UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Un estudio de performance para Bases de Datos NoSQL orientadas a grafos

AUTORES: Mozzon, Federico (14180/6) y Martínez, Manuel (14255/8)

DIRECTOR/A: Lic. Marrero, Luciano – Mg. Thomas Pablo

ASESOR/A PROFESIONAL: Lic. Olsowy, Verena

CARRERA: Licenciatura en Sistemas

Resumen

Se realiza un estudio comparativo sobre motores de bases de datos NoSQL orientadas a grafos. Se plantean un conjunto de pruebas de performance en diferentes casos de estudio y se analizan los resultados obtenidos. Además, se realiza una evaluación de las métricas obtenidas por cada motor de base de datos al escalar horizontalmente. Finalmente, se realiza un análisis y se obtienen conclusiones de todas las pruebas realizadas sobre cada uno de los motores de bases de datos utilizados.

Palabras Clave

Bases de datos NoSQL, Bases de datos Relacionales, Bases de datos orientadas a grafos, almacenamiento no estructurado de datos, escalabilidad horizontal.

Trabajos Realizados

Estudio y análisis de distintos motores de bases de datos relacionales y NoSQL orientados a grafos.

Instalación, configuración y escalabilidad horizontal de base de datos NoSQL orientadas a grafos.

Desarrollo de casos de estudios.

Implementación de algoritmos concurrentes para la carga masiva de datos en una base de datos NoSQL orientada a grafos y de consultas de inserción y lectura de datos.

Estudio y análisis de los resultados obtenidos.

Conclusiones

En este trabajo se han abordado conceptos sobre bases de datos relacionales y no relacionales, más detalladamente las orientadas a grafos. Posteriormente, se plantean un conjunto de casos de estudio en 3 bases de datos, Neo4j, Memgraph y NebulaGraph. El objetivo es realizar una serie de mediciones comparativas de performance entre los 3 motores de bases de datos. Luego de los casos realizados, se demostró que los 3 motores de bases de datos estudiados poseen un comportamiento diferente ante el mismo caso de uso, destacando algunas métricas de performance en entornos concurrentes, así como en entornos de procesamiento secuencial.

Trabajos Futuros

Ampliar este trabajo incorporando nuevas configuraciones para escalar horizontalmente

Realizar pruebas similares para otros tipos de motores de base de datos NoSQL.

Expandir los casos de estudio para incluir nuevos motores de bases de datos orientados a grafos y probar nuevas con consultas específicas para motores de bases de datos NoSQL orientados a grafos.

Fecha de la presentación: 27 de Mayo 2024

Capítulo 1 - Introducción y Objetivos

1.1 Introducción

Actualmente, el modelo de bases de datos relacional aún predomina en el mercado. No obstante, en las últimas décadas, el gran avance tecnológico ha cambiado la naturaleza de los problemas a resolver junto con la necesidad de almacenar y manipular un volumen enorme de datos. En este contexto, los motores de bases de datos relacionales han expuesto algunos problemas de performance. En respuesta a esta problemática, se plantean nuevas alternativas para el almacenamiento de datos conocidas como bases de datos no relacionales o NoSQL [1]. NoSQL representa un conjunto de tipos de bases de datos, en donde cada una posee su propia implementación y características.

Un aspecto común entre los motores de bases de datos NoSQL, es la escalabilidad horizontal. Es decir, distribuir la base de datos en más de un servidor para mejorar aspectos como la disponibilidad, la tolerancia a fallos y el tiempo de respuesta en ambientes donde existe una gran demanda de acceso a los datos.

1.2 Objetivo

Este trabajo se centra en el planteo e implementación de un conjunto de pruebas de rendimiento para motores de bases de datos NoSQL orientados a grafos con el objetivo de obtener métricas de performance para distintos casos de uso con una gran cantidad de datos.

Además, se estudia y analiza la escalabilidad horizontal y el comportamiento de cada uno de los motores de bases de datos seleccionados en cada caso de uso planteado.

También, se presenta un estudio y análisis sobre diferentes motores de bases de datos NoSQL, en donde se exponen sus características principales.

Capítulo 2 - Conceptos generales de Bases de Datos

2.1 Introducción

Según [2] la idea de resguardar información se remonta a la antigüedad, en donde existían librerías en las cuales se resguardaban registros físicos de diversas temáticas, medicina, gubernamentales, negocios, entre otros.

La necesidad de mantener la información resguardada y el avance tecnológico, dio lugar a lo que actualmente se conoce como base de datos. Para [3] se puede definir a una base de datos como una colección organizada de datos, y que se diferencian en la forma que organizan y almacenan sus datos, así como también, en la forma en que se accede a ellos.

En una base de datos, existen aspectos importantes a tener en cuenta, entre ellos, el modelo de datos y el lenguaje de consulta.

Los modelos de datos pueden ser divididos en dos grandes categorías, conceptuales o físicos. Los conceptuales se centran en una abstracción de alto nivel de los datos, usando conceptos como entidades, atributos y relaciones. Los físicos son los que representan el formato concreto de almacenamiento de un sistema (archivos).

El lenguaje de consulta permite especificar la estructura de los datos y además realizar consultas sobre los mismos. Por ejemplo, las bases de datos relacionales poseen el Lenguaje de Consulta Estructurado (SQL, por sus siglas en inglés, Structured Query Language) [4, 5].

Existen distintos tipos de bases de datos [4], entre ellos están:

- **Base de datos jerárquica**: las bases de datos jerárquicas surgen en los años 60 en IBM, con la IBM Information Management System (IMS) [6]. El modelo de datos jerárquico es similar en muchos aspectos al modelo de datos de red, que surgiría unos años después, en donde los datos también se organizan en registros, aunque difiere de este último en la forma en que vincula los tipos de registros. Los DBMS jerárquicos organizan los datos en una estructura de árbol, comenzando con un solo tipo de registro en la parte superior de la estructura de árbol, mientras que los de red, no cuentan con este primer registro. Las bases de datos jerárquicas acceden a los datos utilizando un HDML (lenguaje de manipulación de datos jerárquicos) un solo registro a la vez [7, 8].

  A continuación, en la figura 1 se presenta el fragmento de una base de datos jerárquica.

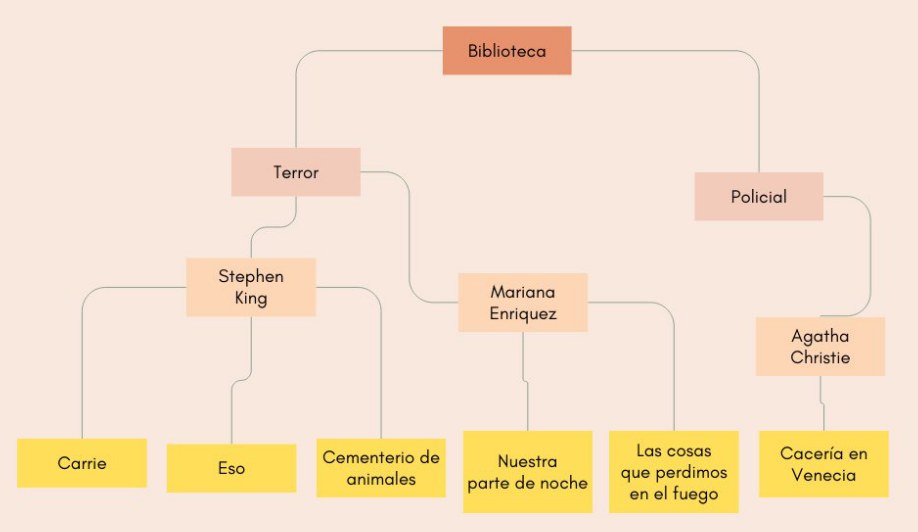

Figura 1. Ejemplo de una base de datos jerárquica.

En este ejemplo, una biblioteca cuenta con géneros literarios, estos géneros literarios están asociados con escritores y estos escritores han escrito diferentes libros.

Algunos ejemplos de bases de datos jerárquicas son: IBM Information Management System [9] y RDM mobile [10].

- **Base de datos en red**: este tipo de bases de datos fue propuesto por CODASYL (Conference on Data Systems Languages) en el año 1971 y organiza los datos en registros, los cuales contienen un grupo de elementos o de atributos relacionados. Se puede pensar este tipo de base de datos como un árbol invertido el cual cuenta con varios registros padres los cuales se relacionan con otro registro, generalmente llamado hijo, en un tipo de relación 1 a N. En una base de datos de red, se accede a los registros de uno en uno utilizando un lenguaje de manipulación de datos integrado en un lenguaje de programación [7].
  A continuación, en la figura 2 se presenta un ejemplo de base de datos en red.

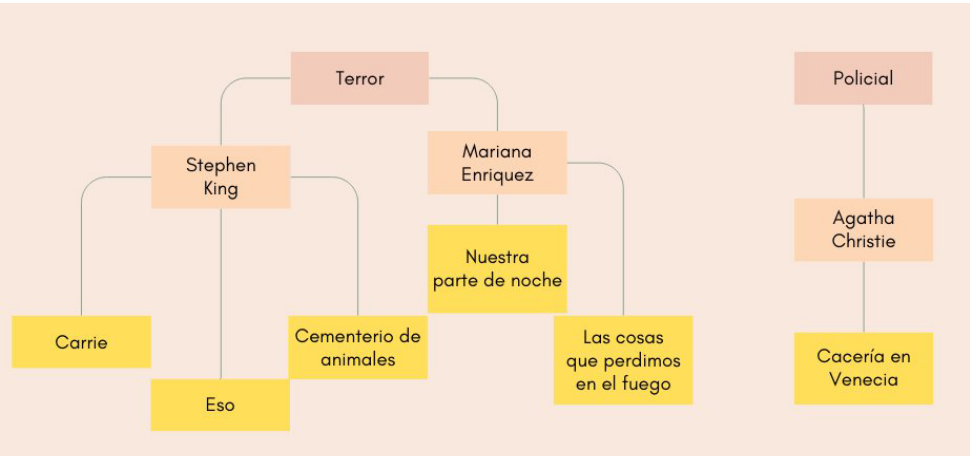

Figura 2. Ejemplo de una base de datos en red.

En este ejemplo, se presentan dos géneros literarios, ambos géneros están asociados con escritores y los escritores han escrito diferentes libros.

Algunos ejemplos de bases de datos en red son: HP IMAGE [11], Integrated Data Store de General Electrics [12] y Oracle CODASYL DBMS [13].

- **Base de datos relacional**: el modelo de bases de datos relacionales fue propuesto por E.F. Codd en su artículo “A Relational Model of Data” en el año 1970 [14]. Este modelo se basa en la teoría de conjuntos y la lógica de predicados, donde el esquema de la base de datos está desconectado del almacenamiento físico, esto es una de las características principales de las bases de datos relacionales [15].
  Según [2] las bases de datos relacionales pueden describirse utilizando dos terminologías:
  - Instancia: una tabla con filas y columnas.
  - Esquema: especifica la estructura, incluido el nombre de la relación, el nombre y el tipo de cada columna.

  A continuación, en la figura 3, se presenta un ejemplo de base de datos relacional.

Libro

ID	Nombre	ID Escritor
1	Carrie	1
2	Eso	1
3	Cementerio de animales	1
4	Nuestra parte de noche	2
5	Las cosas que perdimos en el fuego	2
6	Cacería en Venecia	3

Escritor

ID	Nombre	ID Genero
1	Stephen King	2
2	Mariana Enriquez	2
3	Agatha Christie	1

Genero

ID	Nombre
1	Policial
2	Terror

Figura 3. Ejemplo de un fragmento de una base de datos relacional

En este ejemplo existen dos géneros literarios, los cuales están asociados con escritores, y estos escritores han escrito diferentes libros.

Algunos ejemplos de este tipo de bases de datos son PostgreSQL [16], MySQL [17], MariaDB [18] y Oracle Database [19].

- **Base de datos de objetos**: las bases de datos orientadas a objetos (ODBMS, por su sigla en inglés, Object Database Management System) permiten almacenar objetos en la base de datos y recuperarlos como tales, logrando mayor integración con los lenguajes de programación orientados a objetos

  Este tipo de bases de datos posee un lenguaje de consulta denominado Object Query Language (OQL) que fue definido por la Object Data Management Group (ODMG) el cual permite recuperar los datos, filtrar por diferentes criterios, entre

otras cosas. La forma de acceder a la información es a través de punteros en memoria [20, 21]. También permite integrarse con lenguajes de este paradigma como Delphi [22], Ruby [23] o Java [20, 24].

A continuación, en la figura 4, se presenta un fragmento perteneciente a un diagrama de clases para una base de datos orientada a objetos.

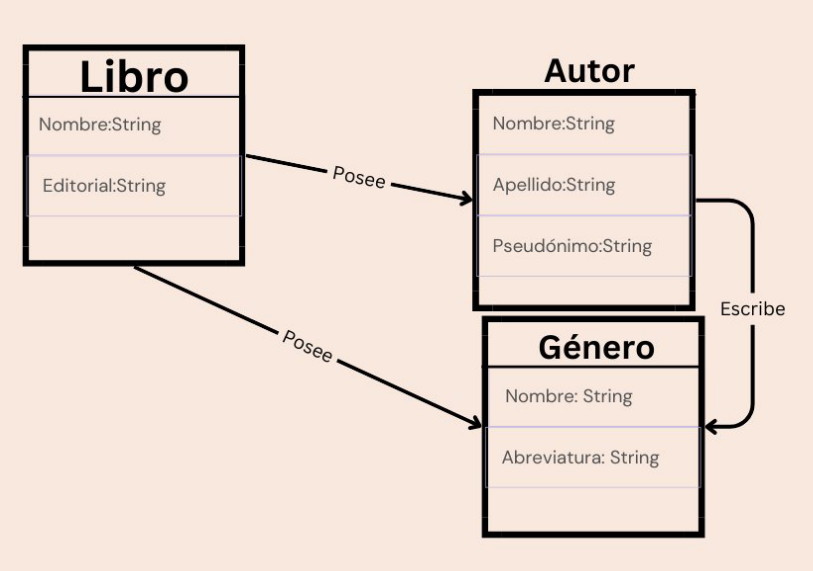

Figura 4. Ejemplo de un diagrama de clases para una base de datos orientada a objetos.

En este ejemplo un libro posee un autor y un autor escribe sobre un género literario.

Algunos ejemplos de bases de datos orientadas a objetos son: Perst [25], ObjectStore [26] y ObjectDB [27].

- **Bases de datos No Relacionales (NoSQL)**: el término NoSQL se utiliza para describir a un conjunto de tipo de bases de datos que utilizan diversas maneras de estructurar los datos que almacenan.
  A continuación, en la figura 5a y 5b, se presentan las 4 categorías de almacenamiento más utilizadas por bases de datos NoSQL [28] (documental, clave - valor, orientado a grafos y familia de columnas).

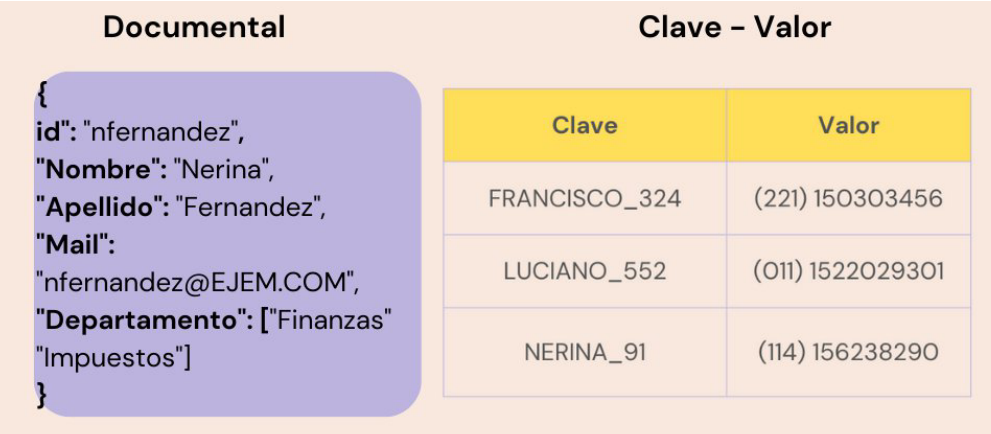

Figura 5a. Ejemplos de bases de datos NoSQL documental y clave-valor.

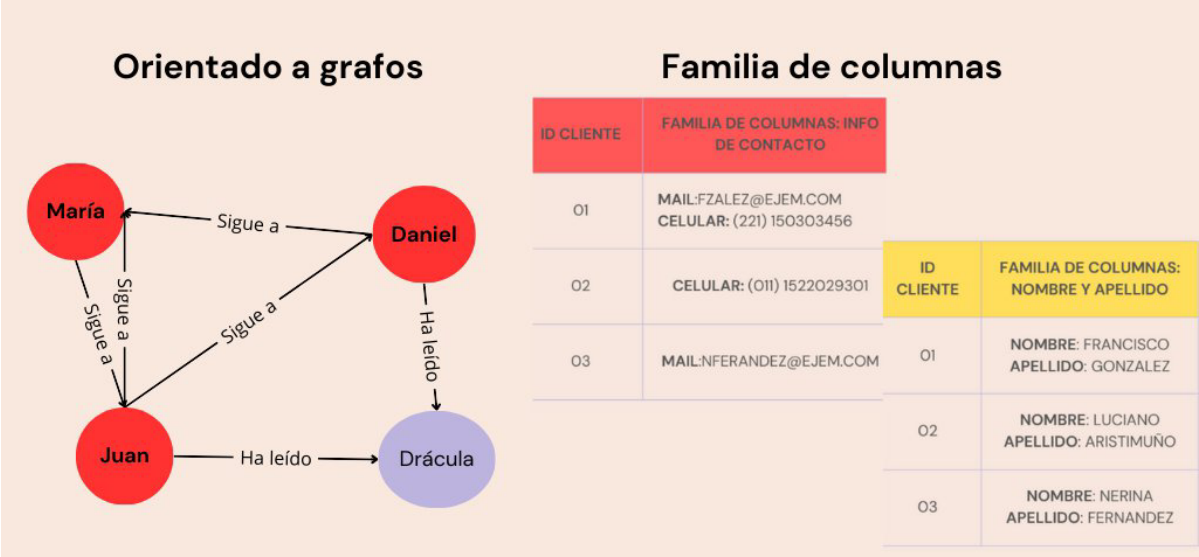

Figura 5b. Ejemplos de bases de datos NoSQL orientada a grafos y familia de columnas.

Algunos ejemplos de bases de datos NoSQL: MongoDB [29], Redis [30], Neo4j [31] y Cassandra [32]. Estas bases de datos son ampliamente utilizadas en la actualidad, por empresas importantes entre las cuales se encuentran Amazon, Microsoft y Adobe [33, 34 y 35], entre otras.

2.2 Bases de Datos Relacionales (BDR)

El modelo de base de datos relacional describe la información por medio de tablas y relaciones entre ellas. Las tablas representan típicamente a un objeto del mundo real, y cada tabla posee propiedades, representadas en columnas, las cuales describen las características del objeto. Dentro de una tabla, una fila representa una instancia individual de ese objeto, y puede estar identificada unívocamente por una clave primaria.

Las tablas pueden estar conectadas entre sí mediante referencias a las claves primarias de otras tablas. Este concepto lleva el nombre de clave foránea, esto es, una columna en una tabla que almacena valores de clave primaria de otra tabla. De esta manera se minimiza la redundancia de datos.

Las bases de datos relacionales se caracterizan por tener un esquema fijo, es decir, cada tabla debe respetar las reglas definidas para dicho esquema. El esquema define la cantidad de atributos para cada tabla, junto con sus tipos y dominios correspondientes, restricciones de relaciones, funciones, reglas de indización, entre otras cosas [15].

Las bases de datos relacionales utilizan un lenguaje de consulta estructurado (SQL [36]) que permite definir su estructura y manipular la información contenida. SQL es un estándar definido por la ANSI (American National Standards Institute) [37] y la ISO (International Organization for Standarization) [38] y su implementación más moderna es SQL2023 [39].

Cada motor de base de datos relacional puede definir variantes sobre su implementación de SQL. No obstante, en la mayoría se respetan ciertas características generales del estándar.

Otras ventajas que presenta SQL es la posibilidad de crear funciones, restricciones de datos y transacciones. Una transacción, es la ejecución de una o más operaciones en la base de datos vistas como una única unidad atómica de trabajo. Las transacciones en una base de datos relacional garantizan la integridad y consistencia de los datos almacenados.

