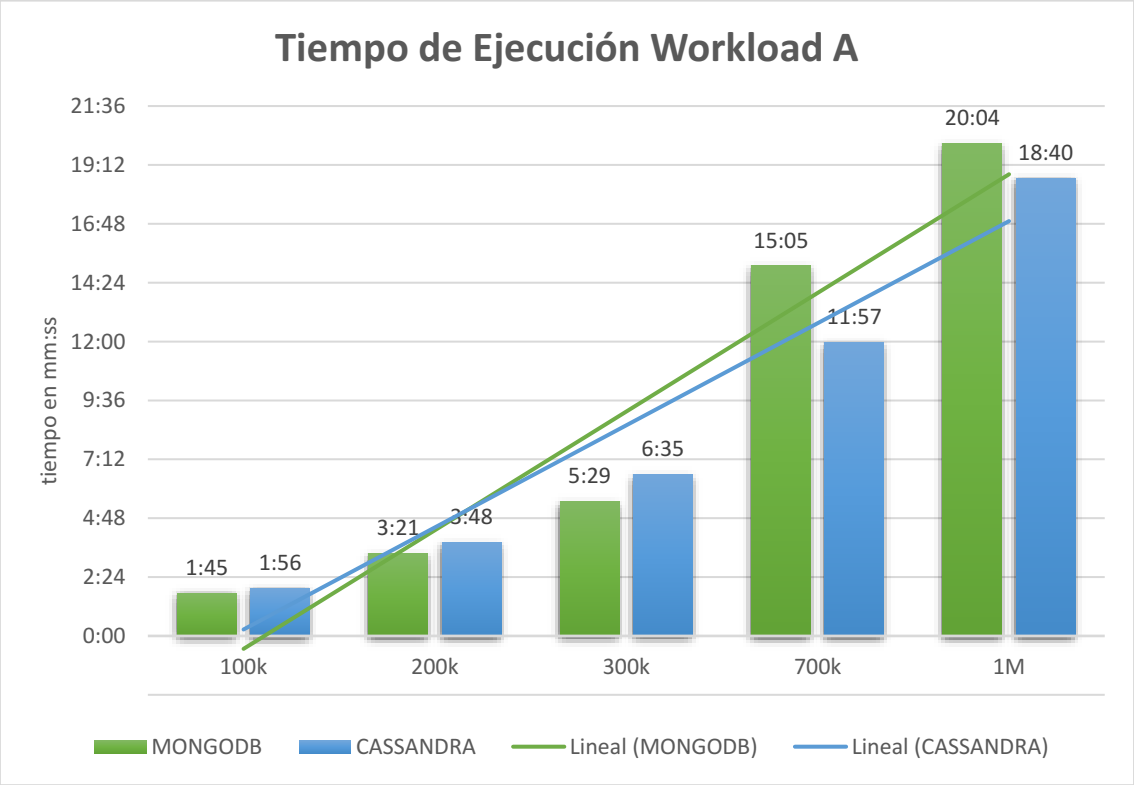

Ilustración 14: Cantidad de registros y tiempo de ejecución en minutos y segundos Workload A.

Al analizar los tiempos de ejecución del workload A para diferentes volúmenes de registros, se pueden identificar patrones clave que resaltan el rendimiento de MongoDB y CassandraDB en términos de eficiencia y escalabilidad.

Para cargas de 100.000 registros, MongoDB supera a CassandraDB con un tiempo de ejecución de 1:45 minutos, frente a los 1:56 minutos de CassandraDB. Una tendencia similar se observa para 200.000 registros, donde MongoDB mantiene su ventaja con un tiempo de 3:21 minutos frente a los 3:48 minutos de CassandraDB. Este comportamiento sugiere que MongoDB maneja mejor las operaciones en lotes pequeños debido a su capacidad para optimizar las consultas y operaciones locales.

Para 300.000 registros, MongoDB completa la ejecución en 5:29 minutos, mientras que CassandraDB lo hace en 6:35 minutos. Sin embargo, para 700.000 registros, CassandraDB demuestra un mejor rendimiento, registrando un tiempo de 11:57 minutos en comparación con los 15:05 minutos de MongoDB. Este cambio refleja la fortaleza de CassandraDB en escenarios donde su arquitectura distribuida permite un procesamiento más eficiente a medida que aumentan los volúmenes de datos.

En el caso de cargas de 1 millón de registros, CassandraDB mantiene su ventaja, completando la ejecución en 18:40 minutos, frente a los 20:04 minutos de MongoDB. Esto destaca la escalabilidad de CassandraDB para manejar grandes volúmenes de datos con tiempos de ejecución relativamente más bajos en comparación con MongoDB.

4.5.2.2 Workload B

Workload B simula una carga de trabajo orientada principalmente a la lectura, típica de aplicaciones donde se accede frecuentemente a los datos pero se actualizan de manera ocasional, como servicios de redes sociales o catálogos de productos. Este perfil realiza un 95% de operaciones de lectura y un 5% de escrituras, lo que refleja un patrón de acceso de "leer mucho, escribir poco". Al ejecutar el comando run con el workload B, se mide cómo la base de datos maneja cargas con una alta proporción de lecturas, proporcionando métricas de rendimiento y latencia para evaluar su eficiencia en entornos donde predominan las consultas sobre las actualizaciones. Esto ayuda a identificar fortalezas y debilidades en la optimización de las operaciones de lectura y consistencia en los datos.