En una base de datos relacional, las transacciones deben cumplir con las propiedades ACID (Atomicity, Consistency, Isolation, Durability [1]):

- **Atomicidad (Atomicity)**: Cada transacción en la base de datos se trata como una unidad atómica, es decir, o se realiza en su totalidad o no se realiza en absoluto. No puede haber un estado intermedio en el que parte de la transacción se haya realizado y otra parte no.
- **Consistencia (Consistency)**: La base de datos debe pasar de un estado válido a otro estado válido después de completar una transacción. Esto significa que todas las restricciones de integridad deben mantenerse antes y después de que se realice una transacción.
- **Aislamiento (Isolation)**: Múltiples transacciones pueden ocurrir simultáneamente en la base de datos, pero deben ser aisladas unas de otras para evitar que interfieran entre sí. Cada transacción debe ser independiente de las otras.
- **Durabilidad (Durability)**: Una vez que una transacción se ha completado con éxito (es decir, ha sido confirmada), los cambios realizados por esa transacción deben ser permanentes y resistir a fallos del sistema, como cortes de energía o errores del sistema.

Por ejemplo, en una transferencia de dinero de una cuenta a otra, se tiene que restar dinero a una de las cuentas, y sumarlo en la otra cuenta. Esto se debe ejecutar como una única unidad lógica de trabajo, ya que si se resta dinero a la primera cuenta y ocurre algún problema al agregarlo a la otra cuenta, los datos quedarán inconsistentes [3]. Acorde a lo presentado en [39 y 40] existen cuatro categorías de lenguajes de operaciones con los datos, DDL, DML, DQL y DCL, las cuales serán abordadas a continuación.

2.2.1 Lenguaje de Definición de Datos (DDL)

El Lenguaje de Definición de Datos (Data Definition Language o DDL por sus siglas en inglés) permite generar la estructura de una base datos relacional. Son cláusulas que permiten crear, modificar o eliminar aspectos de la estructura de la base de datos. A continuación, en la figura 6 se presenta un ejemplo de una consulta del lenguaje de definición de datos [41].

```
CREATE TABLE Empleados (
 id      INTEGER   PRIMARY KEY,
nombre   VARCHAR(50) not null,
apellido  VARCHAR(75) not null,
nacimiento  DATE     not null);
```

Figura 6. Consulta para crear una tabla en SQL**.**

Esta consulta crea una tabla llamada Empleados con las columnas id (clave primaria), nombre, apellido y fecha de nacimiento. Además, todos las columnas están definidas como no nulas, es decir, que ante una operación de alta de datos, todas tendrán que contener un valor obligatoriamente.

2.2.2 Lenguaje de Manipulación de Datos (DML)

El Lenguaje de Manipulación de Datos (Data Manipulation Language o DML por sus siglas en inglés) se utiliza para recuperar, agregar, modificar o eliminar datos. A continuación, en la figura 7 se presenta un ejemplo de una consulta DML [41].

```
INSERT INTO Empleados(nombre,apellido, nacimiento)
VALUES ('Manuel', 'Lopez', '15-05-1998');
```

Figura 7. Consulta de inserción de datos en SQL.

En esta consulta, se inserta una fila en la tabla Empleados con los datos especificados en la cláusula values.

2.2.3 Lenguaje de Consulta de Datos (DQL)

El Lenguaje de Consulta de Datos (Data Query Language o DQL por sus siglas en inglés) agrupa las instrucciones de consulta de los datos. La instrucción principal es SELECT, la cual se usa para indicar los datos específicos que se quieren consultar. A esta instrucción se la utiliza en conjunto con la cláusula FROM, que indica la tabla a consultar, y WHERE que indica condiciones de consulta, esta última resulta opcional. En la figura 8 se presenta un ejemplo de consulta DQL [39].

```
SELECT nombre
FROM Empleados
WHERE nacimiento > 15-05-1996;
```

Figura 8. Ejemplo de consulta de selección en DQL.

Esta consulta proyecta, de la tabla empleados, el nombre de aquellos empleados cuya fecha de nacimiento fue posterior al 15 de mayo de 1996.

2.2.3 Lenguaje de Control de Datos (DCL)

El Lenguaje de Control de Datos (Data Control Language o DCL por sus siglas en inglés) no se utiliza para interactuar con los datos, sino para definir el control de acceso a los mismos. Dos de las instrucciones que posee este lenguaje son GRANT, que permite habilitar acciones específicas a un usuario del sistema, y REVOKE, que permite remover acciones para un usuario. A continuación, en la figura 9, se presentan 2 ejemplos de estas instrucciones [39].

```
GRANT SELECT ON Empleados TO Manuel;

REVOKE INSERT ON Empleados FROM Manuel;
```

Figura 9. Ejemplo de consultas de control en DCL.

En este ejemplo, se otorga permiso de selección y se remueve el permiso de inserción al usuario Manuel sobre la tabla empleados.

Según [28], 7 de los 10 motores de bases de datos más populares en la actualidad son relacionales. Su popularidad se debe a diversas razones, entre las más importantes se destacan la facilidad de uso y sencillez de la estructura, permitiendo describir datos del mundo real en tablas de una forma predecible y conectadas entre sí [1 ,3, 40].

A pesar de la popularidad que poseen las bases relacionales, el crecimiento exponencial de la cantidad de datos generados y administrados por nuevas generaciones de aplicaciones informáticas y la necesidad de representar datos que no posean una estructura fija, evidenciaron ciertas limitaciones que poseen estos tipos de bases de datos.

En particular, la escalabilidad horizontal en una base de datos relacional no es una tarea sencilla. En general, las bases de datos relacionales están diseñadas para una arquitectura centralizada apropiada para escalar principalmente de forma vertical. Además, en algunos casos, requieren mucho espacio en memoria secundaria (disco), por lo que, al aumentar de forma exponencial la cantidad de datos, dificulta algunas de las operaciones, por ejemplo, realizar copias de seguridad o cruces entre tablas (JOINs) [41].

En este contexto, surgen nuevas formas de almacenar y estructurar los datos. Algunos ejemplos son el almacenamiento documental, el almacenamiento de familias de columnas, el almacenamiento clave-valor y el almacenamiento orientado a grafos, entre otros.

Capítulo 3 (Bases de Datos NoSQL)

3.1 Introducción

A finales de la década de 1990 se comienza a usar el término NoSQL que describe a las bases de datos que implementan novedosas alternativas para el almacenamiento de datos. NoSQL hace referencia a “not only SQL”, debido a que algunos motores de bases NoSQL soportan alguna variante de implementación SQL [41].

En el año 2006, NoSQL se hace más popular con el lanzamiento de Google BigTable [42] que introduce la noción de una base de datos con almacenamiento clave-valor. Posteriormente, aparecen otros motores de bases de datos NoSQL como Redis [30] (almacenamiento clave-valor), MongoDB [29] (almacenamiento documental), Cassandra [32] (almacenamiento de familia de columnas), Neo4j [31] (almacenamiento orientado a grafos), entre otros [43].

Este tipo de bases de datos se caracterizan por almacenar datos no estructurados y heterogéneos en una misma base de datos y considerar aspectos para optimizar la escalabilidad horizontal. Estas características permiten diseñar soluciones para los problemas de almacenamiento de datos en las aplicaciones modernas, como por ejemplo, redes sociales, plataformas de contenido, entidades financieras, entre otras [43]. Generalmente, las bases de datos NoSQL son incorporadas por empresas que necesitan manejar grandes volúmenes de datos como, por ejemplo, GitHub, StackOverflow, Microsoft e IBM [33, 34 y 35], entre otras.

El teorema CAP define tres propiedades: Consistencia (Consistency), Disponibilidad (Availability) y Tolerancia a Particiones (Partition Tolerance), y establece que un sistema distribuido sólo puede cumplir con dos de estas tres propiedades de forma simultánea. Como las bases de datos NoSQL están pensadas para ser distribuidas y escalables, pueden ser clasificadas en tres grupos:

- **AP(Disponibilidad y tolerancia a partición)**: Al garantizar la disponibilidad y tolerancia de particiones, se generan escenarios donde hay inconsistencias temporales en las particiones.
- **CP(Consistencia y tolerancia a partición)**: Al garantizar la consistencia y tolerancia de particiones al ocurrir una partición, habrá nodos los cuales no se encuentren disponibles.
- **CA(Consistencia y Disponibilidad)**: Al garantizar la consistencia y disponibilidad en caso de ocurrir una falla, se debe optar por no estar completamente disponible o estarlo y correr riesgo de inconsistencias.

A continuación, en la figura 10, se presenta gráficamente el teorema de CAP.

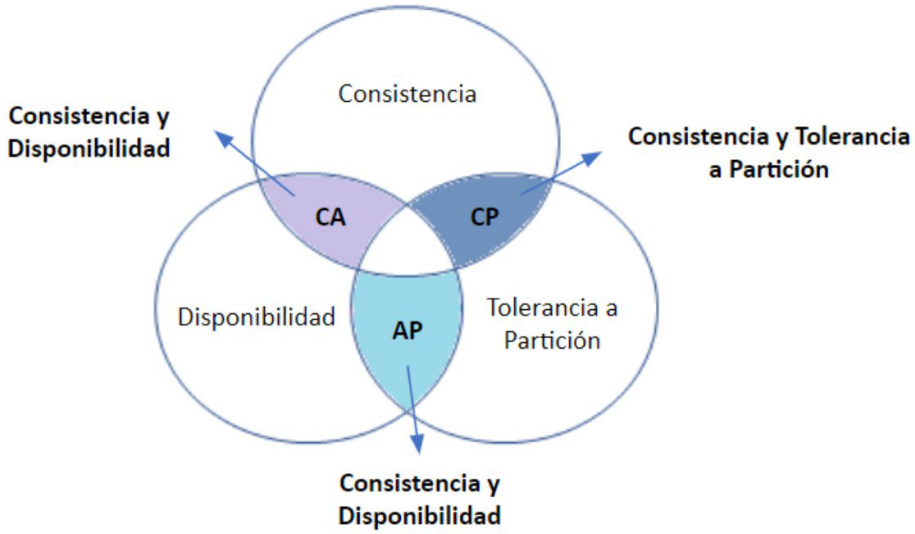

Figura 10. Teorema de CAP.

Actualmente, motores de bases de datos NoSQL como Redis [30], MongoDB [29], Cassandra [32], Neo4j [31], entre otras, ofrecen la posibilidad de brindar una infraestructura de datos como servicios en la nube, es decir, proveen la posibilidad de contratar hardware externo para almacenar y administrar las bases de datos. En la mayoría de los casos, estos servicios ofrecen planes gratuitos de capacidades técnicas limitadas, con la posibilidad de optar por aumentarlas a cambio de un costo periódico de mantenimiento [45]. Mediante estos servicios, el usuario se abstrae del mantenimiento y administración de la infraestructura del hardware. Además, la mayoría de estos servicios proveen escalabilidad automática según la demanda.

Las bases de datos NoSQL poseen diversas formas para la organización de sus datos. Además, establecen distintos criterios para la consulta y/o modificación de los mismos. Cada forma de organización de datos se adapta de forma más eficiente a distintos problemas a resolver.

A continuación, se presentan las cuatro categorías principales de almacenamiento para bases de datos NoSQL.

3.2 Categorías de almacenamiento

3.2.2 Almacenamiento Clave – Valor

Una base de datos NoSQL que implementa almacenamiento clave-valor utiliza un método simple de pares clave-valor, en los que una clave sirve como un identificador único y determina un valor en particular. Tanto las claves como los valores pueden ser cualquier tipo de objetos (simples o complejos). En este tipo de almacenamiento, puede darse que una clave determine un valor simple, por ejemplo un número, mientras que otra clave puede determinar un objeto compuesto por ejemplo una persona con nombre y apellido [46].

Generalmente, las bases de datos que utilizan este tipo de almacenamiento, operan sus estructuras en memoria principal, logrando buen desempeño en cuanto a la velocidad de respuesta, razón por la cual, aplicaciones que necesitan administrar sesiones o almacenamiento caché, hacen uso de este tipo de bases de datos NoSQL clave-valor.

A continuación, en la figura 11 se muestra una representación de almacenamiento clave-valor, en donde las claves son nombres junto con un número aleatorio, y los valores son números de teléfono.

Clave	Valor
FRANCISCO_324	(221) 150303456
LUCIANO_552	(011) 1522029301
NERINA_91	(114) 156238290

Figura 11. Ejemplo de almacenamiento clave - valor.

Algunos de los motores de bases de datos NoSQL más populares que utilizan este tipo de almacenamiento son: Redis [30], Memcached [47], ETCD [48], Ehcache [49] y Amazon DynamoDB [50], entre otros.

3.2.3 Almacenamiento Documental

En este tipo de almacenamiento los datos se estructuran en forma de documentos, los documentos son un conjunto de pares clave-valor donde el valor que pueden contener puede ser un número, un string, un arreglo u otro documento en sí mismo. Este tipo de almacenamiento soporta diferentes formatos para representar los documentos, estos pueden ser JSON [51] o XML [52].

Generalmente, los motores de bases de datos que utilizan almacenamiento documental, agrupan documentos dentro de colecciones. Una colección es análoga a lo que representa una tabla en el modelo relacional, la diferencia es que los documentos pueden tener diferentes estructuras, es decir, documentos que se encuentran en la misma colección y que poseen atributos o campos diferentes. Si bien esta característica brinda flexibilidad, requiere precaución al momento de definir una consulta, ya que se puede referenciar alguna propiedad que no todos los documentos posean.

A continuación, en la figura 12 se presenta un ejemplo de formato para diferentes documentos que pertenecen a la misma colección.

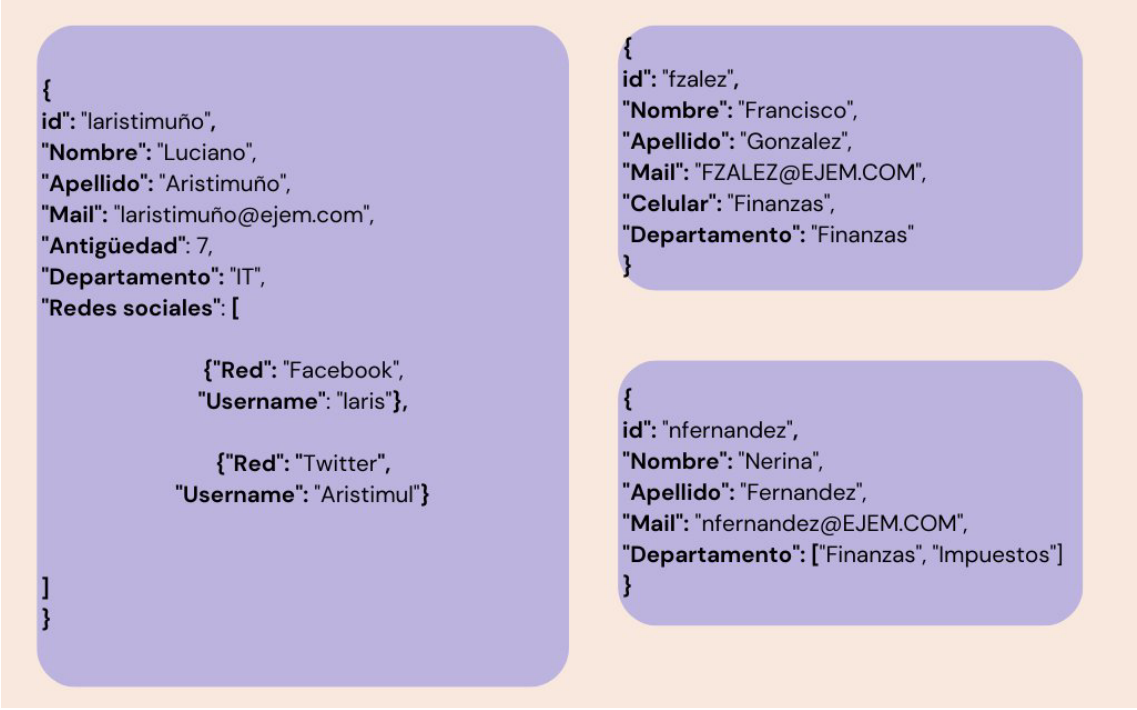

Figura 12. Ejemplo de documentos en formato JSON para almacenamiento documental.

Algunos de los motores de bases de datos NoSQL más populares que utilizan este tipo de almacenamiento son: MongoDB [29] y Couchbase [53] entre otros.

3.2.4 Almacenamiento en Familia de Columnas

Las bases de datos NoSQL de familia de columnas u orientadas a columnas son un tipo de base de datos que se componen de filas y columnas, similar al modelo relacional, pero en donde las columnas de una tabla se agrupan bajo algún criterio, de forma tal que estén relacionadas entre ellas y que sean manipuladas de forma conjunta.

Una de las ventajas de esta implementación es que las columnas suelen contener datos del mismo tipo, es más probable que haya patrones repetitivos y redundancias que puedan comprimirse mejor que en un modelo relacional, donde los datos se almacenan por filas y en general, las consultas requieren un subconjunto de columnas de una tabla [4].

Este tipo de almacenamiento permite un acceso rápido a los datos, ya que solo se accede a las columnas necesarias, reduciendo así el tiempo de acceso y mejorando el rendimiento en comparación con el modelo relacional [54].

Dentro de una familia de columnas, nuevas columnas pueden ser agregadas de forma dinámica y no existe la necesidad de que las filas posean valores para cada una de ellas.

A continuación, en la figura 13a y 13b se presentan un ejemplo de dos esquemas de familia de columnas.

ID CLIENTE	FAMILIA DE COLUMNAS: NOMBRE Y APELLIDO
01	**NOMBRE**: FRANCISCO **APELLIDO**: GONZALEZ
02	**NOMBRE**: LUCIANO **APELLIDO**: ARISTIMUÑO
03	**NOMBRE**: NERINA **APELLIDO**: FERNANDEZ

Figura 13a. Ejemplo de almacenamiento en familia de columnas.

ID CLIENTE	FAMILIA DE COLUMNAS: INFO DE CONTACTO
01	**MAIL**:FZALEZ@EJEM.COM **CELULAR:** (221) 150303456
02	**CELULAR:** (011) 1522029301
03	**MAIL**:NFERANDEZ@EJEM.COM

Figura 13b. Ejemplo de almacenamiento en familia de columnas.

Generalmente, las bases de datos que implementan este tipo de almacenamiento, utilizan un concepto denominado keyspace. El keyspace es la abstracción de un conjunto de familias de columnas que están relacionadas entre ellas.

A diferencia del modelo relacional, el almacenamiento de familia de columnas permite que una familia de columnas pueda poseer atributos que otra no. En el modelo relacional, esto no es posible ya que si no se desea completar algún campo, se debe completar con un valor nulo.

Algunos de los motores de bases de datos NoSQL más populares que utilizan este tipo de almacenamiento son: Cassandra [32], Bigtable [42] y HBase [55], entre otros.

3.2.5 Almacenamiento orientado a Grafos

Este tipo de almacenamiento utiliza una estructura de grafos para representar los datos. Los nodos representan las unidades individuales de información que pueden estar conectados (o no) con uno o más nodos, estas conexiones reciben el nombre de aristas o relaciones. Ambas estructuras se persisten e indexan en la base de datos, lo que permite respuestas rápidas a consultas complejas de relaciones entre nodos[56].

En este tipo de almacenamiento, es importante la información que posee cada uno de los nodos y sus relaciones correspondientes. Por ejemplo, una red social se puede representar mediante un grafo, en donde los nodos representan usuarios, y se implementa la red de conexiones entre usuarios y la forma en que se conectan mediante relaciones. Representar la información físicamente de la misma manera en la que se piensa un problema, evita tener que complejizar la solución con el objetivo de adaptarla a otro tipo de almacenamiento. Algunos ejemplos en los que se puede aplicar este almacenamiento son las redes sociales, sistemas de recomendaciones, búsqueda de viajes y combinaciones de transporte, entre otras [56].

En este tipo de almacenamiento se pueden aplicar algoritmos de búsqueda de la teoría de grafos, como son el BFS (Búsqueda en Anchura) [57], DFS (Búsqueda en Profundidad) [58], Single-source shortest path (camino más corto) [59], All-pairs shortest path (caminos más cortos entre todos los pares), Degree centrality (centralidad) [61] entre otros.

Algunos ejemplos de motores de bases que utilizan este tipo de almacenamiento son: Neo4j [31], OrientDB [62], Memgraph [63] y NebulaGraph [64].

A continuación, en la figura 14, se presenta un ejemplo de almacenamiento de datos orientado a grafos.

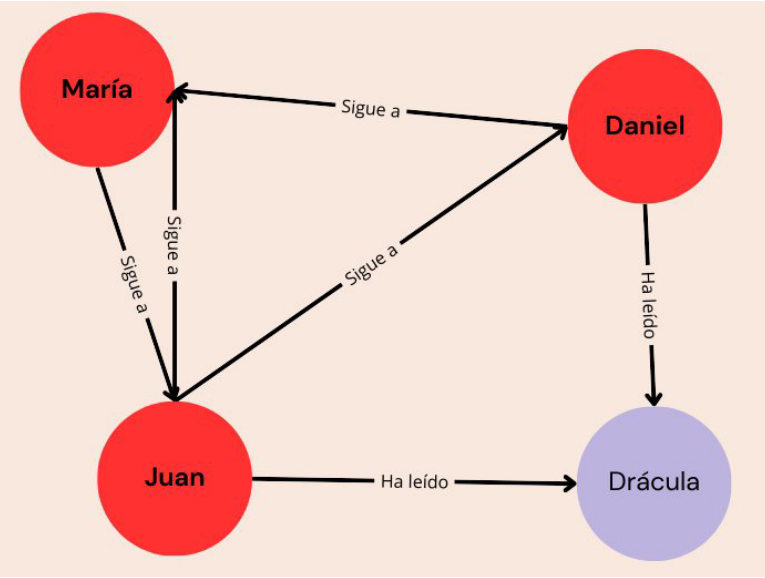

Figura 14. Ejemplo de almacenamiento orientado a grafos.

En este caso se tiene una red social con usuarios (nodos rojos) y libros (nodos violetas). Las aristas representan dos tipos de relaciones, una es de seguimiento entre usuarios y las otras entre usuarios y los libros que han leído.

3.3 Otras categorías de almacenamiento

Existen otros tipos de almacenamiento con menor popularidad. Estos incluyen motores de búsqueda, base de datos vectoriales, bases de datos de series temporales, entre otras [40].

Además, algunos motores de bases de datos NoSQL cuentan con la posibilidad de implementar más de un tipo de almacenamiento, esto permite que con un solo producto puedan abarcar diversas necesidades.

A continuación, se describen brevemente algunos de estos tipos de bases de datos NoSQL.

3.3.1 Motores de búsqueda

Este tipo de bases de datos NoSQL se caracteriza por el indexado de datos, particularmente datos en formato de texto. Se utilizan para agilizar consultas que en los motores de bases de datos relacionales poseen menor performance. Esto se debe a que en general, las bases relacionales no indexan los contenidos de cualquiera de las columnas, sino que solo generan un índice de la clave primaria y/o secundaria [40].

Los motores de búsqueda permiten implementar índices optimizados de contenido en formato de texto, por lo que permiten búsquedas muy rápidas por coincidencias, como por ejemplo, en artículos o blogs. Además, proveen funciones para generar rankings de relevancias y brindar resultados ordenados de forma dinámica.

Algunos de los principales proveedores de este tipo de almacenamiento son Elasticsearch [65], Microsoft Azure Search [66], Amazon Cloud Search [67], y Splunk [68]. Este tipo de almacenamiento, especialmente Elasticsearch, ha sido utilizado en la industria por clientes importantes como Ebay, Facebook, T-Mobile, Telefónica, entre muchos otros [69].

3.3.2 Base de datos de Vectores

Este tipo de bases de datos NoSQL se utiliza para almacenar datos en forma de vectores de altas dimensiones. En general, estos vectores se generan aplicando alguna función de transformación como por ejemplo algoritmos de machine learning o algoritmos de extracción de información, sobre datos como imágenes, audio o vídeo [70].

Este tipo de base de datos ha tomado relevancia debido a la inteligencia artificial, principalmente para generar los LLM (Large Language Models) [71], que luego se utilizan en aplicaciones de chats inteligentes que responden de forma natural. También se utilizan para guardar grandes representaciones numéricas de palabras para un futuro procesamiento [72].

Una de las ventajas de este tipo de almacenamiento es la capacidad de búsquedas semánticas de datos, en lugar de coincidencias exactas o parciales como se hace en las bases de datos relacionales. Esto significa que en una base de datos que implementa este tipo de almacenamiento, dada una imagen, puede retornar otras

imágenes similares, calculando la similitud sobre la imagen original utilizando vectores según métricas matemáticas [73].

Algunos ejemplos de este tipo de base de datos son Qdrant [74], Choma [75], Pinecone [76] entre otros.

3.3.3 Base de datos de series temporales

Una base de datos de series temporales es un sistema de software optimizado para almacenar y servir series temporales mediante pares de tiempo y valor [77].

Las primeras bases de datos de series temporales están asociadas a aplicaciones industriales que podían almacenar de forma eficiente valores medidos procedentes de equipos sensoriales, por ejemplo, para Internet de las Cosas (Internet of Things o IoT por sus siglas en inglés) [78].

Los repositorios de datos de series temporales utilizan algoritmos de compresión para gestionar los datos de manera eficiente. Aunque es posible almacenar datos de series temporales en muchos tipos de bases de datos diferentes. El diseño de estos sistemas es diferente al de las bases de datos relacionales, en donde se reducen las relaciones discretas a través de modelos referenciales.

Algunos de los motores de bases de datos NoSQL más populares que utilizan este tipo de almacenamiento son: Prometheus [79], Graphite [80] entre otros.

3.4 Servicios de base de datos NoSQL en la nube (Cloud Database Service)

Es una tecnología que permite la utilización de sistemas, servidores, aplicaciones o bases de datos remotas a través de internet. En algunos casos puede ser sin costo y en otros casos mediante una licencia con el proveedor del servicio.

Actualmente, gran parte de los servicios de internet aprovechan los beneficios que provee la computación en la nube. Dentro de los servicios en la nube, se distinguen tres modelos [81]:

- **SaaS** (Software as a Service), provee acceso a aplicaciones de software que se ejecutan en un servidor externo.
- **PaaS** (Platform as a Service), provee acceso a un entorno en donde los usuarios finales pueden almacenar aplicaciones o datos.
- **IaaS** (Infrastructure as a Service), provee acceso a recursos de computación como servidores, almacenamiento o redes, de forma remota, permitiendo al usuario administrador, gestionar las características de la infraestructura según sus necesidades.

A continuación, en la figura 15 se presentan los diferentes modelos de servicios en la nube y al tipo de usuario que está dirigido.

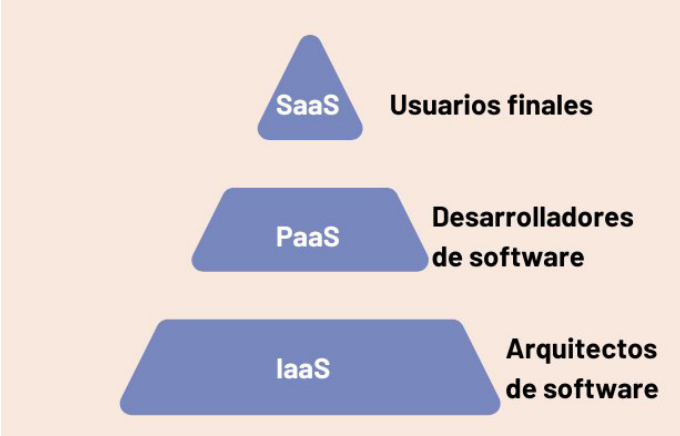

Figura 15. Diferentes capas en modelos de servicios en la nube.

El objetivo es la abstracción de la infraestructura de base de datos para el desarrollador de la aplicación y que el servicio se encargue automáticamente de proveer las instancias de la base de datos, automatizar el manejo del tráfico, garantizar disponibilidad de los datos en todo momento, entre otras características [82].

El usuario puede realizar configuraciones de infraestructura que sean necesarias dentro de las restricciones que provea el servicio, pero sin necesidad de encargarse de disponer y mantener el hardware directamente.

Muchos proveedores de bases de datos NoSQL ofrecen su propio servicio de DBaaS, algunos ejemplos son: MongoAtlas para MongoDB [83], Apache Cassandra Cloud [84], AuraDB para Neo4j [85] , Redis Cloud para redis [86], entre otros.

Además, proveedores de servicios de cloud, como Microsoft Azure[87], Amazon Web Services [88] o Google Cloud [89] proveen múltiples servicios de bases de datos en la nube, y también permiten soportar distintos modelos NoSQL dentro del mismo servicio, e incluso bases de datos relacionales. Un ejemplo es Azure Cosmos DB [90], un solo servicio que permite disponer de las bases de datos PostgreSQL [91] (relacional), MongoDB [29] (documental) y Apache Cassandra [84] (familia de columnas), según las necesidades del cliente.

3.5 Escalabilidad de Base de Datos

La escalabilidad es la habilidad que tiene un sistema de adaptarse a un crecimiento de carga y/o demanda sin perder calidad en los servicios ofrecidos [92].

La necesidad de escalar en una base de datos se debe a que existen nuevas aplicaciones de software que son utilizadas mundialmente por miles de usuarios generando un gran volumen de datos diariamente, y que necesitan ser accedidos de forma concurrente [93].

Existen dos posibilidades para escalar un sistema, escalamiento horizontal y escalamiento vertical. A continuación, en la figura 16 se presentan gráficamente ambas formas de escalar un sistema.

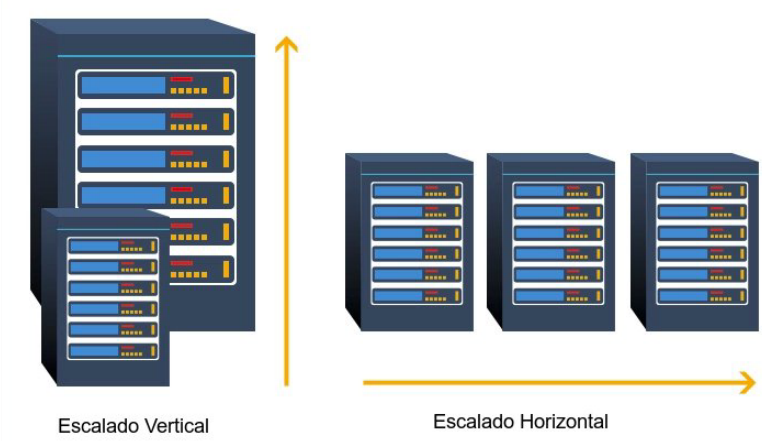

Figura 16. Escalamiento horizontal y escalamiento vertical de un sistema.

Escalabilidad vertical: la escalabilidad vertical, consiste en agregar prestaciones al mismo servidor para aumentar su capacidad de cómputo. Es decir, se tiene un solo servidor, y se le agrega por ejemplo, memoria y/o capacidad de CPU.

Escalabilidad horizontal: la escalabilidad horizontal, a diferencia de la vertical, consiste en el incremento de la cantidad de servidores. Generalmente, esto se realiza bajo demanda, es decir, cuando el sistema necesita más capacidad de cómputo, puede agregar más servidores paralelos, sin aumentar los recursos de los existentes, y cuando la demanda baje, se liberan los servidores no necesarios. Puede ser que todos los servidores posean las mismas prestaciones, o que exista un nodo maestro con más recursos, que coordine a varios servidores secundarios. Escalar una base de datos horizontalmente permite la distribución del trabajo y no concentrar el tráfico en un solo lugar.

La consistencia de datos en bases de datos NoSQL es más relajada que en una base de datos relacional. La consistencia eventual, permite que el sistema no sea consistente inmediatamente después de una transacción, pero eventualmente lo será La consistencia eventual, junto con el estado suave, hacen que las bases NoSQL sean adecuadas para escalar horizontalmente [92, 93].

Teniendo en cuenta lo anteriormente mencionado, las bases de datos NoSQL generalmente deciden relajar la consistencia de datos en favor de la tolerancia a particiones y disponibilidad.

Capítulo 4 - Motores de Bases de Datos NoSQL orientado a grafos utilizados

4.1 Introducción

Para seleccionar los motores a utilizar, uno de los principales criterios fue su popularidad [28]. En el caso de Neo4j [31], es el motor de base de datos orientado a grafos más popular. Memgraph [63] y NebulaGraph [64], eran los motores de bases de datos puramente orientados a grafos siguientes en popularidad.

Además, se tuvo en cuenta que los tres motores elegidos soportan el mismo lenguaje de consulta, Cypher [94] lo que simplificó el proceso de prueba al momento de realizar operaciones.

4.2 Motor de Base de Datos Neo4j

Neo4j [31] es un motor de base de datos orientado a grafos desarrollado en Java [24]. Fue lanzado en el año 2007, aunque su primera versión estable fue publicada en el año 2010. Cuenta con una licencia GNU (General Public License) y aunque el código fuente está disponible, tanto la edición community como la enterprise se distribuyen de forma propietaria, en el caso de la edición community de forma gratuita [95].

Este motor de base de datos utiliza el lenguaje de consulta Cypher [94] y las transacciones a la base de datos cumplen con las propiedades ACID [96].

A continuación, en la figura 17 y en la figura 18 se presenta un ejemplo de la sintaxis en Cypher [94] para una sentencia de creación de nodo y para la consulta de nodos respectivamente.

```
CREATE (:Person {name: “Bob”})-[:KNOWS]->(:Person {name: “Alice”});
```

Figura 17. Creación de un nodo Persona y la relación “conoce” con otro nodo Persona en Cypher.

```
MATCH (p:Person) WHERE p.name = "Bob" RETURN p;
```

Figura 18. Consulta de recuperación de un nodo de clase Person donde el nombre de la persona coincida con Bob.

Una de las principales características de Neo4j [31] es su performance sin comprometer la integridad de los datos. Neo4j [31] utiliza una combinación de varias

características, entre ellas, cuenta con almacenamiento de grafos nativos, es decir, no se necesita una conversión intermedia para transformar los datos de la base de datos a un grafo en memoria y viceversa. También es escalable y cumple con las propiedades ACID para garantizar consultas predecibles [96, 97].

Además, posee una gran variedad de productos tanto para modelar o transformar la información, como para hacer analítica, integración con aplicaciones de bajo nivel o visualizar la información. Algunos ejemplos son Neo4j AuraDB (servicio cloud) [85], Neo4j GraphQL Library (librería para integrar lenguaje GraphQL) [98] y Neo4j Bloom (herramienta de visualización de datos) [99].

Debido a su flexibilidad y su buen desempeño con grandes volúmenes de datos, Neo4j [31] se ha convertido en uno de los motores de bases de datos NoSQL orientado a grafos más popular en el mercado y es utilizado por empresas como Adobe, Novartis, Cisco, NASA y Comcast [33].

4.3 Motor de base de datos Memgraph

Fue lanzado en el año 2017 y está desarrollado en C++ [100]. Al igual que Neo4j, sus transacciones cumplen con las propiedades ACID y su lenguaje de consulta es Cypher [94]. Posee una licencia del tipo BSL (Business Source Licence), lo que significa que es un producto licenciado de código abierto, y provee libertad de modificación y utilización del código fuente [101].

Memgraph, utiliza almacenamiento en memoria principal, esto hace que para el procesamiento de datos y los cálculos en tiempo real se obtenga el menor tiempo posible. Además, está pensado para grandes volúmenes de datos que requieren análisis frecuentes y necesitan respuestas en tiempo real. El almacenamiento en memoria principal resulta en un aumento de performance respecto de Neo4j [31], la cual utiliza almacenamiento en disco de forma exclusiva [102].

Provee la posibilidad de configurar tres tipos de modos de almacenamiento [67]:

- En memoria transaccional (utiliza transacciones que cumplen ACID).
- En memoria analítica (agiliza la consulta de datos pero no garantiza transacciones ACID).
- En disco transaccional (equivalente al primer modo pero funciona en disco en lugar de memoria principal).

Algunas de las ventajas que que posee Memgraph [63] es que provee una potente interfaz gráfica integrada directamente en el producto, llamada Memgraph Lab, que no solo posee herramientas de visualización de datos y consultas, sino que también cuenta con herramientas integradas para generar reportes, importar datos desde distintos formatos como CSV o JSON, y migrar desde otras bases de datos relacionales como MySQL y PostgreSQL.

Memgraph [63] también posee librerías oficiales para diversos lenguajes e incluye docenas de algoritmos y funciones de agregación preparadas y optimizadas para cada lenguaje en particular.

4.4 Motor de base de datos NebulaGraph

Es el motor de base de datos orientado a grafos más reciente con respecto a Neo4j y Memgraph, fue lanzado en 2019 por Vesoft Inc [69]. Al igual que Memgraph está desarrollado principalmente en C++ [100].El tipo de licencia asociado es Apache 2.0, es completamente de código abierto, y provee libertad de reutilizar, modificar y distribuir el código fuente [103].

NebulaGraph [69] posee un lenguaje de consulta propio que se denomina nGQL (NebulaGraph Query Language), aunque también es compatible con el lenguaje Cypher [94].

A continuación, en la figura 19 se presenta una consulta que inserta dos nodos (denominados en nGQL como “vértices”), y crea una relación de conocimiento definida “knows”.

```
CREATE TAG IF NOT EXISTS Person(name string);
INSERT VERTEX Person(name) VALUES “1”:(”Bob”), “2”:(“Alice”);
INSERT EDGE knows() VALUES “1”->”2":();
```

Figura 19. Consulta para insertar dos nodos en nGQL.

En la figura 20, se conectan ambos nodos mediante la relación definida anteriormente.

```
CREATE (:Person {name: “Bob”})-[:KNOWS]->(:Person {name: “Alice”});
```

Figura 20. Consulta para insertar dos nodos en Cypher.

Al igual que como sucede con Memgraph [68] y Neo4j [31], las transacciones en NebulaGraph [69] cumplen con las propiedades ACID. [31, 68, 69]

NebulaGraph [69] posee características diferentes con respecto a Neo4j [31] y Memgraph [68]. Entre estas características se encuentra un esquema de tipado fuerte, es decir, que un atributo no puede cambiar su tipo de dato, mientras que los otros 2 motores de bases de datos, Neo4j y Memgraph, cuentan con un esquema flexible [31, 68, 69].

Además, su lenguaje de consulta nGQL provee interoperabilidad con la especificación OpenCypher [104] en la que se basa el lenguaje Cypher, permitiendo incorporar cláusulas de Cypher [94].

También soporta integración directa con algunos frameworks populares de procesamiento de datos, como Apache Spark [105], mediante la herramienta oficial NebulaGraph Exchange [106].

Capítulo 5 - Experimentación con motores de bases de datos orientados a grafos.

5.1 Introducción

En la experimentación realizada se presentan dos casos de estudio, el primer caso de estudio se basa en obtener una comparativa en un ambiente en donde se realizan 4 millones de operaciones, divididas en 2 millones de lecturas y 2 millones de escrituras de forma concurrente para una base de datos orientada a grafos que almacena información sobre usuarios de una red social.

El objetivo es observar el comportamiento de cada motor de base de datos orientado a grafos cuando es sometido a múltiples operaciones de lecturas y escrituras.

En una primera instancia, se instaló y configuró cada motor de base datos en un único servidor. Posteriormente, se configuró a cada uno de los motores de bases de datos en una red de 3 servidores diferentes.

El segundo caso de estudio se basa en medir el comportamiento de cada motor de bases de datos al importar un conjunto de datos de gran tamaño, en este caso los datos importados pertenecen a un conjunto de usuarios que recomiendan juegos. Luego, en una segunda parte de la prueba se realizó una consulta sobre los datos cargados con el objetivo de medir tiempos de respuestas. Este procedimiento se realizó utilizando las mismas configuraciones que el primer caso de estudio.

5.1.1 Condiciones técnicas de las pruebas

Las pruebas se llevaron a cabo en una Apple MacBook Air, que cuenta con un chip M1 de 8 núcleos con arquitectura ARM, y 8 GB de memoria RAM. Los motores de bases de datos fueron usados mediante la herramienta de virtualización Docker [107], la cual se limitó a 6 GB de memoria virtualizada.

En este caso se optó por Docker debido a que proporciona una solución que permite definir múltiples contenedores que simulan servidores distintos con una mínima configuración y que permite tener un ambiente controlado, personalizado y homogéneo para realizar las consultas [107].

5.2 Caso de estudio 1

Este primer caso de estudio se basa en un modelo de datos que representa usuarios que poseen un identificador, nombre y apellido.Estos usuarios pueden tener una relación de conocimiento con otros usuarios.

A continuación, en la figura 21, se presenta gráficamente un fragmento del grafo generado para este primer caso de estudio.

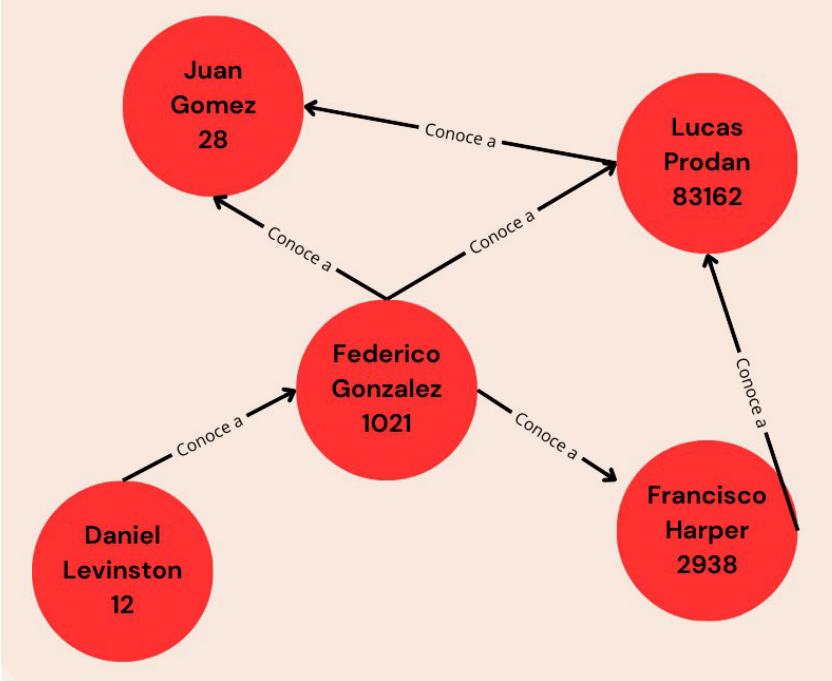

Figura 21. Parte del grafo generado para el caso de estudio 1.