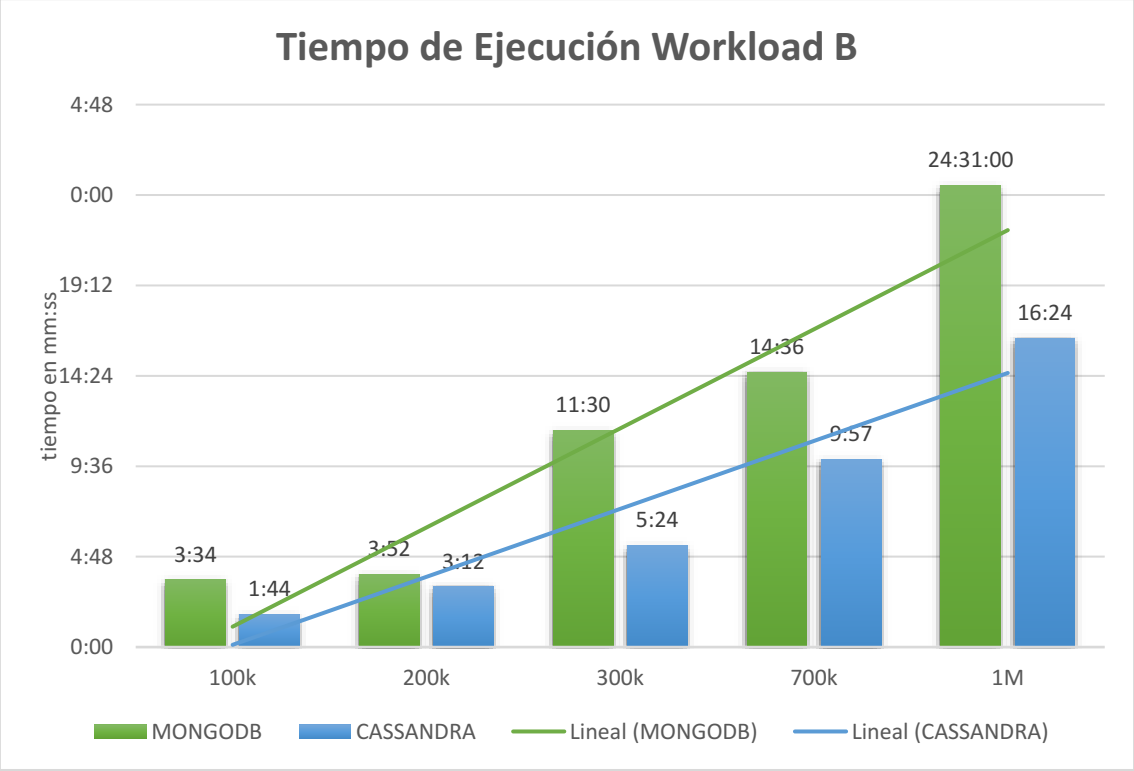

Ilustración 15: Cantidad de registros y tiempo de ejecución en minutos y segundos Workload B.

En cargas pequeñas, CassandraDB demuestra ser más eficiente, completando la ejecución para 100.000 registros en 1:44 minutos, mientras que MongoDB toma 3:34 minutos, más del doble del tiempo. Una tendencia similar se observa con 200.000 registros, donde CassandraDB mantiene una ventaja significativa con un tiempo de 3:12 minutos frente a los 3:52 minutos de MongoDB. Esto evidencia la capacidad de CassandraDB para manejar consultas rápidas en su arquitectura distribuida.

Con 300.000 registros, CassandraDB mantiene su superioridad, logrando un tiempo de ejecución de 5:24 minutos, mientras que MongoDB requiere 11:30 minutos, mostrando una brecha de rendimiento cada vez más amplia. Para 700.000 registros, CassandraDB completa la ejecución en 9:57 minutos, mientras que MongoDB tarda significativamente más, con 14:36 minutos. Este patrón sugiere que CassandraDB escala de manera más eficiente en aplicaciones con predominancia de lecturas.

Para cargas de 1 millón de registros, la diferencia es aún más marcada. CassandraDB completa la ejecución en 16:24 minutos, mientras que MongoDB registra un tiempo extremadamente alto de 24:31 minutos, lo que podría indicar limitaciones en su capacidad para manejar eficientemente grandes volúmenes de consultas simultáneas bajo el perfil de acceso de este workload.

4.5.2.3 Workload C

El workload C una carga de trabajo exclusivamente de lectura, ideal para aplicaciones de solo consulta, como sistemas de búsqueda, portales de contenido estático o dashboards analíticos. Este perfil realiza un 100% de operaciones de lectura, lo que permite evaluar el rendimiento de la base de datos en entornos donde no hay actualizaciones ni escrituras. Al ejecutar el comando run con el workload C, se generan consultas aleatorias sobre los datos previamente cargados, midiendo la latencia y el rendimiento de las lecturas. Esto proporciona una evaluación clara de la capacidad de la base de datos para optimizar las operaciones de lectura en diferentes condiciones de carga.