Para la ejecución de múltiples operaciones de lecturas y escrituras, se desarrolló un algoritmo utilizando el lenguaje de programación Python [108] y Github [109] como repositorio del código .

Para cada uno de los 3 motores de bases de datos utilizados (Neo4j, Memgraph y NebulaGraph), se generaron 2000 hilos de ejecución, siendo un hilo una unidad mínima de ejecución que simula una consulta a la base de datos por parte de un usuario determinado.

Estos hilos se ejecutan de forma concurrente y realizan operaciones de lectura o escritura de manera aleatoria. El total de operaciones resulta en 2 millones de inserciones y 2 millones de lecturas, es decir 4 millones de operaciones en total.

A continuación, en la figura 22, se presenta un ejemplo de una operación de escritura (creación de un nodo) en el lenguaje Cypher para los 3 motores de base de datos utilizados.

```
CREATE (n:Person {name: $name, id: randomUUID()}) RETURN n.id AS node_id
```

Figura 22. Consulta utilizada para las operaciones de escritura.

A continuación, en la figura 23, se presenta un ejemplo de operación de lectura en lenguaje Cypher, representa una selección de los primeros 100 nodos.

```
MATCH (n) RETURN n LIMIT 100
```

Figura 23. Consulta utilizada para las operaciones de lectura.

Para este caso de estudio, se realizaron las siguientes pruebas:

- **Primera prueba**: se ejecuta el algoritmo desarrollado para cada base de datos instalada y configurada en un solo servidor.
- **Segunda prueba**: se ejecuta el algoritmo desarrollado para cada base de datos en una configuración de 3 servidores diferentes, un servidor primario y dos secundarios. Los servidores secundarios contienen una réplica de los datos del servidor primario. Esto implica que cada inserción de datos en el servidor primario se refleja en los otros dos servidores.

Los resultados obtenidos para cada motor de base de datos fueron almacenados en un archivo con extensión CSV donde se especifica el tipo de operación realizada (si fue de lectura o escritura), el hilo que la realizó, y la duración de la consulta realizada expresada en segundos.

A continuación, en la figura 24, se presenta un fragmento del formato establecido para mostrar los resultados obtenidos.

OPERACIÓN	HILO	TIEMPO
READ	10	0,02 S
READ	24	0,021 S
WRITE	11	0,03 S
WRITE	312	0,06 S
READ	2000	0,08 S
WRITE	671	1,02 S
READ	348	1,05 S
WRITE	1250	0,212 S
WRITE	998	0,872 S

Figura 24 . Ejemplo de CSV resultante luego de la ejecución del algoritmo.

Utilizando estos archivos resultantes de cada una de las pruebas, se presentan gráficas de comparación con el objetivo de evaluar el desempeño de cada motor de base de datos orientado a grafos a medida que se realizan las operaciones de lecturas y escrituras.

Debido a que los resultados obtenidos componen un gran volumen de información, los archivos CSVs resultaron de aproximadamente 4 millones de líneas. En este contexto, se divide a cada uno de los archivos CSVs resultantes en tramos de 250 mil líneas y se calculó la mediana [110] para cada uno de estos intervalos, de esta manera se presenta la información en un formato más claro.

5.2.1 Primer caso de estudio en un solo servidor

A continuación, en la figura 25, se presenta gráficamente la mediana de las operaciones de lecturas (2 millones) y escrituras (2 millones) medidas en segundos para el motor de base de datos Neo4j.

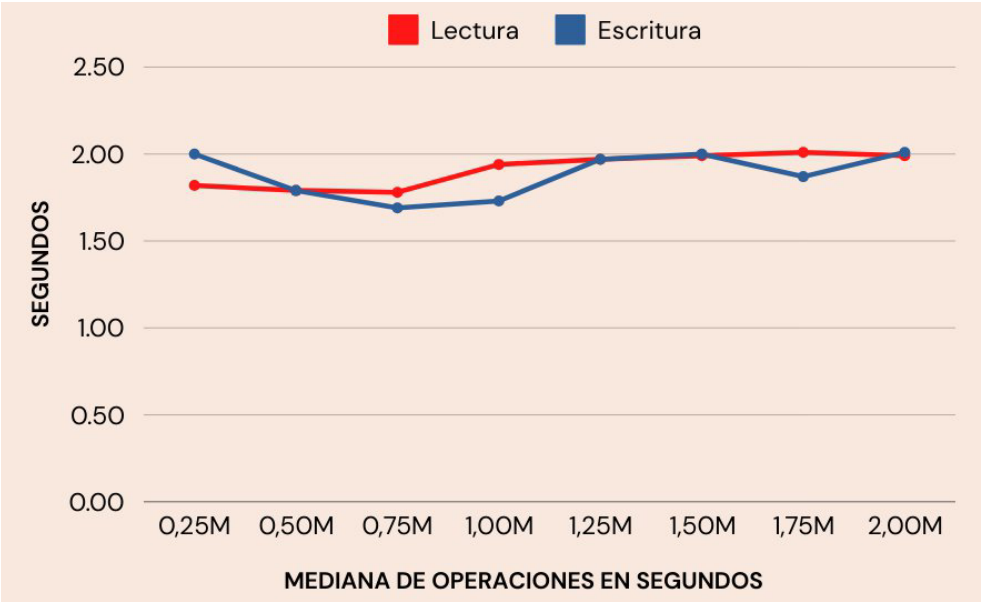

Figura 25. Evolución de Neo4j en un solo servidor para las operaciones de lecturas y escrituras.

En este caso, ambas operaciones (lecturas y escrituras) presentan gráficas similares.

A continuación, en la figura 26, se presenta gráficamente la mediana de las operaciones de lecturas (2 millones) y escrituras (2 millones) medidas en segundos para el motor de base de datos Memgraph.

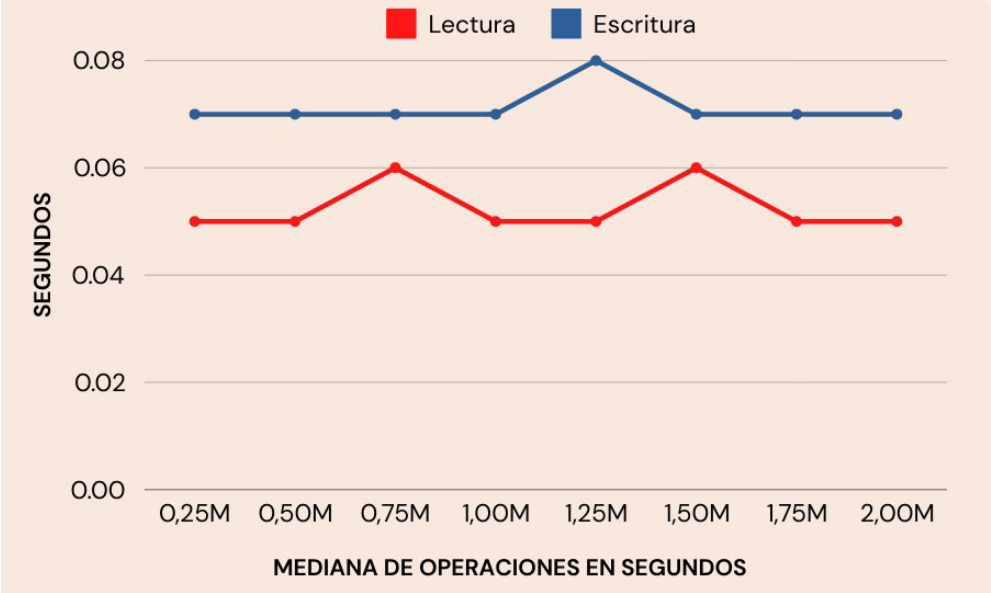

Figura 26. Evolución de Memgraph en un solo servidor para las operaciones de lecturas y escrituras.

En este caso se observa un comportamiento lineal en las operaciones de escritura. No obstante, se destaca un pico al llegar al millón de inserciones. Algo similar, sucede para las operaciones de lectura.

Si bien hay una diferencia entre los tiempos de las lecturas y las escrituras, esta diferencia es de una milésima de segundo.

A continuación, en la figura 27, se presenta gráficamente la mediana de las operaciones de lecturas (2 millones) y escrituras (2 millones) medidas en segundos para el motor de base de datos NebulaGraph.

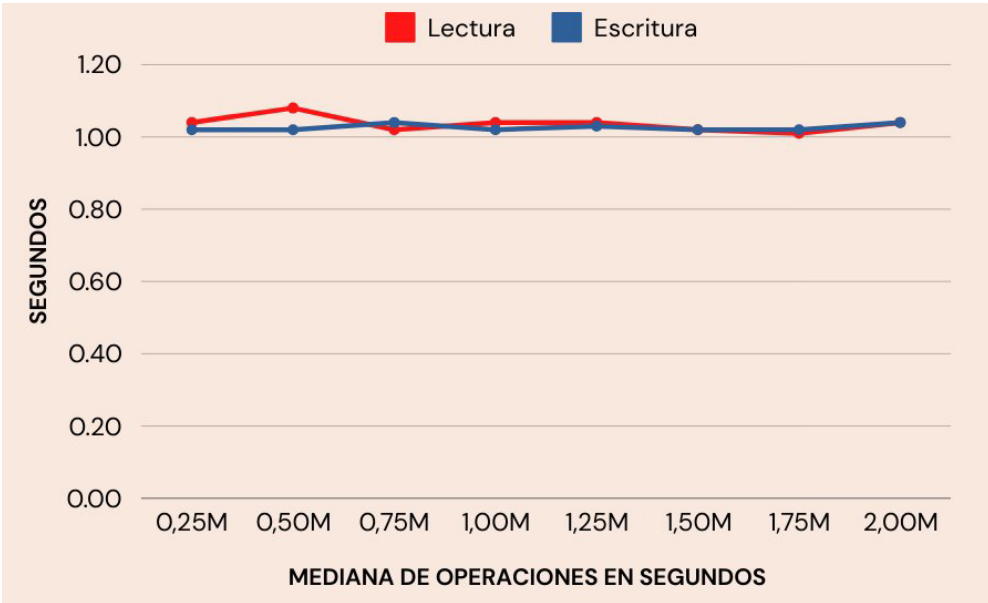

Figura 27. Evolución de NebulaGraph en un solo servidor para las operaciones de lecturas y escrituras.

En este caso al inicio de la prueba y hasta completar la carga de 500 mil nodos en el grafo, existe una leve diferencia entre las operaciones de lectura y las operaciones de escritura, luego se estabiliza y ambas poseen un comportamiento similar.

A continuación, en la figura 28, se presenta la comparativa entre operaciones de lectura para cada uno de los 3 motores de bases de datos utilizados.

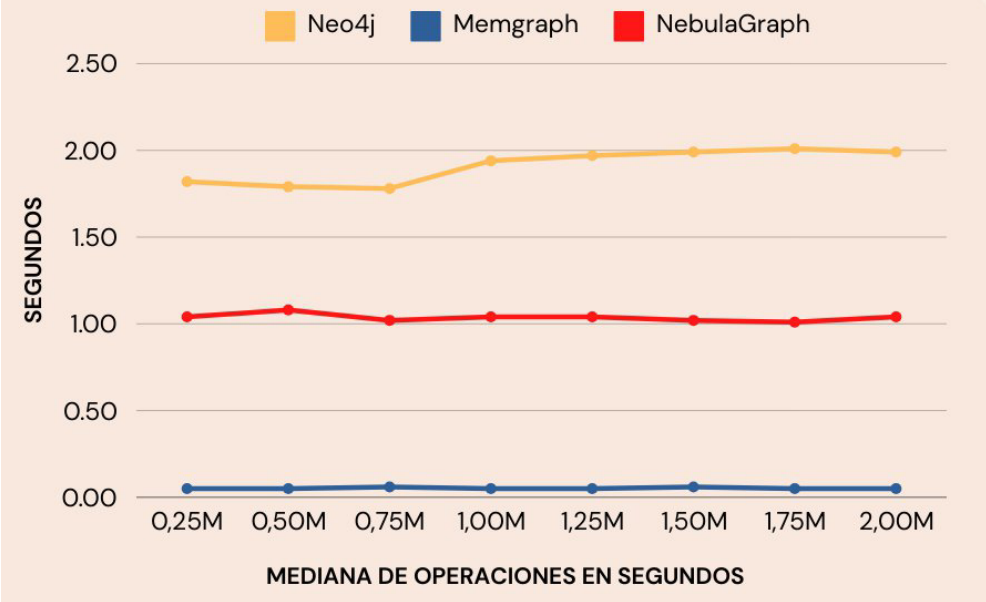

Figura 28. Comparativa de las operaciones de lectura en un solo servidor para los 3 motores de bases de datos.

En este caso existe una demora de Neo4j en comparación con los otros 2 motores de bases de datos. Esta demora representa aproximadamente el doble que NebulaGraph y unas 8 veces más que Memgraph.

A continuación, en la figura 29, se presenta la comparativa de las operaciones de escrituras para cada uno de los 3 motores de bases de datos utilizados.

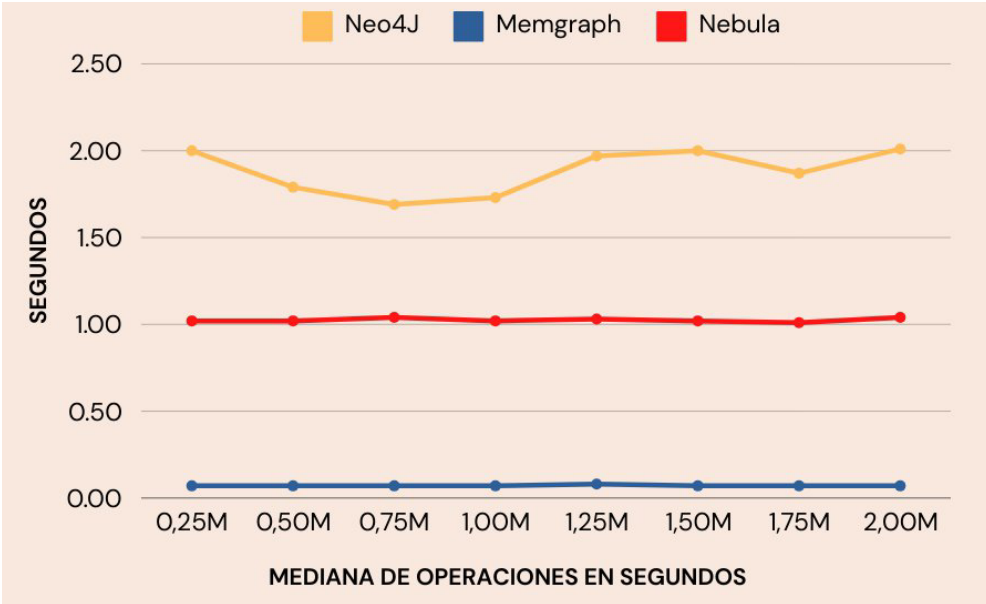

Figura 29. Comparativa de las operaciones de escritura en un solo servidor para los 3 motores de bases de datos.

En este caso, Neo4j posee el tiempo de respuesta más alto para las operaciones de escritura con respecto a Memgraph y NebulaGraph. Memgraph es el motor de base de datos con mejor tiempo de respuesta, seguido de NebulaGraph.

A continuación, en la figura 30, se presenta el tiempo total acumulado de cada motor de base de datos para completar las operaciones de lectura y escritura.

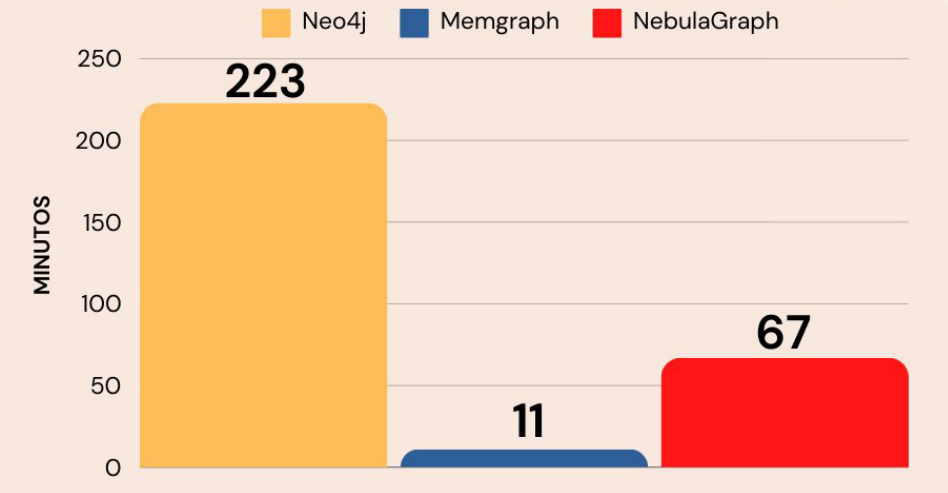

Figura 30. Tiempo total acumulado para completar las operaciones de lectura y escritura en un solo servidor.

En resumen, el comportamiento del motor de base de datos Neo4j posee tiempos menos lineales con respecto a los otros dos motores de bases de datos para las operaciones de lecturas y escrituras.

Este primer caso de estudio presenta una diferencia entre Neo4j y los otros 2 motores de bases de datos. Esta diferencia puede ser debido a que Neo4j generalmente se orienta al análisis de datos, mientras que los otros 2 motores de bases de datos se centran en la performance de las operaciones realizadas para este primer caso de estudio.

5.2.2 Primer caso de estudio en tres servidores

A continuación, en la figura 31, se presentan gráficamente las operaciones de lecturas (2 millones) y escrituras (2 millones) para el motor de base de datos Neo4j. En este caso, Neo4j se ha configurado para escalar horizontalmente en un cluster de tres servidores.

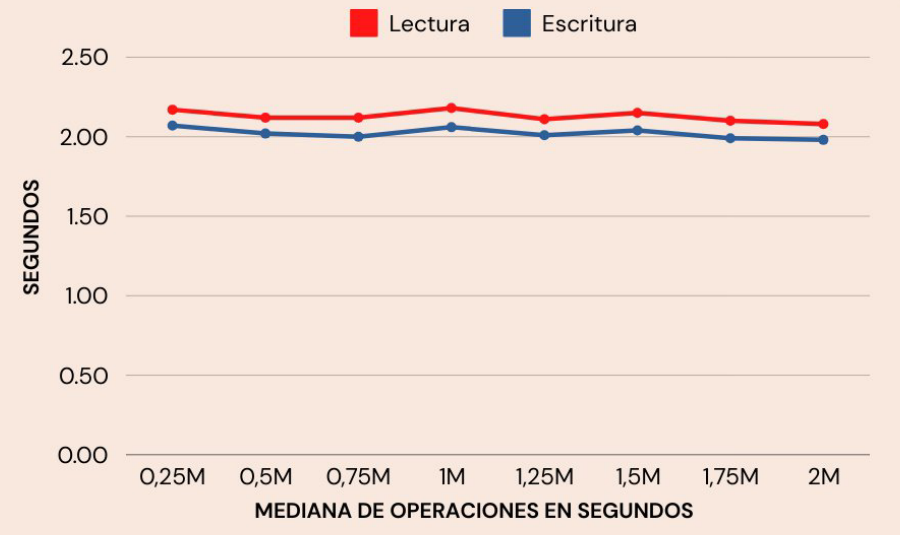

Figura 31. Evolución de Neo4j en tres servidores para las operaciones de lectura y escritura

En este caso, las escrituras tardan menos que las lecturas. Además, los tiempos se mantienen constantes durante la ejecución, oscilando entre los 2 y los 2,2 segundos con mínimas variaciones que son simultáneas entre las lecturas y las escrituras.

Luego, se procede a realizar el mismo caso de estudio pero para el motor de base datos Memgraph.

A continuación, en la figura 32, se presenta gráficamente el resultado de esta experimentación en Memgraph.

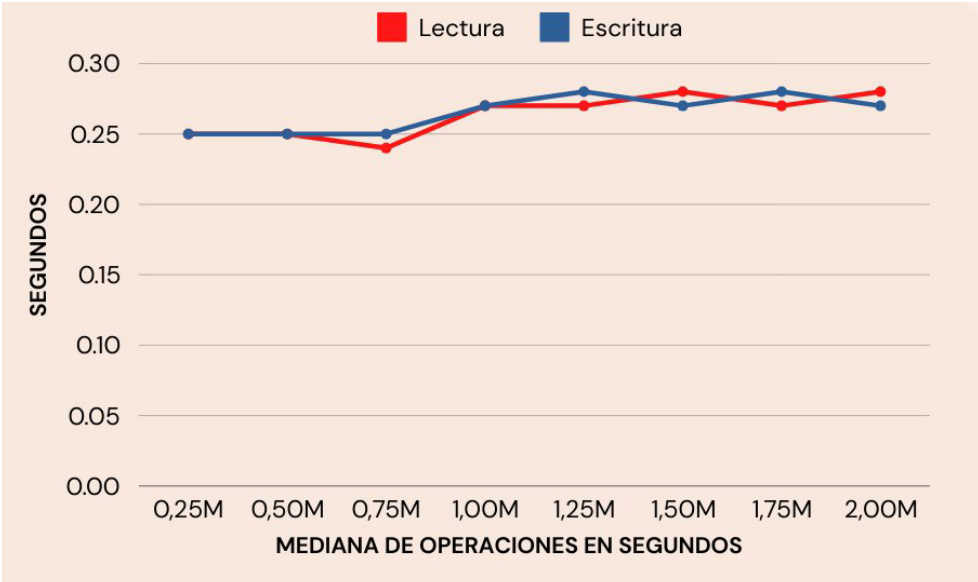

Figura 32. Evolución de Memgraph en tres servidores para las operaciones de lectura y escritura

En este caso no existe una diferencia a destacar entre las diferentes operaciones, ambas oscilan entre los 0,25 y los 0,3 segundos. Luego de 1.000.000 de operaciones, ocurre una leve variación en los tiempos de respuesta.

Finalmente, se procede a realizar el mismo caso de estudio con NebulaGraph. A continuación, en la figura 33, se presenta gráficamente el resultado de esta experimentación en NebulaGraph.

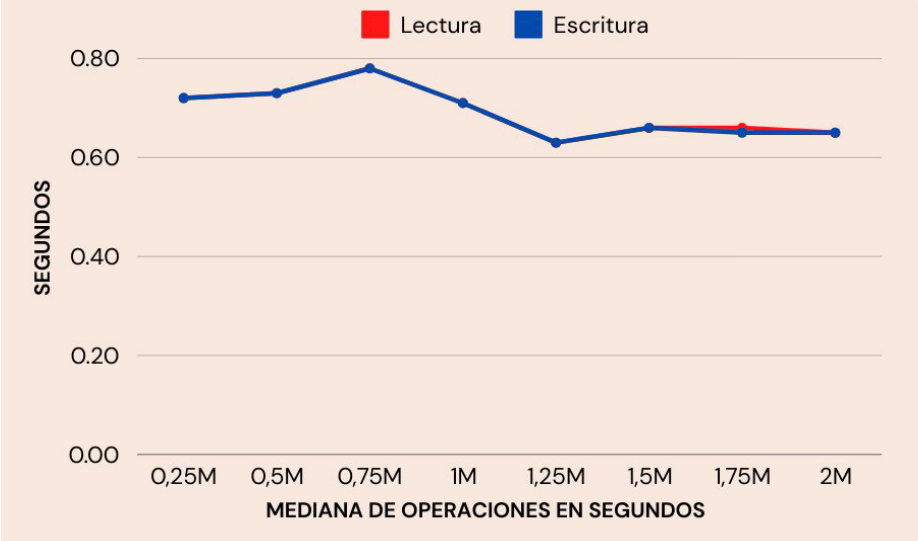

Figura 33. Evolución de NebulaGraph en tres servidores para las operaciones de lectura y escritura.

En este caso, se destaca que para ambas operaciones, el punto más alto se encuentra en los 750.000 datos en donde más demora unos 0,8 segundos. Además, el punto más bajo se da a los 1.250.000 datos, con un valor cercano a los 0,6 segundos.

A continuación, en la figura 34 se presenta la comparativa de las operaciones de lecturas para cada uno de los 3 motores de bases de datos utilizados y configurados para escalar horizontalmente en un clúster de 3 servidores.

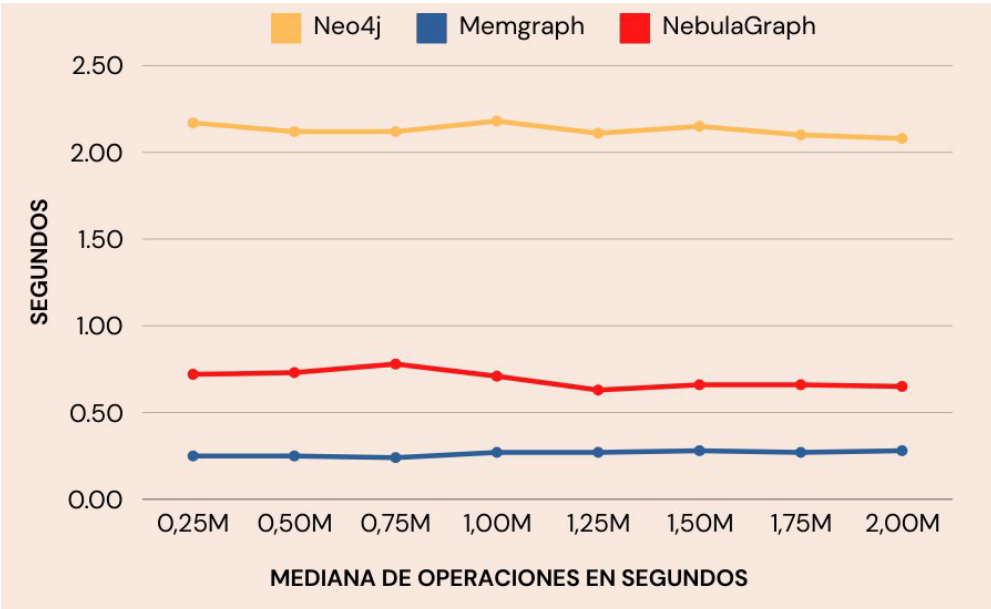

Figura 34. Evolución de NebulaGraph, Memgraph y Neo4j para la operación de lectura.

A continuación, en la figura 35, se presenta la comparativa de las operaciones de escrituras para cada uno de los 3 motores de bases de datos utilizados y configurados para escalar horizontalmente en un clúster de 3 servidores.

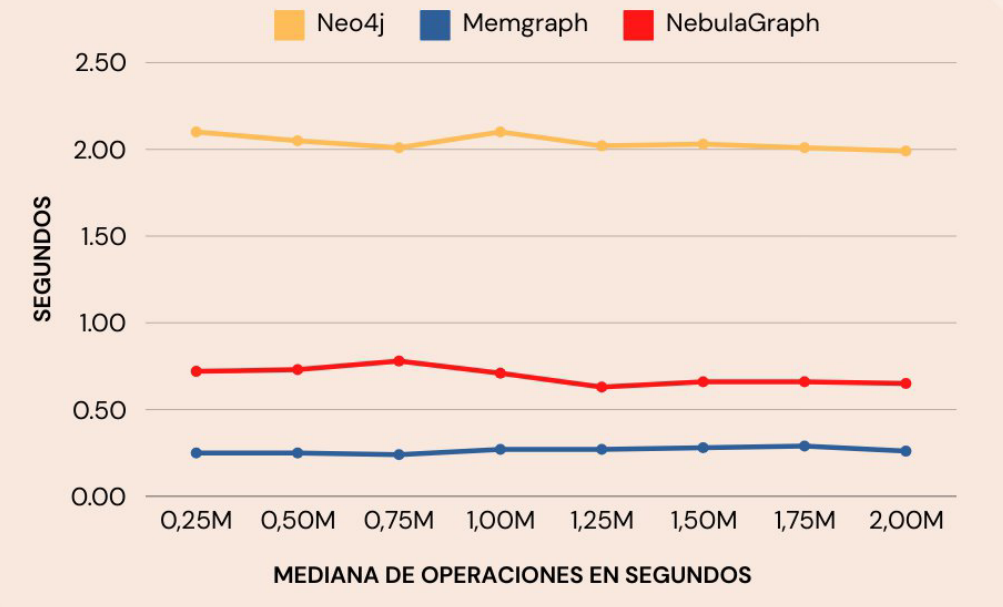

Figura 35. Evolución de NebulaGraph, Memgraph y Neo4j para la operación de escritura.

Ambas figuras (figura 33 y figura 34), presentan un comportamiento similar al obtenido en la experimentación 5.2.1.

Finalmente, en la figura 36, se presenta el tiempo total acumulado de cada motor de base de datos para completar las operaciones de lecturas y escrituras.

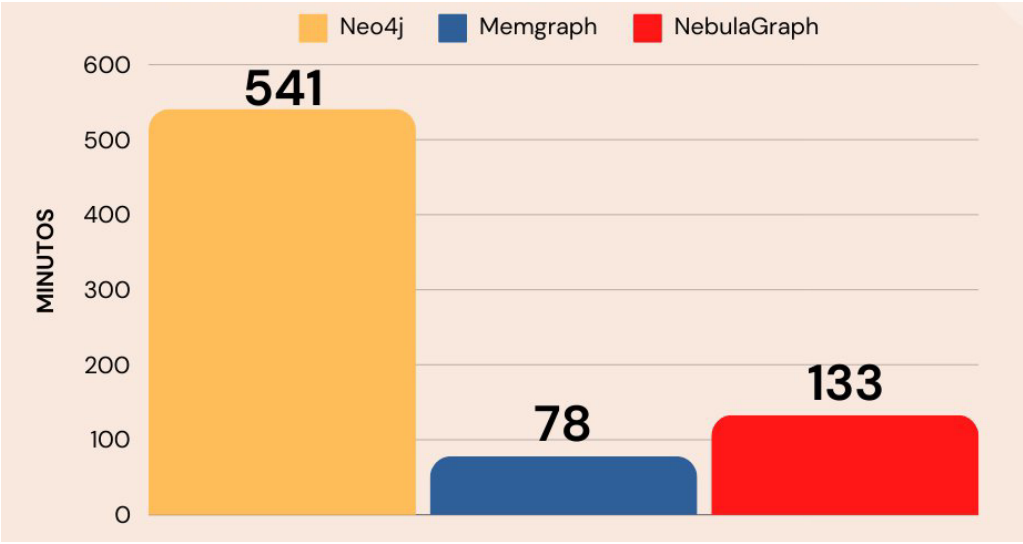

Figura 36. Tiempo de NebulaGraph, Memgraph y Neo4j para completar las operaciones de lecturas y escrituras.

En este caso, se tiene un resultado similar al obtenido en la sección 5.2.1. No obstante, la diferencia entre NebulaGraph y Memgraph es menos estrecha. Esto puede deberse a que Memgraph y NebulaGraph son adecuados para el procesamiento en tiempo real. La concurrencia y el diseño del caso de estudio favorece a estos dos motores de bases de datos [111, 112, 113].

5.2.3 Análisis comparativo de caso de estudio 1

En los 3 motores de bases (Neo4j, Memgraph y NebulaGraph), tanto en la configuración en un solo servidor, como al escalar horizontalmente en un cluster de 3 servidores, se destacan algunos aspectos importantes.

A continuación, en la figura 37 se presentan los tiempos promedios de carga en ambas configuraciones.

Base de datos	Un servidor	Tres servidores
Neo4j	223 MIN	541 MIN
Memgraph	11 MIN	78 MIN
NebulaGraph	67 MIN	133 MIN

Figura 37. Comparación de los tiempos de carga para los 3 motores de bases de datos.

En este caso existe una diferencia de Memgraph con respecto de los otros 2 motores de bases de datos. En la sección 5.2.1 (un solo servidor), es aproximadamente 6 veces más rápido que NebulaGraph y 22 veces más rápido que Neo4j. En la sección 5.2.2 (un cluster con 3 servidores) es aproximadamente 2 veces más rápido que NebulaGraph y 7 veces más rápido que Neo4j.

Además, los 3 motores de bases de datos, bajan su performance cuando se configuran para un cluster de 3 servidores. En este sentido el más afectado es Memgraph, que pasó de 11 minutos a 78 minutos (aproximadamente un 700% más lento).

A continuación, en la figura 38, se presenta el tiempo en promedio que necesita cada motor de bases de datos en realizar una operación de lectura o escritura.

Base de datos	Un servidor	Tres servidores
Neo4j	0,003345 S	0,008115 S
Memgraph	0,000165 S	0,00117 S
NebulaGraph	0,,001005 S	0,001995 S

Figura 38. Tiempo promedio que tarda cada motor de base de datos en realizar una operación de lectura o escritura.

A continuación, en la figura 39, se presenta un gráfico comparativo para los tres motores de bases de datos y el tiempo obtenido en cada configuración.

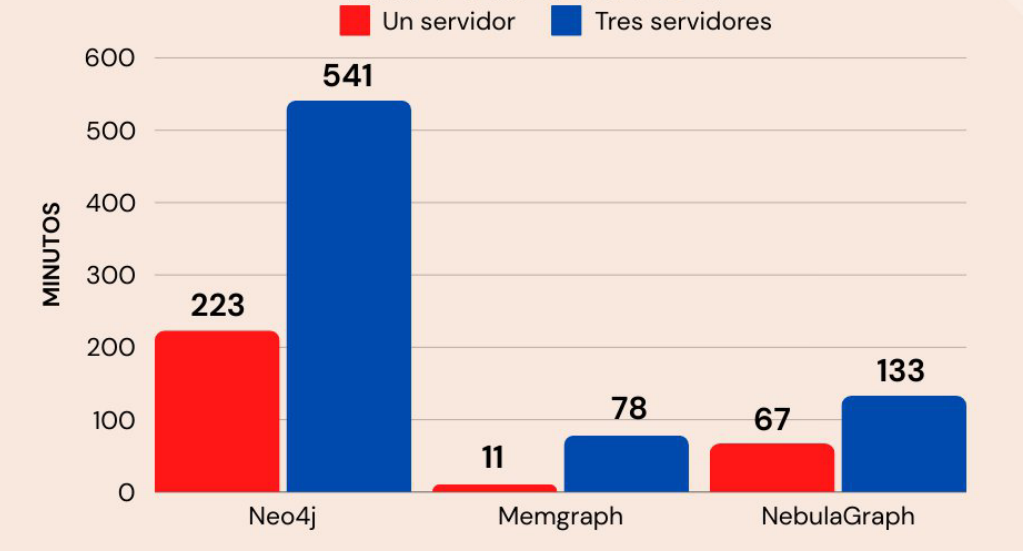

Figura 39. Comparación de los tiempos de carga para los 3 motores de bases de datos con un servidor y tres servidores

Se puede observar que los 3 motores de bases de datos sufren una baja de performance al escalar horizontalmente. La diferencia proporcional más notoria es la que se obtiene con Memgraph.

También, se destaca la demora que obtuvo Neo4j en comparación con los otros 2 motores de bases de datos, siendo el más lento de los tres.

Además, se destaca que existe una muy pequeña diferencia en los tiempos obtenidos por NebulaGraph al escalar horizontalmente a un cluster de 3 servidores. Siendo el motor de base de datos que mejor aprovecha la escalabilidad horizontal en este entorno de prueba.

La baja general en la performance para los 3 motores de bases de datos, se puede deber a que escalar horizontalmente con uno servidor maestro y dos servidores de réplicas, la información necesita ser replicada dos veces (una por cada réplica) por cada inserción realizada.

Asimismo, la alta concurrencia simulada en el caso de estudio, genera que la demora adicional en las operaciones de escrituras, también pueda impactar en la performance de las operaciones de lecturas, debido a la utilización de los recursos compartidos para procesar las distintas operaciones.

5.3 Caso de estudio 2

Para el segundo caso de estudio, se prueban los motores de bases de datos realizando una carga masiva de datos mediante un algoritmo realizado en Python (https://github.com/fedemozzon/scalability_test), el cual importará datos desde archivos CSV. El algoritmo hace uso de librerías específicas a cada motor de base de datos, para traducir las filas de los archivos CSV a nodos del grafo y sus respectivas relaciones.

Cantidad de datos a importar:

- 46.069 nodos que representan juegos.
- 1.111.111 nodos que representan usuarios.
- 20.000 relaciones que representan recomendaciones de juegos por parte de usuarios.

La importación se realiza de forma secuencial, es decir, se lee el CSV y se inserta fila por fila.

El algoritmo de carga calcula el tiempo que demora cada inserción y lo vuelca en un archivo CSV resultante que posee con una única columna que representa dicho tiempo. Luego, los 3 archivos CSV resultantes (uno para los juegos, uno para los usuarios y uno para las recomendaciones) se utilizan para realizar tanto los gráficos evolutivos de dicha importación, como para calcular el tiempo promedio de cada motor de base de datos.

A continuación, en la figura 40 se presenta un fragmento del modelo del grafo generado en los 3 motores de bases de datos.

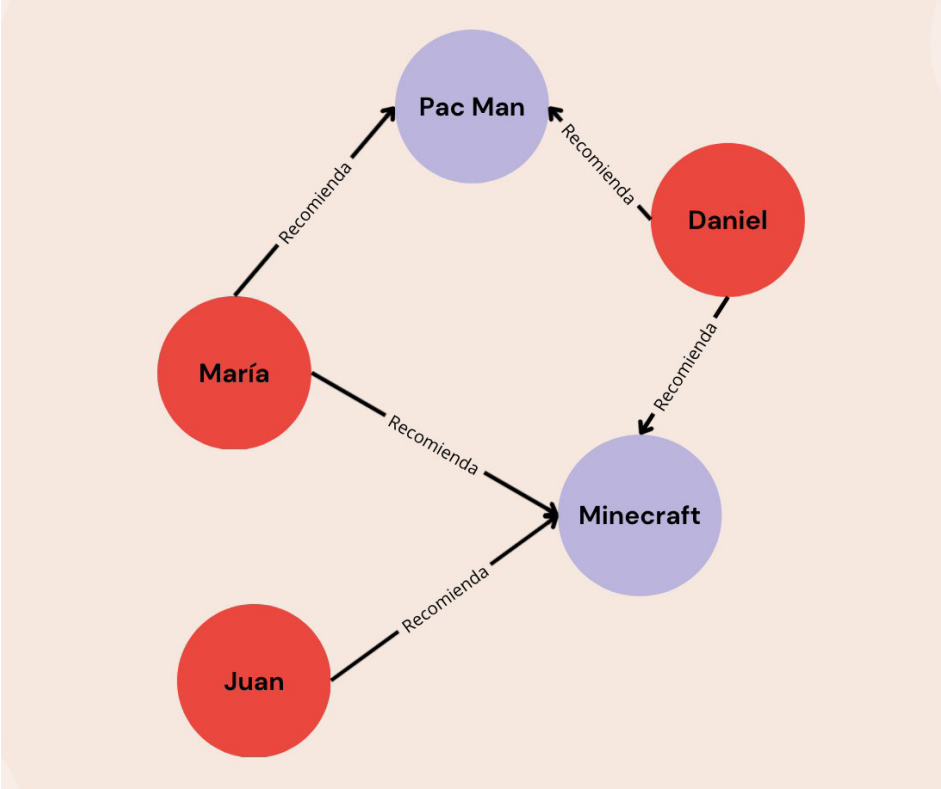

Figura 40. Fragmento del grafo generado para los 3 motores de bases de datos.

A continuación, en la figura 41, se presenta una consulta a ejecutar en los 3 motores de bases de datos, con el objetivo de realizar métricas de performance.

```
MATCH (u1:User)-[:RECOMMENDS]->(g:Game)<-[:RECOMMENDS]-(u2:User)
WHERE u1.user_id <> u2.user_id
RETURN u1.user_id, u2.user_id, g.title
```

Figura 41. Consulta en Cypher utilizada para medir tiempos en cada motor de base de datos.

Esta consulta retorna los juegos que fueron recomendados por más de un usuario distinto.

El objetivo principal de este caso de estudio es determinar en los 3 motores de bases de datos cual es la performance para la consulta planteada.

5.3.1 Generación del grafo en cada motor de base de datos a partir de archivos CSV - Un servidor

A continuación, en la figura 42 se presenta el tiempo en minutos que demoró cada uno de los 3 motores de bases de datos para importar los archivos CSV y generar el grafo correspondiente.

Base de datos	Juegos	Usuarios	Recomendaciones	Total
Neo4j	1,3 MIN	37,03 MIN	43,33 MIN	81 MIN
Memgraph	1,4 MIN	27,7 MIN	133,33 MIN	161 MIN
NebulaGraph	0,8 MIN	18,51 MIN	110 MIN	129 MIN

Figura 42. Tiempos obtenidos en minutos para la carga de datos.