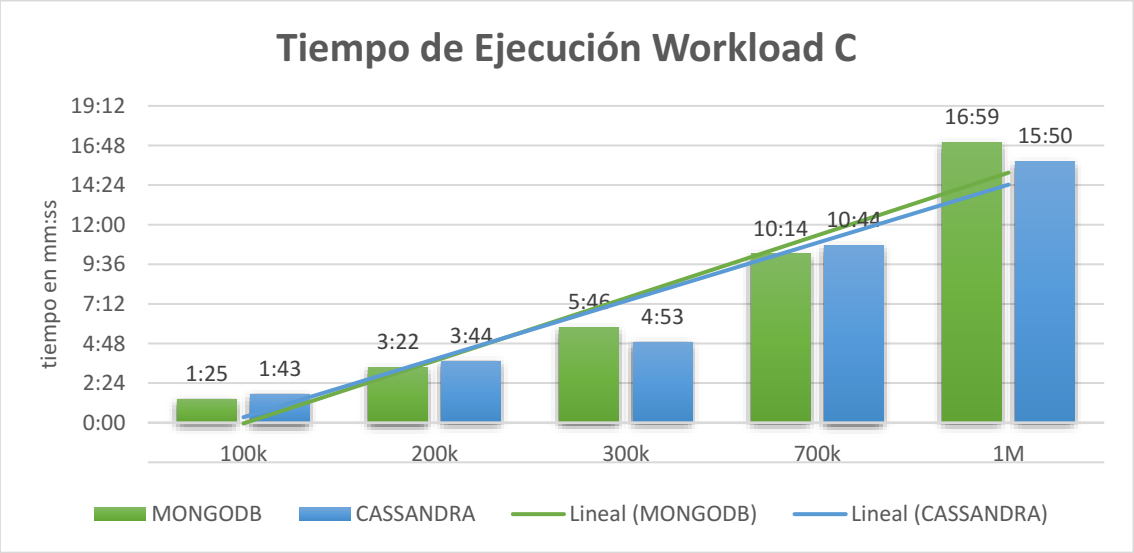

Ilustración 16: Cantidad de registros y tiempo de ejecución en minutos y segundos Workload C.

Este workload se caracteriza por un 95% de operaciones de lectura y un 5% de escrituras, presenta variaciones de rendimiento significativas entre MongoDB y CassandraDB, dependiendo del volumen de datos procesados.

Con 100.000 registros, MongoDB muestra un tiempo de ejecución más rápido (1:25 minutos) en comparación con CassandraDB (1:43 minutos), lo que refleja su capacidad para optimizar consultas en volúmenes pequeños. En el caso de 200.000 registros, MongoDB sigue manteniendo una ventaja ligera, completando la ejecución en 3:22 minutos, frente a los 3:44 minutos de CassandraDB.

A medida que aumenta el volumen de datos, CassandraDB empieza a superar a MongoDB. Para 300.000 registros, CassandraDB registra un tiempo de ejecución de 4:53 minutos, mientras que MongoDB toma 5:46 minutos. Con 700.000 registros, CassandraDB conserva esta ventaja, completando la ejecución en 10:44 minutos, frente a los 10:14 minutos de MongoDB. Este cambio sugiere que CassandraDB maneja mejor la escalabilidad en operaciones de lectura intensiva en volúmenes moderados.

Para 1 millón de registros, CassandraDB vuelve a destacar, logrando un tiempo de ejecución de 15:50 minutos, ligeramente superior al de MongoDB, que toma 16:59 minutos. Esto evidencia que CassandraDB mantiene su rendimiento más consistente a medida que los datos crecen, mientras que MongoDB experimenta una mayor degradación en cargas muy altas.

4.5.2.4 Workload D

El siguiente workload simula una carga de trabajo orientada a accesos recientes, típica de aplicaciones que priorizan el acceso a los datos más nuevos, como sistemas de recomendaciones o feeds de redes sociales. Este perfil realiza un 95% de operaciones de lectura y un 5% de inserciones, utilizando una distribución de acceso basada en recency (distribución de último acceso). Al ejecutar el comando run con el workload D, se evalúa cómo la base de datos maneja patrones de acceso no uniformes, midiendo métricas de latencia y rendimiento mientras se priorizan las lecturas de los datos más recientes sobre los más antiguos.

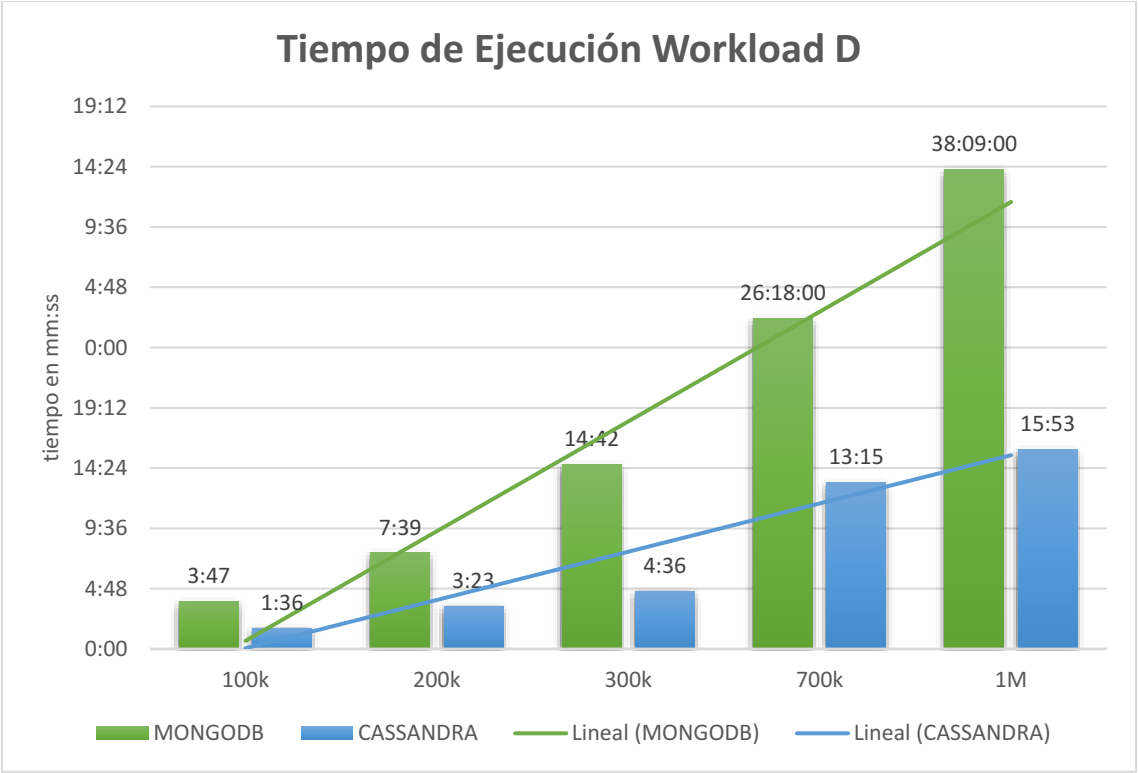

Ilustración 17: Cantidad de registros y tiempo de ejecución en minutos y segundos Workload D.

En volúmenes pequeños, CassandraDB demuestra una clara ventaja en eficiencia. Para 100.000 registros, CassandraDB toma 1:36 minutos, mientras que MongoDB requiere 3:47 minutos, más del doble del tiempo. Esta brecha persiste con 200.000 registros, donde CassandraDB completa en 3:23 minutos, frente a los 7:39 minutos de MongoDB, indicando la capacidad de CassandraDB para manejar accesos rápidos desde el inicio.

Con 300.000 registros, CassandraDB se mantiene significativamente más rápido, con un tiempo de ejecución de 4:36 minutos, mientras que MongoDB necesita 14:42 minutos, casi cuatro veces más. Para 700.000 registros, CassandraDB logra completar en 13:15 minutos, mientras que MongoDB sufre una notable degradación, alcanzando un tiempo de 26:18 minutos, lo que resalta problemas de escalabilidad bajo cargas medianas.

Para 1 millón de registros, CassandraDB conserva su superioridad, con un tiempo de 15:53 minutos, mientras que MongoDB registra 38:09 minutos, más del doble del tiempo necesario. Este resultado subraya la robustez de CassandraDB para manejar grandes volúmenes de datos y operaciones concurrentes de lectura.

Capítulo 5: Conclusión

La comparación entre MongoDB y Cassandra en términos de rendimiento y capacidad de carga ofrece una visión valiosa para los arquitectos y desarrolladores de sistemas de bases de datos. Después de analizar los tiempos de ejecución y la eficiencia en diferentes tamaños de trabajo, podemos llegar a conclusiones significativas que guían en la selección del sistema más adecuado para una aplicación específica.

En primer lugar, MongoDB, con su enfoque orientado a documentos, demuestra un rendimiento destacado en cargas de trabajo de tamaño pequeño a mediano, particularmente en escenarios que combinan lecturas y escrituras aleatorias (Workload A) o aquellas con una alta proporción de lecturas en datos recientes (Workload C). Su capacidad para manejar operaciones mixtas con bajos costos de latencia en entornos más pequeños permite que se desempeñe eficientemente en aplicaciones de escala moderada.

Por otro lado, CassandraDB, diseñado para ofrecer alta disponibilidad y escalabilidad en grandes volúmenes de datos, presenta una ventaja significativa en workloads que involucran grandes volúmenes de registros y lecturas recientes (Workload B y D). Su arquitectura distribuida le permite mantener un rendimiento más constante a medida que la cantidad de datos crece, lo que lo convierte en una opción preferente para aplicaciones de misión crítica que requieren manejo de grandes cantidades de datos distribuidos y consultas con alta demanda de lectura en entornos de alta disponibilidad.