No se observan diferencias importantes entre los 3 motores de bases de datos con respecto al tiempo de importación de los datos. NebulaGraph demostró en general una mejor performance. En el caso de las recomendaciones, Neo4j resultó ser más rápido pero no obtuvo esa ventaja en el caso de importación de nodos referentes a juegos y usuarios, en general Neo4j se comportó de forma más eficiente al momento de importar relaciones entre nodos del grado que NebulaGraph y Memgraph.

A continuación, en la figura 43 se presenta el tiempo obtenido por Neo4j en cargar el conjunto de datos según el tipo de nodo del grafo y las relaciones entre los mismos.

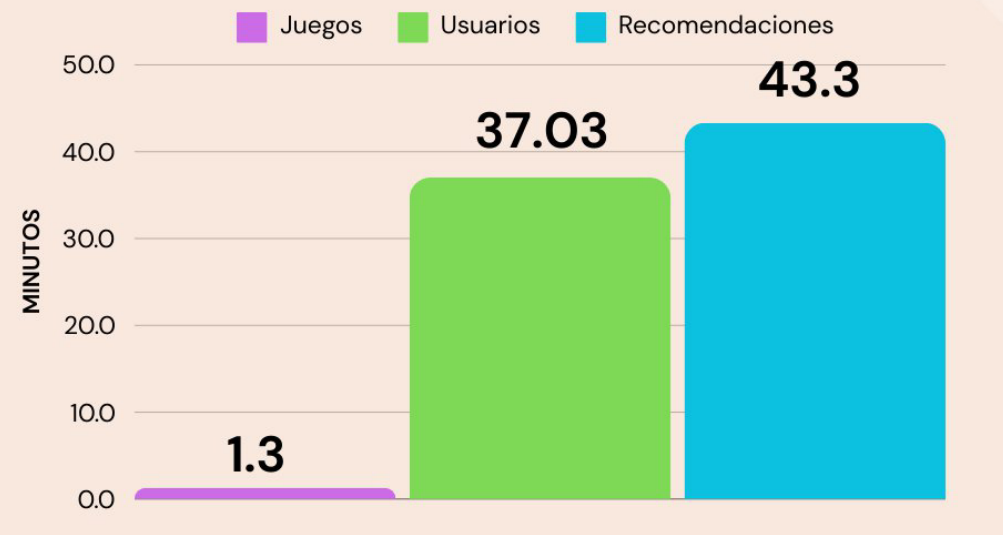

Figura 43. Tiempos obtenidos por Neo4j para la generación del grafo.

Existe una notable diferencia entre la generación de nodos que representan juegos respecto a los nodos que representan usuarios y las relaciones entre ambos. Hay que tener en cuenta que el conjunto de los nodos que representan juegos es menor al conjunto de nodos que representan usuarios.

A continuación, en la figura 44 se presenta el tiempo que demoró Memgraph en cargar el conjunto de datos según el tipo de nodo del grafo y las relaciones entre los mismos.

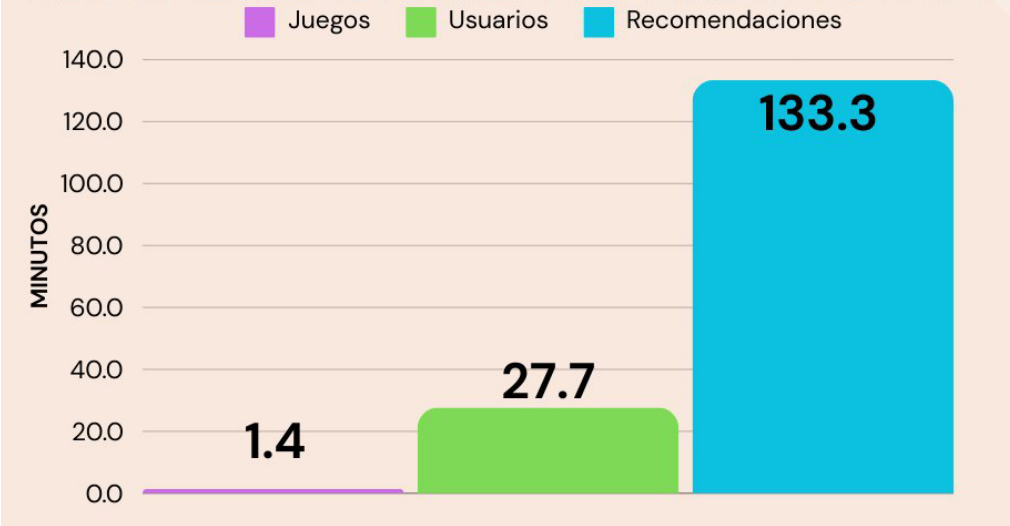

Figura 44. Tiempos obtenidos por Memgraph para la generación del grafo.

En este caso los tiempos entre importar los nodos que representan juegos y los nodos que representan usuarios presentan valores no tan distantes. Hay que tener en cuenta que el conjunto de los nodos que representan juegos es menor al conjunto de nodos que representan usuarios. No obstante, el tiempo para las relaciones es mucho más alto. En general Memgraph no resultó eficiente al momento de establecer las relaciones en el grafo.

A continuación, en la figura 45 se presenta el tiempo que demoró NebulaGraph en cargar el conjunto de datos según el tipo de nodo del grafo y las relaciones entre los mismos.

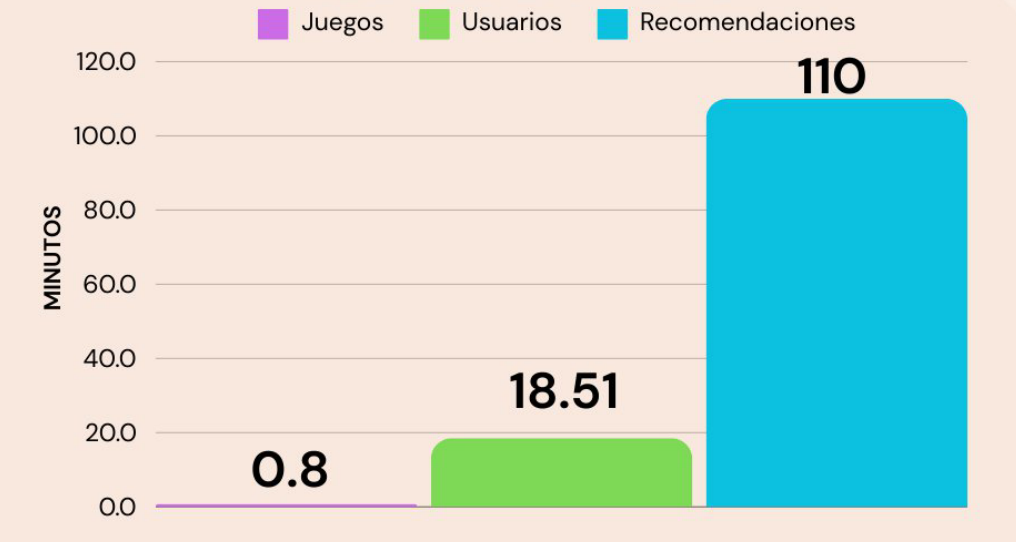

Figura 45. Tiempos obtenidos por NebulaGraph para la generación del grafo.

5.3.2 Medianas – Juegos - Un servidor

A continuación, en la figura 46 se presenta el gráfico con las medianas obtenidas para las escrituras en Neo4J al momento de generar los nodos del grafo que representan juegos utilizando un clúster con un solo servidor. Al ser muchos los datos importados, al igual que el caso de estudio 1 los gráficos se encuentran segmentados en tramos para mejor lectura, representando el eje horizontal el conteo de datos, y el eje vertical el tiempo medio obtenido.

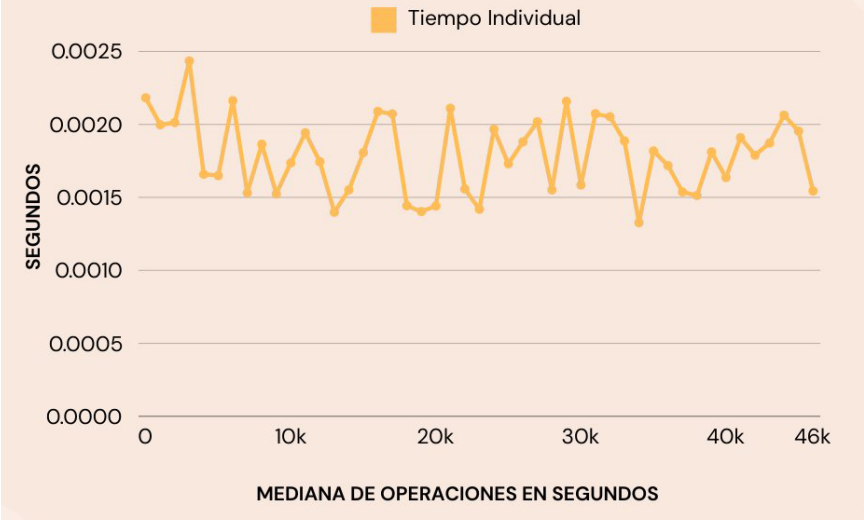

Figura 46. Evolución de Neo4j en un solo servidor para la carga de los nodos que representan juegos.

En este caso, Neo4j presenta una oscilación entre los 0,0015 segundos y los 0,0025 segundos por operación.

A continuación, en la figura 47 se presenta la evolución de Memgraph para la carga de los nodos que representan juegos.

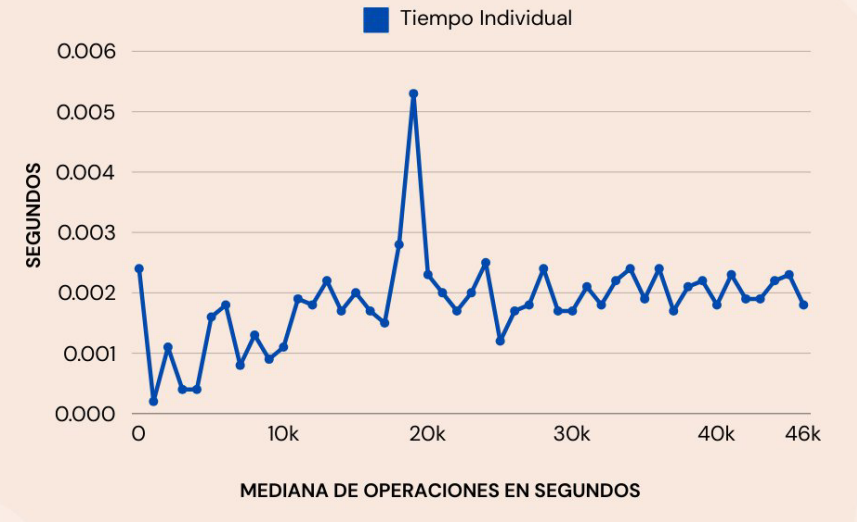

Figura 47. Evolución de Memgraph en un solo servidor para la carga de los nodos que representan juegos.

En este caso, para cargar los nodos que representan juegos, Memgraph se mantiene entre los 0,001 segundos y los 0,003 segundos por operación. Exceptuando un determinado momento en donde se presenta un pico que llega a los 0,006 segundos. Estos tiempos se comportan con mayor variación que los de Neo4j.

A continuación, en la figura 48 se presenta la evolución de NebulaGraph para la carga de los nodos que representan juegos.

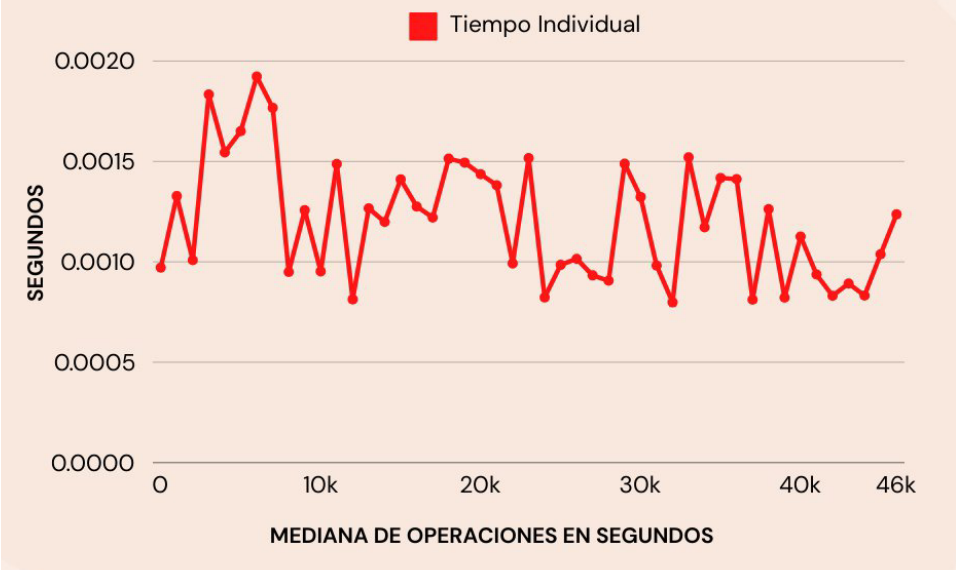

Figura 48. Evolución de NebulaGraph en un solo servidor para la carga de los nodos que representan juegos.

En este caso, los tiempos oscilan entre los 0,0008 segundos y los 0,002 segundos. Al inicio aparecen los tiempos más altos y luego se estabiliza.

A continuación, en la figura 49 se presenta una comparativa de la carga de los nodos que representan juegos para los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph).

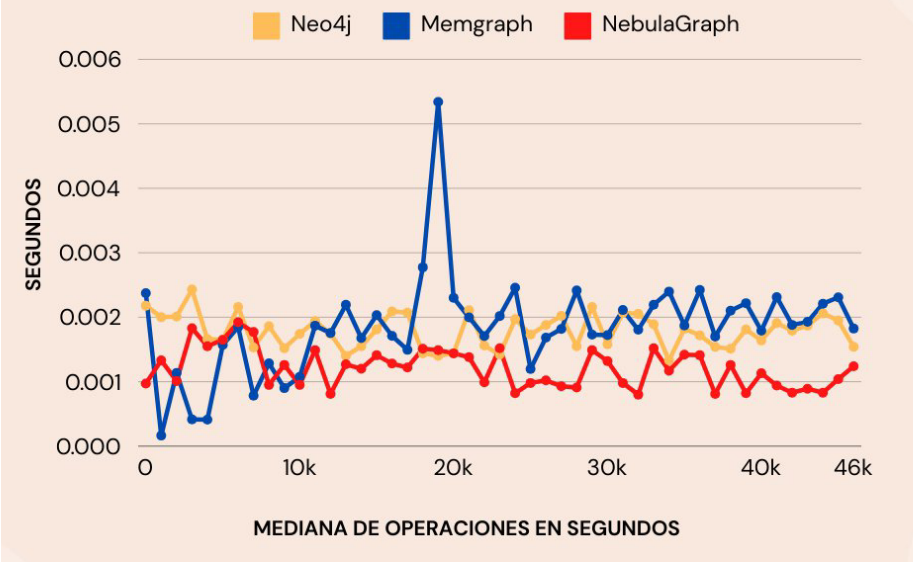

Figura 49. Evolución de la carga de los nodos que representan juegos para Neo4j, Memgraph y NebulaGraph.

A continuación, en la figura 50, se presentan los tiempos totales obtenidos por cada motor de bases de datos al generar los nodos que representan juegos.

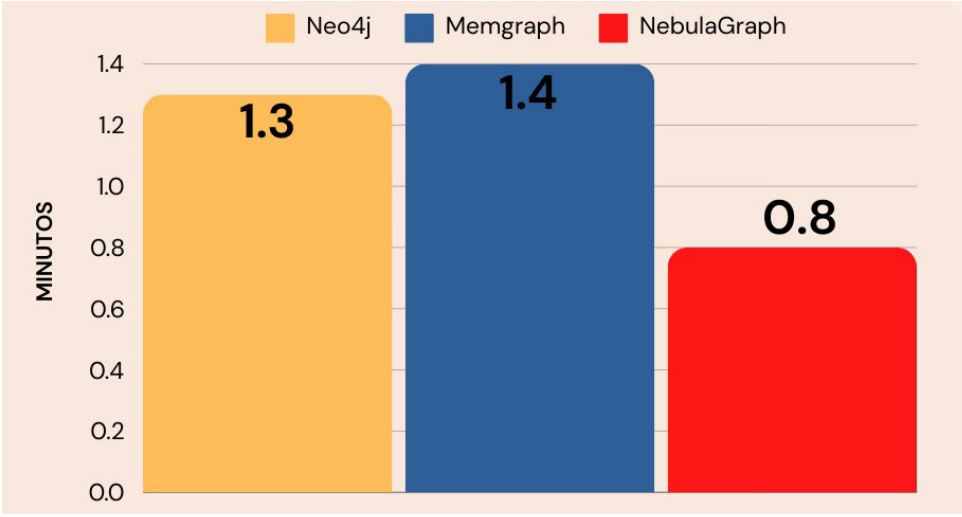

Figura 50. Tiempos totales para los 3 motores de base datos (Neo4j, Memgraph y NebulaGraph) para la generación de los nodos que representan juegos en el grafo.

Los 3 motores de bases de datos obtuvieron tiempos cercanos, siendo NebulaGraph el más rápido.

5.3.3 Medianas - Usuarios - Un servidor

A continuación, la figura 51, presenta los tiempos obtenidos en Neo4j para la carga de los nodos que representan usuarios.

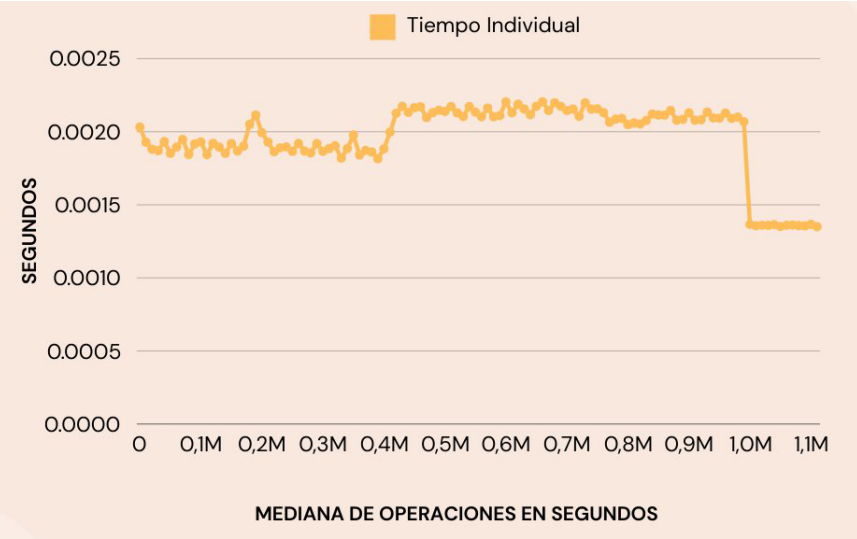

Figura 51. Evolución de Neo4j en un solo servidor para la carga de los nodos que representan usuarios.

En este caso, los tiempos resultan en general estables, oscilando muy poco. En el tramo final del caso de estudio, el tiempo promedio baja. Aunque parece una reducción importante, la diferencia resulta de solo 0,0005 segundos.

A continuación, la figura 52, presenta los tiempos obtenidos en Memgraph para la carga de los nodos que representan usuarios.

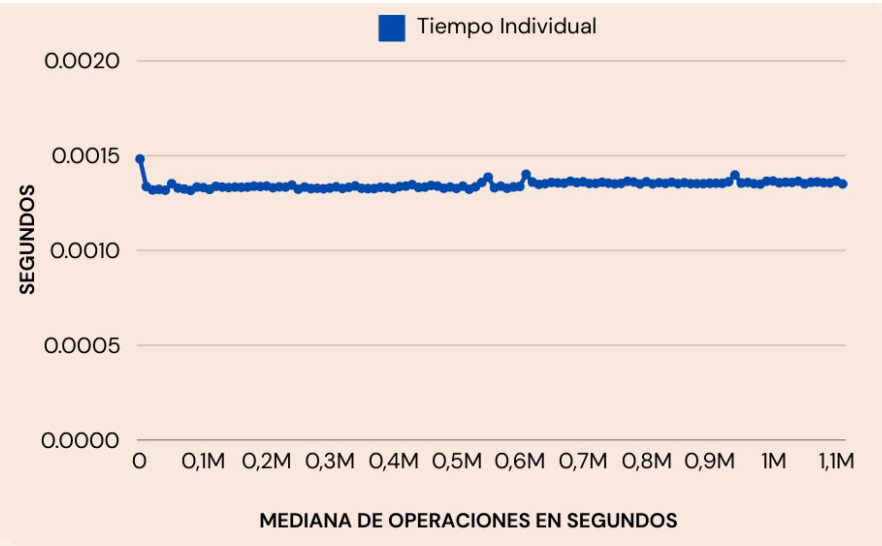

Figura 52. Evolución de Memgraph en un solo servidor para la carga de los nodos que representan usuarios.