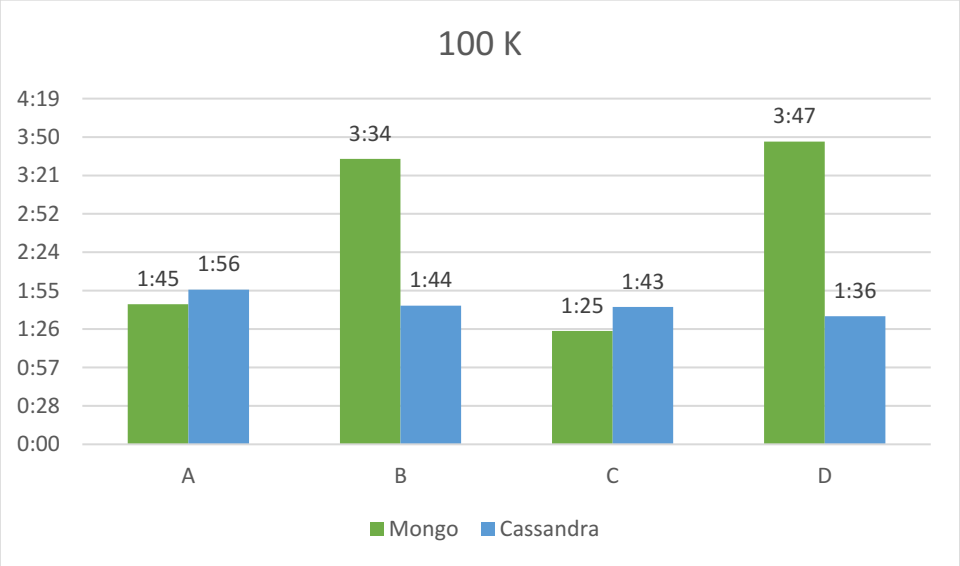

Ilustración 18: Comparativa visual en minutos y segundos con 100 mil registros.

En términos de escalabilidad, CassandraDB sobresale en cargas más grandes y en aquellos escenarios que requieren alta disponibilidad y distribución geográfica, mientras que MongoDB tiende a ser más adecuado para aplicaciones que manejan volúmenes más pequeños de datos y que priorizan la flexibilidad de modelo de datos y facilidad de uso.

Los tiempos de respuesta de Cassandra tienden a mejorar a medida que aumenta el volumen de datos, destacándose en entornos donde la escalabilidad horizontal y la alta disponibilidad

son prioritarias. Esto la convierte en una opción sólida para aplicaciones distribuidas y de alta concurrencia que necesitan una base de datos confiable en operaciones a gran escala.

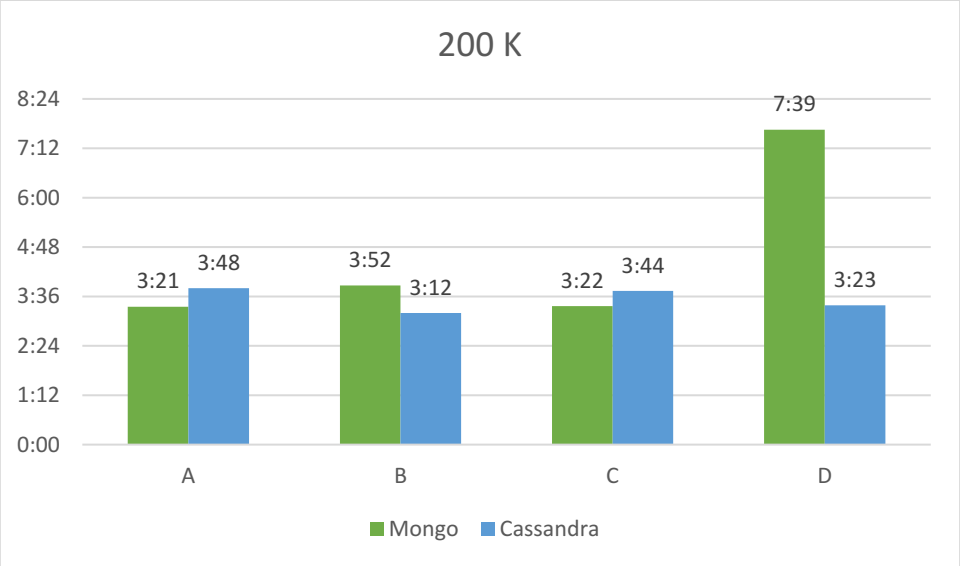

Ilustración 19: Comparativa visual en minutos y segundos con 200 mil registros.