En este caso, Memgraph presenta tiempos regulares en comparación con el motor anterior. Estos tiempos se mantienen estables hasta el final de la carga.

A continuación, la figura 53, presenta los tiempos obtenidos en NebulaGraph para la carga de los nodos que representan usuarios.

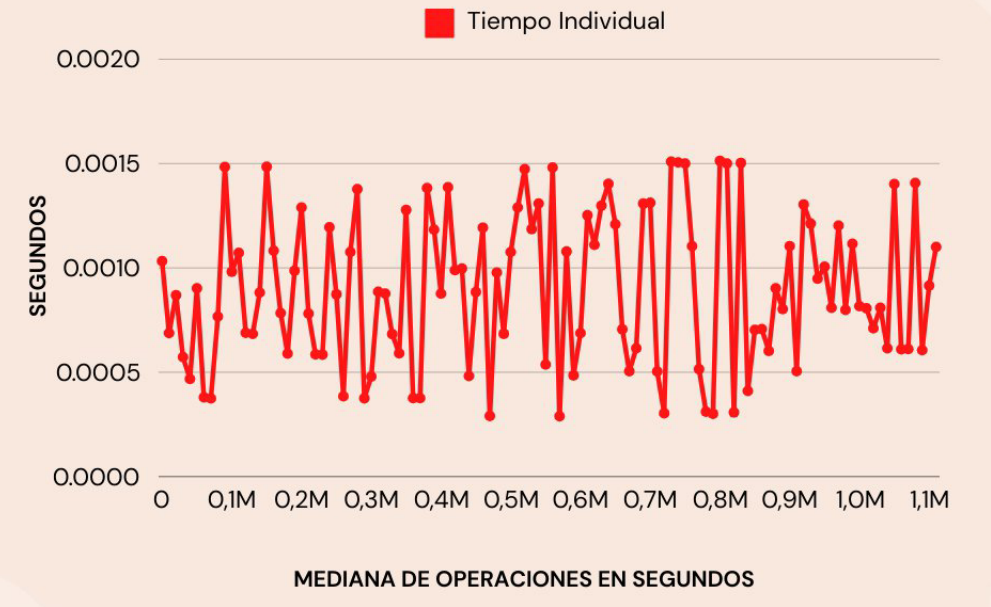

Figura 53. Evolución de NebulaGraph en un solo servidor para la carga de los nodos que representan usuarios.

En este caso, los tiempos resultantes poseen un comportamiento irregular. En ningún momento la carga se estabiliza. La cantidad de nodos que representan usuarios es un conjunto mayor que los generados para los nodos que representan juegos.

A continuación, la figura 54, presenta una comparativa de los tiempos obtenidos por los 3 motores de bases de datos (NebulaGraph, Memgraph y Neo4j.) para la carga de los nodos que representan los usuarios.

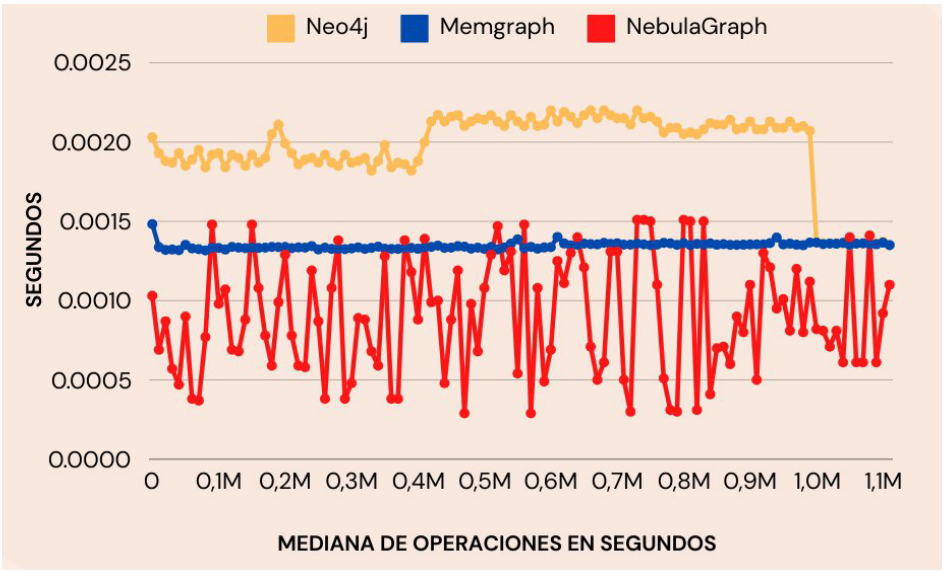

Figura 54. Evolución de la carga de los nodos que representan usuarios para Neo4j, Memgraph y NebulaGraph.

A pesar de la oscilación de los tiempos que presenta el motor de base de datos NebulaGraph, es el que mejores tiempos obtiene. En el caso de Memgraph, es el motor de base de datos que presenta los resultados más regulares en comparación con

NebulaGraph y Neo4j. Este último resultó ser el que menos performante obtiene, aunque presenta tiempos regulares.

A continuación, la figura 55, presenta una comparativa de los tiempos totales obtenidos por los 3 motores de bases de datos (NebulaGraph, Memgraph y Neo4j.) para la carga de los nodos que representan los usuarios.

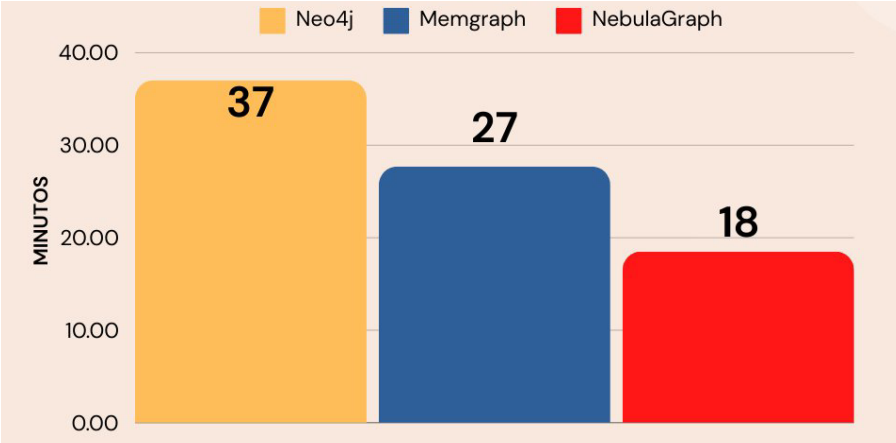

Figura 55. Tiempos totales para los 3 motores de base datos (Neo4j, Memgraph y NebulaGraph) para la generación de los nodos que representan usuarios en el grafo.

5.3.4 Medianas - Recomendaciones - Un servidor

A continuación, la figura 56, presenta los tiempos obtenidos en Neo4j para la carga de las relaciones que representan las recomendaciones, en un solo servidor.

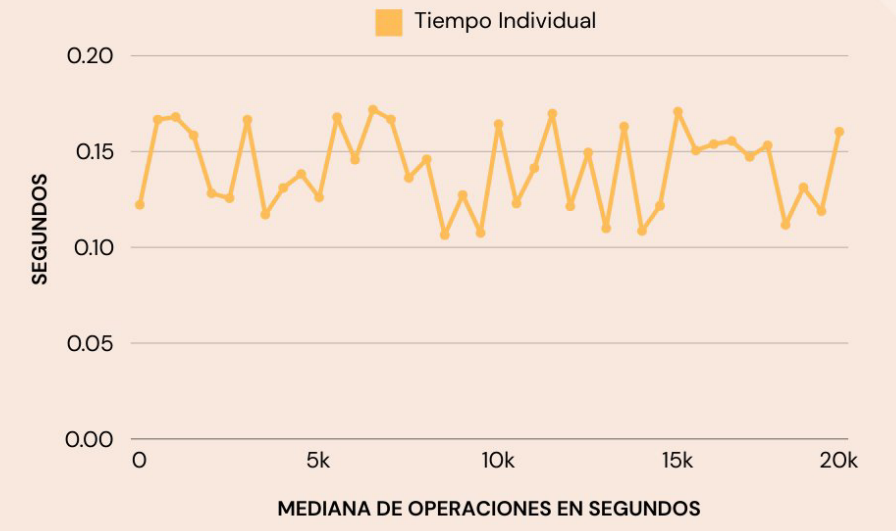

Figura 56. Evolución de Neo4j en un solo servidor para la carga de las relaciones entre los nodos del grafo.

En este caso, a pesar de que Neo4j presenta tiempos poco regulares.

A continuación, la figura 57, presenta los tiempos obtenidos en Memgraph para la carga de las relaciones que representan las recomendaciones.

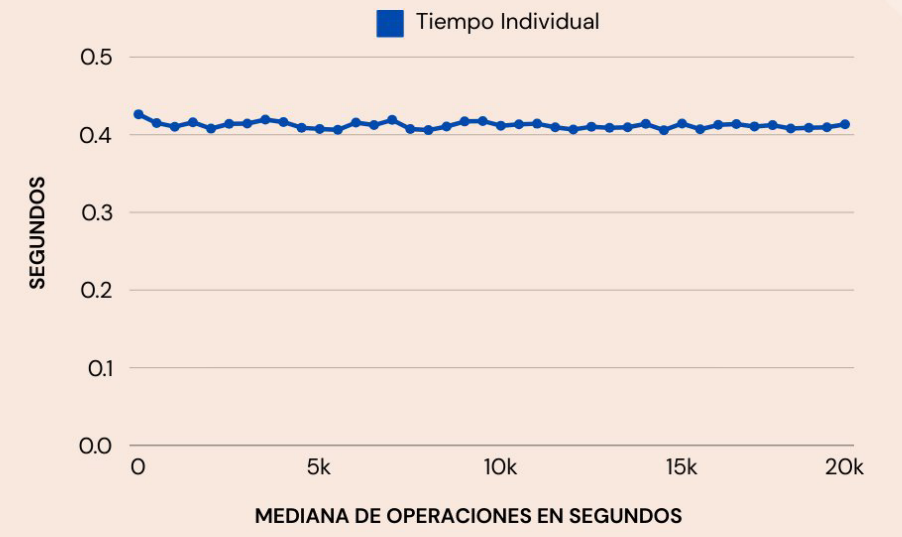

Figura 57. Evolución de Memgraph en un solo servidor para la carga de las relaciones entre los nodos del grafo.

En este caso, los tiempos se mantienen de forma regular. No obstante, generar las relaciones en el grafo, para Memgraph, requiere de más tiempo que generar los nodos, en promedio, el tiempo es 200 veces mayor. La complejidad para generar la relación es mayor, primero se deben buscar los nodos involucrados y luego establecer la conexión.

A continuación, la figura 58, presenta los tiempos obtenidos en NebulaGraph para la carga de las relaciones que representan las recomendaciones.

Figura 58. Evolución de NebulaGraph en un solo servidor para la carga de las relaciones entre los nodos del grafo.

En este caso los tiempos resultan regulares, oscilando entre los 0,3 y los 0,35 segundos. NebulaGraph resultó ser en promedio 0,1 segundo más rápido que Memgraph al momento de generar las relaciones entre los nodos del grafo.

A continuación, la figura 59, presenta una comparativa de los tiempos obtenidos por los 3 motores de bases de datos (NebulaGraph, Memgraph y Neo4j.) para la generación de las relaciones en el grafo.

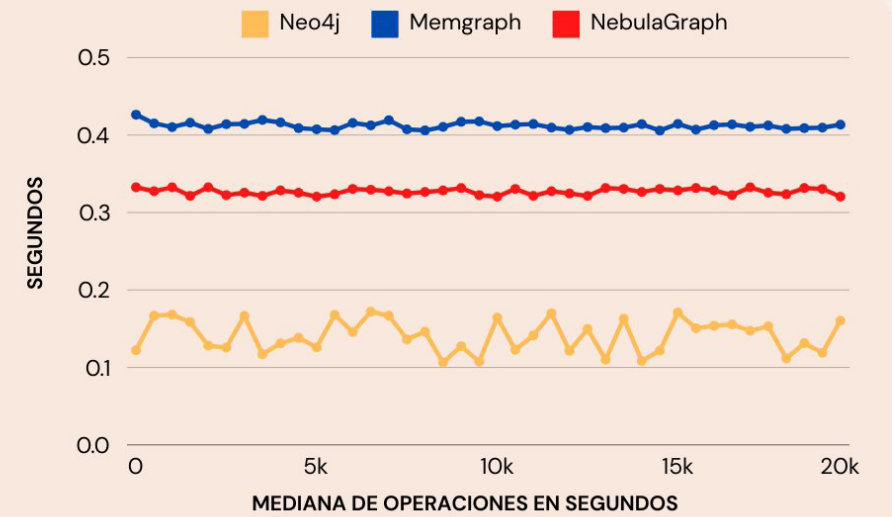

Figura 59. Evolución de la carga de las relaciones entre los nodos del grafo para los 3 motores de bases datos (Neo4j, Memgraph y NebulaGraph).

Se observa que Neo4j, posee mejores tiempos al momento de generar las relaciones entre los nodos del grafo, oscilando entre los 0,1 segundo y los 0,15 segundos aproximadamente. Esto representa, que en promedio, resultó ser aproximadamente 2 veces más rápido que NebulaGraph y 3 veces más rápido que Memgraph.

A continuación, la figura 60, presenta una comparativa de los tiempos totales obtenidos por los 3 motores de bases de datos (NebulaGraph, Memgraph y Neo4j.) para la carga de las relaciones entre los nodos del grafo.

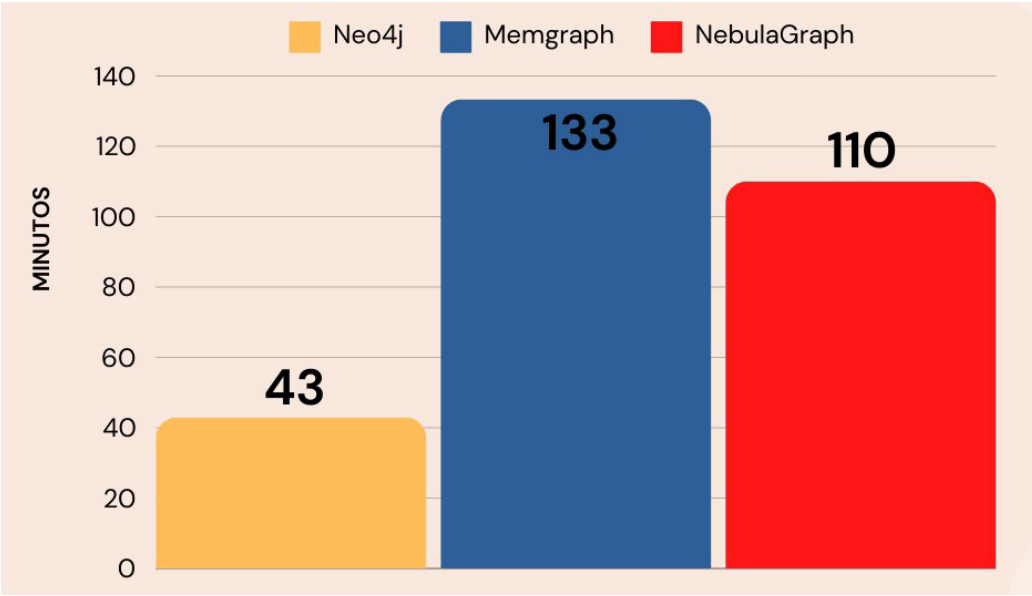

Figura 60. Tiempos totales para los 3 motores de base datos (Neo4j, Memgraph y NebulaGraph) para la generación de las relaciones entre los nodos del grafo.

5.3.5 Ejecución de una consulta implementada en Cypher en los 3 motores de bases de datos - Un servidor

A continuación, en la figura 61, se presenta la consulta desarrollada en Cypher para ejecutar en los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph).

Esta consulta busca los juegos que fueron recomendados por más de un usuario distinto, y retorna subgrafos conformados por los juegos encontrados y los usuarios que lo recomendaron.

```
MATCH (u1:User)-[:RECOMMENDS]->(g:Game)<-[:RECOMMENDS]-(u2:User)
WHERE u1.user_id <> u2.user_id
RETURN u1.user_id, u2.user_id, g.title
```

Figura 61. Consulta en Cypher utilizada para medir tiempos en cada motor de base de datos.

A continuación, en la figura 62, se presentan los tiempos expresados en segundos para la ejecución de la consulta en cada uno de los 3 motores de bases de datos.

Base de datos	Tiempo
Neo4j	18,1 S
Memgraph	15,7 S
NebulaGraph	6,2 S

Figura 62. Tiempos expresados en segundos para la ejecución de la consulta en los 3 motores de bases de datos en un solo servidor

NebulaGraph presenta el mejor tiempo de los 3 motores de bases de datos, y las 3 se encontraron por debajo de los 20 segundos.

A continuación, en la figura 63, se presenta una gráfica de los resultados obtenidos para la ejecución de la consulta en los 3 motores de bases de datos.

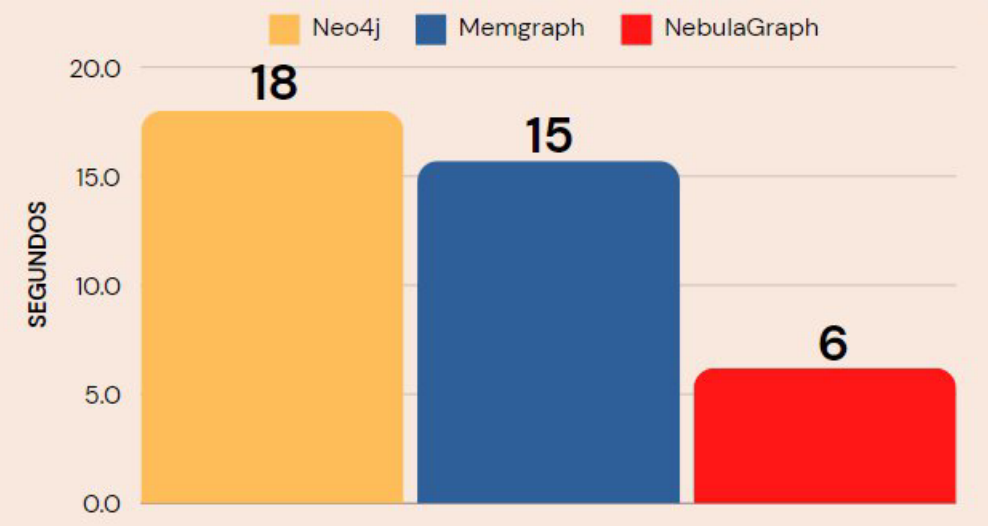

Figura 63. Gráfica comparativa para la ejecución de la consulta en los 3 motores de bases de datos utilizando un solo servidor.

La ejecución de la consulta logra un mejor tiempo en NebulaGraph, ubicando a este motor de base de datos en el primer lugar. Luego Memgraph y finalmente Neo4j.

Existe una gran diferencia entre los tiempos que se obtienen de Memgraph y Neo4j con respecto a NebulaGraph, el cual resulta ser un 300% más rápido respecto a Neo4j y un 250% más rápido que Memgraph.

Por otro lado, el tiempo obtenido por Neo4j es un 20% mayor que el de Memgraph.

5.3.6 Generación del grafo en cada motor de base de datos a partir de archivos CSV - Tres servidores

En esta experimentación, se realizó nuevamente la carga de datos en cada motor de base de datos (Neo4j, Memgraph y NebulaGraph) pero esta vez en un cluster con 3 servidores.

A continuación, en la figura 64, se presenta los tiempos en minutos que necesitó cada uno de los 3 motores de bases de datos para completar la carga de nodos y relaciones.

Base de datos	Juegos	Usuarios	Recomendaciones	Total
Neo4j	1,71 MIN	38,05 MIN	61,56 MIN	100 MIN
Memgraph	2,21 MIN	47,63 MIN	139,91 MIN	183 MIN
NebulaGraph	1,3 MIN	30,15 MIN	111,15 MIN	142 MIN

Figura 64. Tiempos obtenidos en minutos para la carga de datos de set de recomendaciones, con tres servidores

El motor más rápido en importar los 2 archivos CSV que contienen los datos de los nodos del grafo (juegos y usuarios) es NebulaGraph, obteniendo un tiempo de importación de aproximadamente 1 minuto en el caso de los juegos, representando una ventaja por sobre la importación de los mismos datos en Neo4j y sobre todo Memgraph, que tardó más de 2 minutos. Asimismo, presenta un tiempo de aproximadamente 30 minutos en importar los usuarios, de nuevo, una ventaja por sobre Neo4j y Memgraph, tardando este último casi 50 minutos.