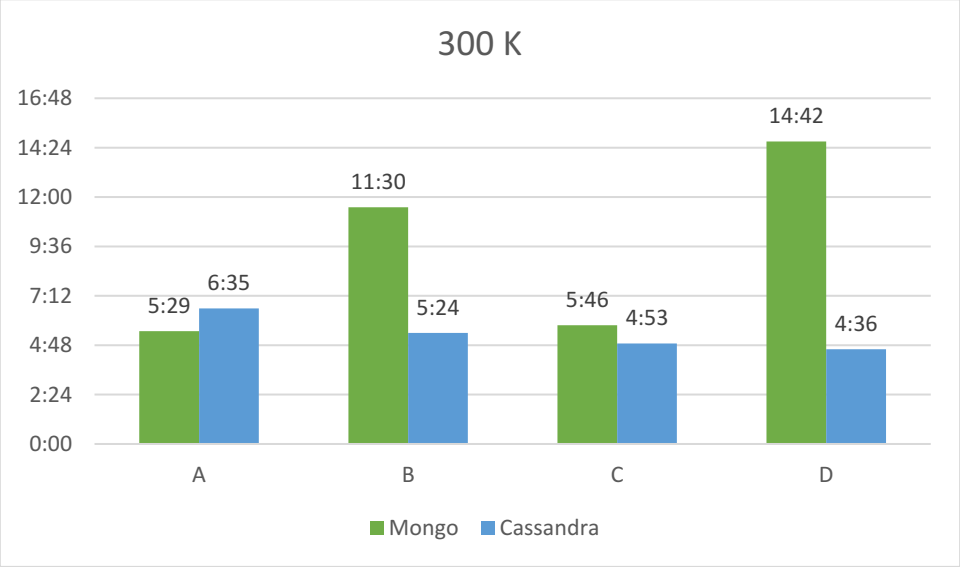

Ilustración 20: Comparativa visual en minutos y segundos con 300 mil registros.

Además del rendimiento, la flexibilidad y la adaptabilidad son aspectos cruciales a considerar. MongoDB destaca por su capacidad para manejar datos no estructurados o semiestructurados, ofreciendo una flexibilidad excepcional en la estructura de los datos. Por otro lado, Cassandra está diseñado para escenarios donde la escalabilidad y la disponibilidad son prioritarias, con una arquitectura altamente optimizada para la replicación y distribución geográfica de los datos.

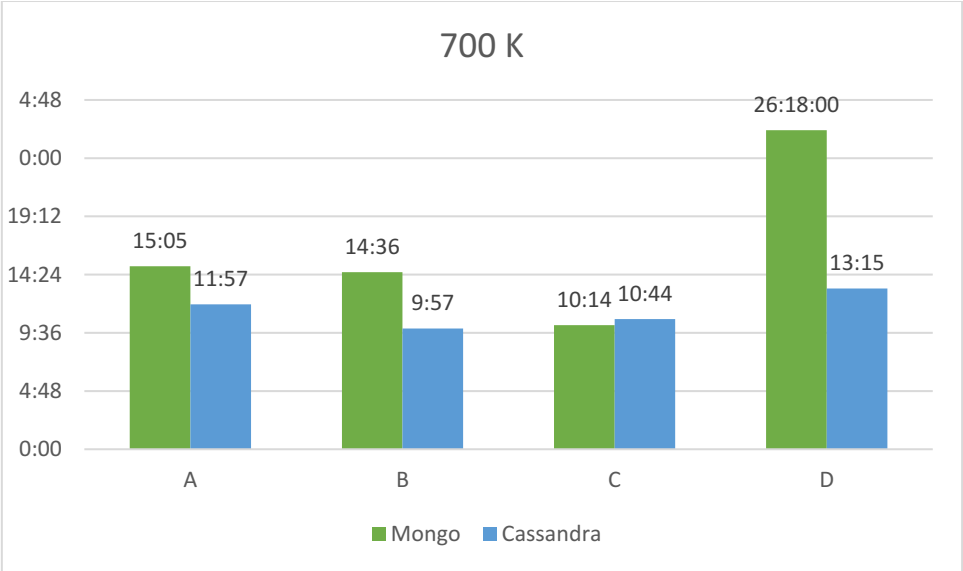

Ilustración 21: Comparativa visual en minutos y segundos con 700 mil registros.

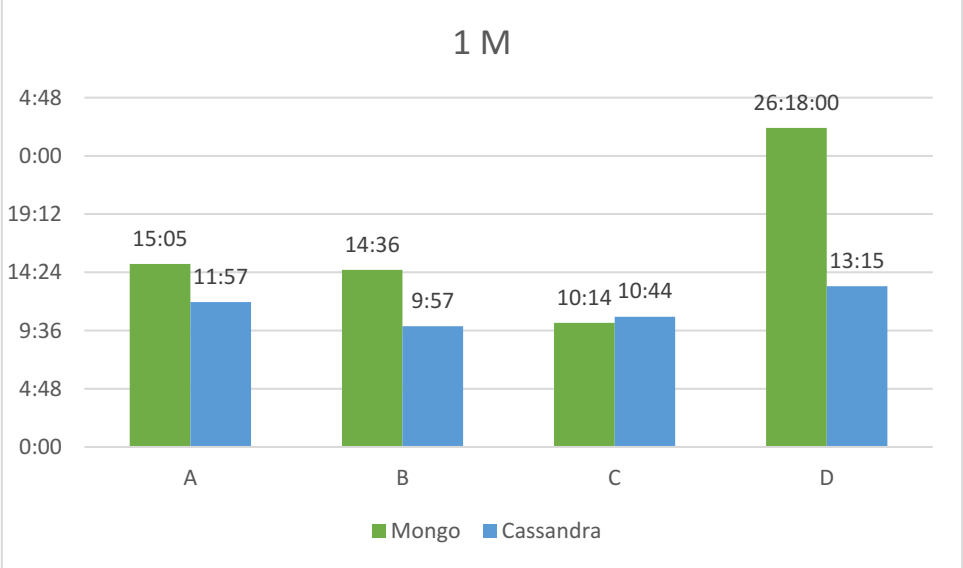

Ilustración 20: Comparativa visual en minutos y segundos con 1 millón de registros.

Otro factor importante a tener en cuenta es la consistencia de los datos. MongoDB ofrece una consistencia fuerte por defecto, garantizando la coherencia de los datos en todo el sistema. En contraste, Cassandra adopta un modelo de consistencia eventual, lo que prioriza la disponibilidad y la tolerancia a fallos sobre la consistencia inmediata.

El objetivo del presente trabajo fue mostrar la comparativa entre ambas bases de datos No Sql, y brindar una inclinación por alguna de ellas dependiendo de la necesidad de cada situación y de una evaluación detallada de los requisitos específicos del problema o sistema a desarrollar, incluyendo el tamaño y la naturaleza de la carga de trabajo, las expectativas de escalabilidad, disponibilidad y las necesidades de consistencia de datos.

La elección entre MongoDB y CassandraDB debe basarse en las necesidades específicas de la aplicación: MongoDB es más eficiente para cargas de trabajo pequeñas o medianas con

requerimientos de flexibilidad, mientras que CassandraDB es ideal para aplicaciones que necesitan manejar grandes volúmenes de datos con alta disponibilidad y consistencia. La naturaleza del workload y el perfil de la carga de trabajo son determinantes clave para seleccionar la base de datos más adecuada, y ambas tecnologías ofrecen ventajas en diferentes contextos de uso.

Para concluir, sería interesante considerar trabajos futuros enfocados en el análisis de desempeño de MongoDB y CassandraDB en aplicaciones específicas, como sistemas de recomendación, análisis de grandes volúmenes de datos en tiempo real. Asimismo, una línea de investigación podría ser la integración de estas bases de datos con tecnologías emergentes como inteligencia artificial o blockchain, evaluando su impacto en términos de escalabilidad, seguridad y eficiencia.

Finalmente, explorar nuevas herramientas y metodologías para optimizar el diseño de esquemas y la gestión de datos en estas tecnologías podría aportar un valor significativo a su implementación en escenarios del mundo real.

Bibliografía

1. What is MongoDB?. [Online] https://www.mongodb.com/docs/manual/.

2. Martin Fowler y Pramod J. Sadalage (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.

3. Install MongoDB Community with Docker. [Online] https://www.mongodb.com/docs/v4.4/tutorial/install-mongodb-community-with-docker/.

4. Welcome to Apache Cassandra's documentation!. [Online] https://cassandra.apache.org/doc/latest/.

5. Installing the docker image [Online] https://cassandra.apache.org/doc/4.1/cassandra/getting_started/installing.html#installing-the-docker-image.

6. Docker Engine overview. [Online] https://docs.docker.com/engine/.

7. About Docker and DataStax. [Online] https://docs.datastax.com/en/docker/docs/overview/docker-about.html

8. 15 comandos de Docker para administrar contenedores [Lista, Detener, Iniciar, Eliminar y más]. [Online] https://geekflare.com/es/docker-manage-containers/.

9. Brianfrankcooper / YCSB. [Online] https://github.com/brianfrankcooper/YCSB.

10. Rohini G Gaikwad (Enero 2021). Performance Evolution of Wide Column Store NoSQL Database in non-distributed environment. [Online]. https://www.researchgate.net/publication/348705890.

11. Adity Gupta, Swati Tyagi, Nupur Panwar, Shelly Sachdeva (Octubre 2017). NoSQL Databases: Critical Analysis and Comparison. [Online] https://www.researchgate.net/publication/323057709.

12. Veronika Abramova, Jorge Bernardino (Julio 2013). NoSQL Databases: MongoDB vs Cassandra. [Online]. https://dl.acm.org/doi/10.1145/2494444.2494447.

13. Cornelia A. Gyorödi, Diana V. Dumse-Burescu, Doina R. Zmaranda and Robert ¸ S. Gyorödi (Mayo 2022). A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management. [Online] https://mdpi-res.com/d_attachment/BDCC/BDCC-06-00049/article_deploy/BDCC-06-00049.pdf?version=1651756.

14. Álvarez-Pallete: "Los datos son el petróleo del siglo XXI" (Julio 2015). https://www.iese.edu/es/noticias/alvarez-pallete-datos-son-petroleo-siglo-xxi.

15. Scores of the DB-Engines Ranking, clasificación, puntuación y ranking de bases de datos (Agosto 2023). https://db-engines.com/en/ranking.

16. Edgar F. Codd (1970). A Relational Model of Data for Large Shared Data Banks. https://dl.acm.org/doi/10.1145/362384.362685..

17. Dan McCreary y Ann Kelly (2013). Making Sense of NoSQL.

18. ¿Qué es NoSQL?. [Online]. https://aws.amazon.com/es/nosql/.

19. Martin Kleppmann (2017). Designing Data-Intensive Applications" (Diseño de Aplicaciones Intensivas en Datos).

20. Kristina Chodorow y Michael Dirolf (2010). "MongoDB: The Definitive Guide" (MongoDB: La Guía Definitiva).

21. Jessica Castillo, Jonathan Garcés, Milton Navas, Diego Jácome Segovia (2017). Base de Datos NoSQL: MongoDB vs. Cassandra en operaciones CRUD (Create, Read, Update, Delete). https://revistapublicando.org/revista/index.php/crv/article/view/398.

22. Tim Wellhausen (2012). ”Highly Scalable, Ultra-Fast and Lots of Choices: A Pattern Approach toNoSQL”. [Online]. https://tim-wellhausen.de/papers/NoSQL-Patterns.pdf.

23. Eric Brewer (2000). Towards robust distributed systems. [Online]. https://www.researchgate.net/publication/221343719_Towards_robust_distributed_systems

24. Sharma, S.; Shandilya, R.; Patnaik, S.; Mahapatra, Ashok. Leading NoSQL models for handling Big Data: A brief review. (2015). https://www.researchgate.net/publication/270273828_Leading_NoSQL_models_for_handling_Big_Data_A_brief_review

25. Docker Compose overview. [Online] https://docs.docker.com/compose/.

Anexos

Anexo I: Instalación y uso de Docker y Docker-Compose

Para obtener una versión estable de Docker debemos de actualizar los repositorios del Sistema Ubuntu. Aunque para obtener la última versión debemos añadir el repositorio oficial de Docker, para ello podemos seguir la guía oficial de Docker.

- sudo apt-get update

Instalación de "docker", instalar dos paquetes, docker y docker.io.

- sudo apt-get install docker docker.io

Continuamos con la instalación de "docker-compose".

- sudo apt-get install docker-compose

Una vez finalizada la instalación de Docker y Docker-Compose, los archivos tipo "***.YAML***", que en los próximos Anexos se describen, van a poder ser levantados con el siguiente comando:

- sudo docker-compose up -d

Ese comando debe ser ejecutado en una terminal que esté abierta desde la carpeta donde se encuentre dicho archivo "***DOCKER-COMPOSE.YAML***".

Anexo II: Instalación de YCSB

Para la instalación del software YCSB, procedemos a ejecutar los siguientes comandos, siempre con los permisos de ROOT.

- curl -O --location https://github.com/brianfrankcooper/YCSB/releases/download/0.17.0/ycsb-0.17.0.tar.gz

- tar xfvz ycsb-0.17.0.tar.gz

Anexo III: Creación de archivo YAML MongoDb

Para la creación de la base de datos MongoDb, partimos de la siguiente estructura YAML. En Ubuntu podemos crear un archivo de texto, copiando la siguiente estructura, y por ultimo cambiando el tipo de archivo, a, "***.YAML***", podemos obtener un archivo tipo YAML.

```
version: '3.8'

networks:
  my_network:
    name: my_network

services:
  db-mongo:
    container_name: mongo4
    hostname: mongo4
    image: mongo:4.4.18
    environment:
      - MONGO_INITDB_ROOT_USERNAME=root
      - MONGO_INITDB_ROOT_PASSWORD=x
    ports:
      - '27017:27017'
    volumes:
      - ./mongo_data:/data/db
    networks:
      - my_network
```

En cuanto al software para visualizar la información y/o poder verificar los datos cargados, se utilizó MongoDB COMPASS, se lo puede descargar e instalar.

Anexo IV: Creación de archivo YAML Cassandra

Para la creación de la base de datos CASSANDRA, partimos de la siguiente estructura YAML. En Ubuntu podemos crear un archivo de texto, copiando la siguiente estructura, y por ultimo cambiando el tipo de archivo, a, "***.YAML***", podemos obtener un archivo tipo YAML.

```
version: '3.8'

services:
  studio:
    image: datastax/dse-studio:6.8.26
    container_name: datastax
    ports:
      - 9091:9091
    depends_on:
      - cassandra
    environment:
      DS_LICENSE: accept

  cassandra:
    image: datastax/dse-server:6.8.35 #5.1.37
    container_name: cassandra
    volumes:
      - ./data:/lib/romano
      - ./cassandra.yml:/etc/dse/cassandra/cassandra.yaml
    ports:
      - 9042:9042
    environment:
      DS_LICENSE: accept
      CASSANDRA_USER : x
      CASSANDRA_PASSWORD : x
```

En cuanto al software para visualizar la información y/o poder verificar los datos cargados, se utilizó DSE-STUDIO v 6.8.26, que también está configurado en el archivo YAML antes mencionado.