En el caso de la importación de recomendaciones, el más rápido resultó ser Neo4j, con un tiempo aproximado de 61 minutos, significativamente menos que los otros dos, siendo Memgraph el más lento con casi 140 minutos, más del doble que Neo4j. Esto indica que, aunque Neo4j resultó segundo en la importación de nodos, a la hora de importar relaciones es más efectivo que los otros dos motores.

En la siguiente figura 65, se observa el gráfico comparativo de tiempos para Neo4j, con tres servidores, en donde se muestran los tiempos de cada tipo de dato por separado, para contrastar la diferencia entre los tiempos de nodos y de relaciones.

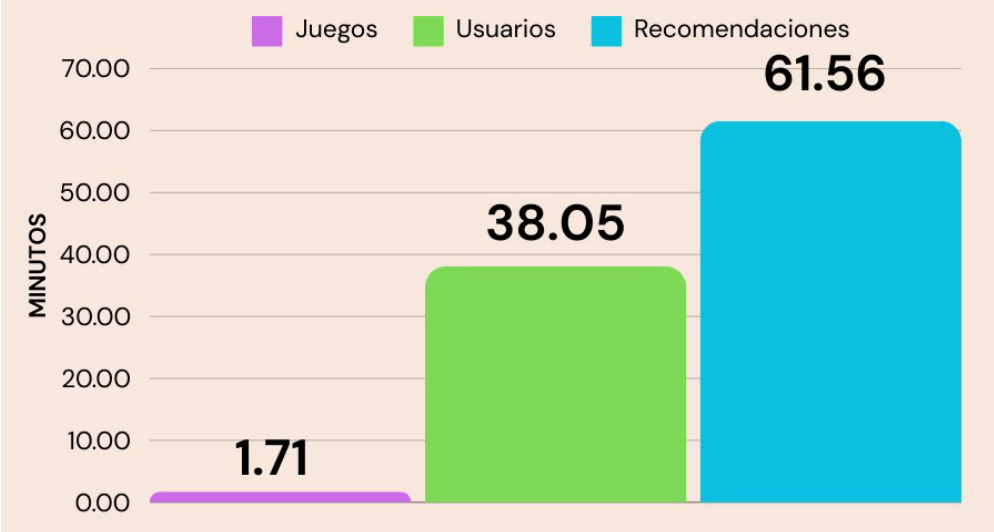

Figura 65. Tiempos obtenidos por Neo4j para la generación del grafo en un clúster con 3 servidores.

Para la generación de los nodos del grafo, Neo4j obtuvo tiempos similares a la ejecución con una sola máquina. No obstante, para la carga de las relaciones entre los nodos, el tiempo obtenido por este motor de base de datos presenta un incremento de aproximadamente unos 18 minutos.

En la siguiente figura 66, de forma similar, se muestran los resultados comparativos de los tres tipos de datos para el motor Memgraph con tres servidores.

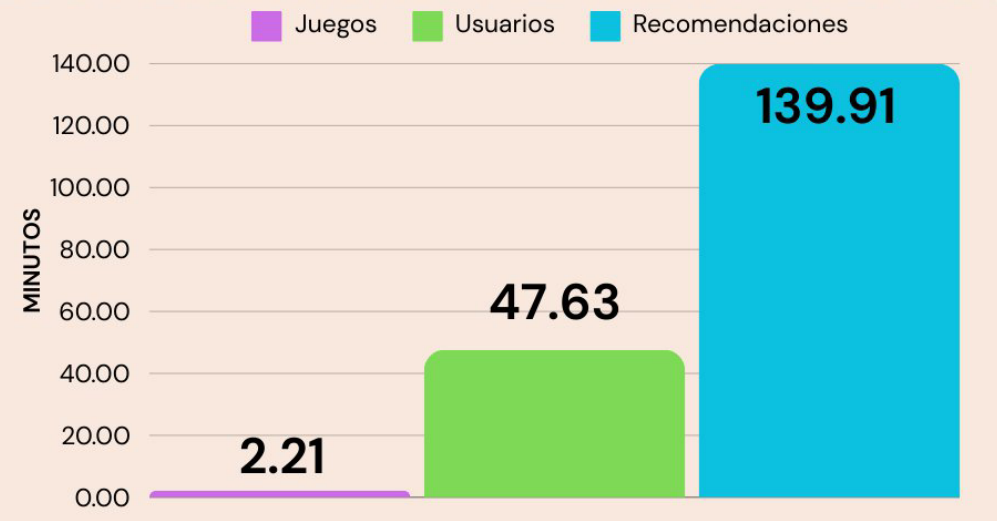

Figura 66. Tiempos obtenidos por Memgraph para la generación del grafo en un clúster con 3 servidores.

Para la generación de los nodos del grafo, Memgraph obtuvo tiempos que representan el doble que el tiempo obtenido en una sola máquina. Además, para la carga de las relaciones entre los nodos, el tiempo obtenido por este motor de base de datos presenta un incremento de aproximadamente unos 6 minutos.

Por último, en la figura 67 se presentan los tiempos para el motor NebulaGraph con tres servidores.

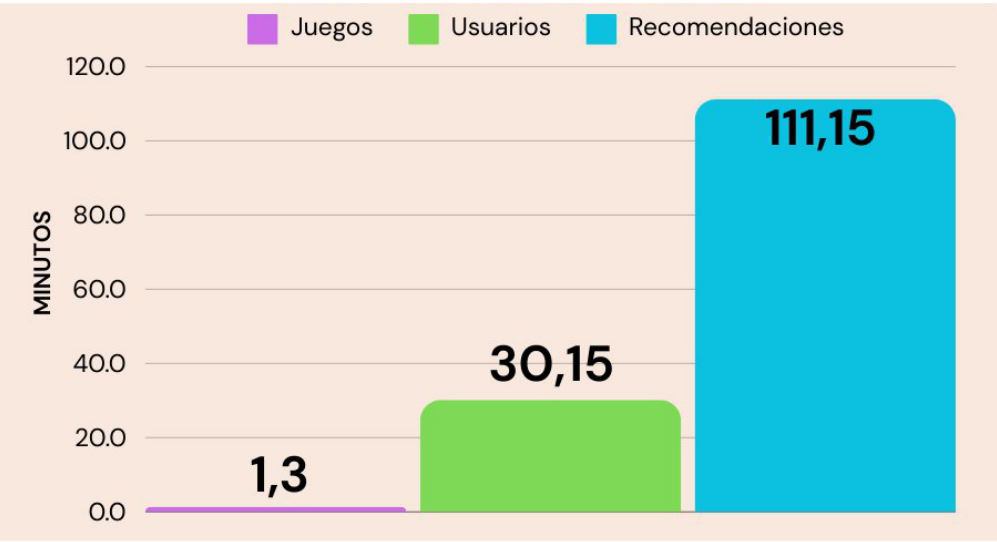

Figura 67. Tiempos obtenidos por NebulaGraph para la generación del grafo en un clúster con 3 servidores.

En el caso de NebulaGraph, los tiempos obtenidos para la carga de los nodos que representan juegos es similar al obtenido en una sola máquina, el tiempo obtenido para la carga de los nodos que representan usuarios es 2 veces mayor al obtenido en una sola máquina y el tiempo obtenido para la generación de las relaciones entre los nodos es un minuto más lento que utilizando una sola máquina.

5.3.7 Medianas - Juegos – Tres servidores

A continuación, en la figura 68, se presentan los tiempos obtenidos por el motor de base de datos Neo4j para la carga en el grafo de los nodos que representan juegos en un cluster configurado con 3 servidores.

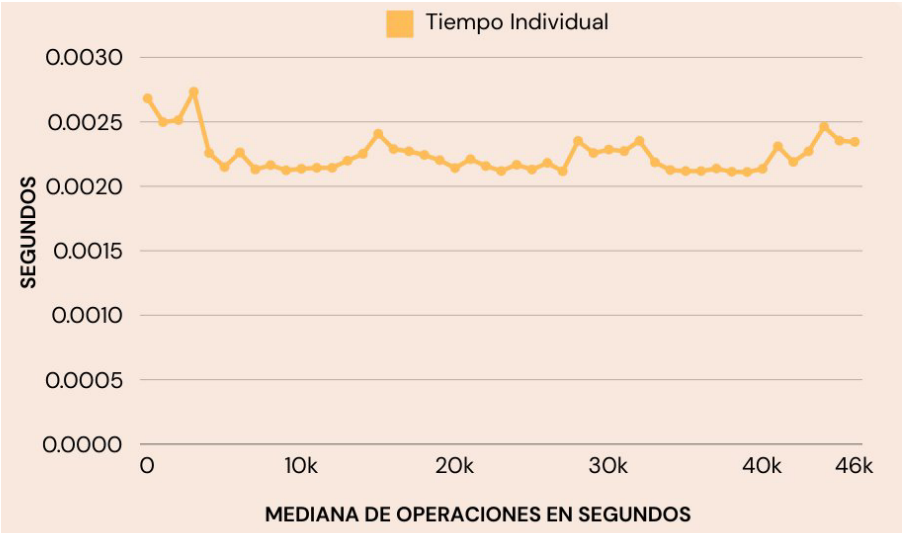

Figura 68. Evolución de Neo4j en un clúster con 3 servidores para la carga de los nodos que representan juegos.

En la gráfica presentada, los tiempos oscilan entre los 0,002 y 0,0025 segundos aproximadamente. En este caso, no presenta picos de valores a destacar.

A continuación, en la figura 69, se presentan los tiempos obtenidos por el motor de base de datos Memgraph para la carga en el grafo de los nodos que representan juegos en un cluster configurado con 3 servidores .

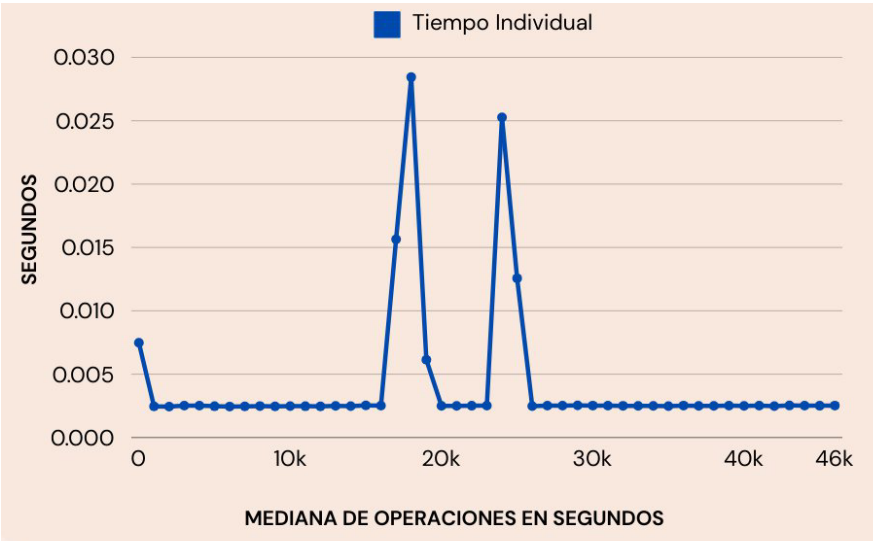

Figura 69. Evolución de Memgraph en un clúster con 3 servidores para la carga de los nodos que representan juegos.

En la gráfica presentada, se destacan dos saltos que llegan a los 0,03 y 0,025 segundos. El resto de la ejecución se mantiene estable en aproximadamente 0,0025 segundos.

A continuación, en la figura 70, se presentan los tiempos obtenidos por el motor de base de datos NebulaGraph para la carga en el grafo de los nodos que representan juegos en un cluster configurado con 3 servidores.

Figura 70. Evolución de NebulaGraph en un clúster con 3 servidores para la carga de los nodos que representan juegos.

En la gráfica presentada, los tiempos oscilan entre los 0,0015 y 0,002 segundos. En este caso, no presenta picos de valores a destacar.

A continuación, en la figura 71, se presenta una comparativa con los tiempos obtenidos por los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph) para la carga en el grafo de los nodos que representan juegos en un cluster configurado con 3 servidores.

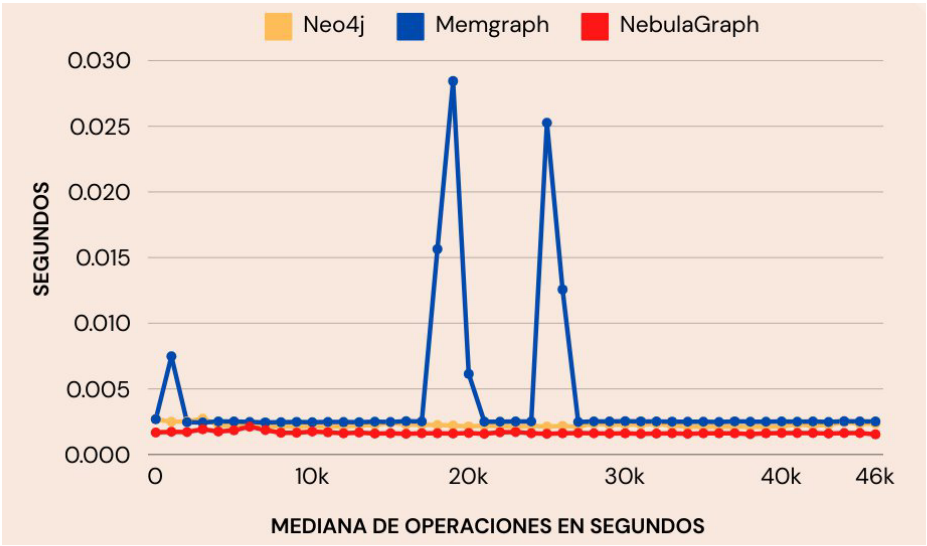

Figura 71. Evolución de la carga de los nodos que representan juegos para Neo4j, Memgraph y NebulaGraph en un clúster con 3 servidores.

Memgraph es el motor de base de datos que presenta algunos picos de tiempos elevados con respecto a los otros 2 motores de bases de datos. No obstante, los 3 motores de bases de datos, poseen un comportamiento similar.

A continuación, en la figura 72, se presenta una comparativa con los tiempos totales obtenidos por los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph) para la carga en el grafo de los nodos que representan juegos en un cluster configurado con 3 servidores.

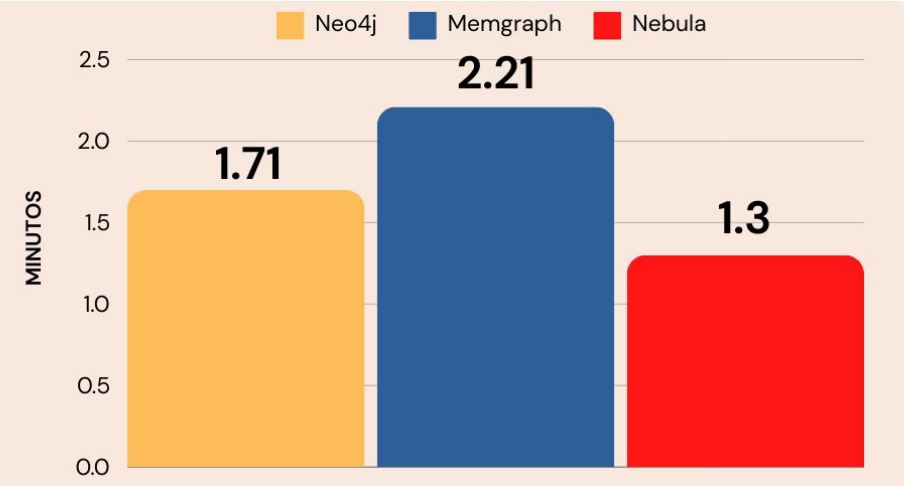

Figura 72. Tiempos totales para los 3 motores de base datos (Neo4j, Memgraph y NebulaGraph) para la generación de los nodos que representan juegos en el grafo en un cluster con 3 servidores.

5.3.8 Medianas - Usuarios - Tres servidores

A continuación, en la figura 73, se presentan los tiempos obtenidos por el motor de base de datos Neo4j para la carga en el grafo de los nodos que representan usuarios en un clúster configurado con 3 servidores.

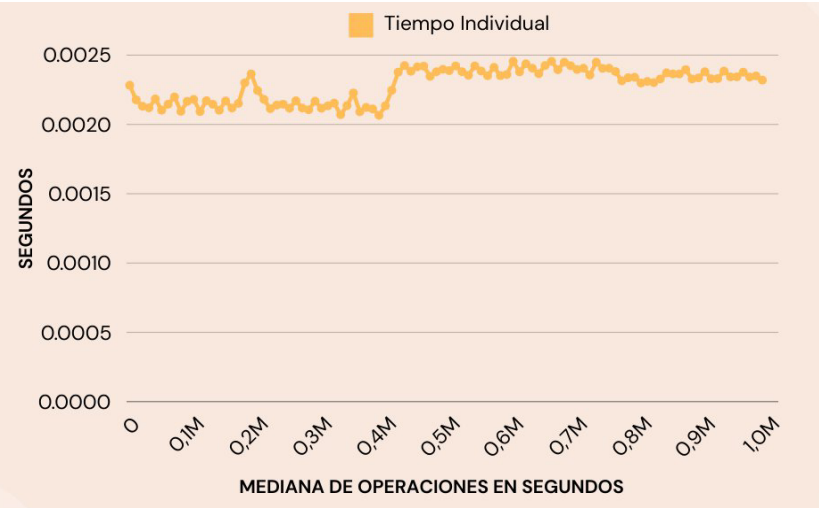

Figura 73. Evolución de Neo4j para la carga de los nodos que representan usuarios en un clúster con 3 servidores.

En este caso, a diferencia de los otros 2 motores de bases de datos, se observa un comportamiento menos regular y que oscila entre los 0,002 y los 0,0025 segundos.

A continuación, en la figura 74, se presentan los tiempos obtenidos por el motor de base de datos Memgraph para la carga en el grafo de los nodos que representan usuarios en un clúster configurado con 3 servidores.

Figura 74. Evolución de Memgraph para la carga de los nodos que representan usuarios en un clúster con 3 servidores.

A continuación, en la figura 75, se presentan los tiempos obtenidos por el motor de base de datos NebulaGraph para la carga en el grafo de los nodos que representan usuarios en un clúster configurado con 3 servidores.

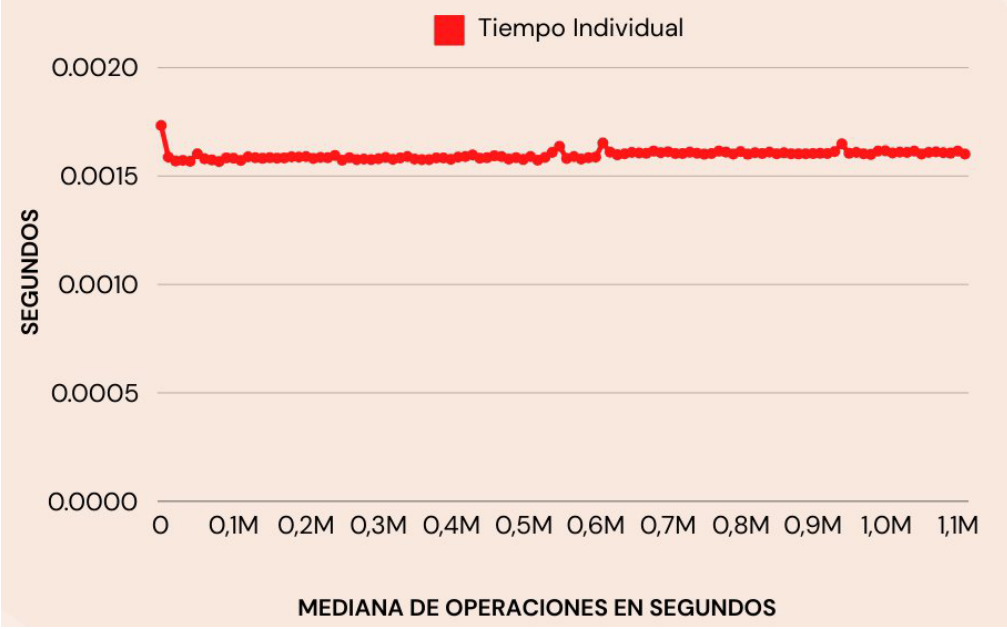

Figura 75. Evolución de NebulaGraph para la carga de los nodos que representan usuarios en un clúster con 3 servidores.

A continuación, en la figura 76, se presenta una comparativa con los tiempos obtenidos por los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph) para la carga en el grafo de los nodos que representan usuarios en un cluster configurado con 3 servidores.

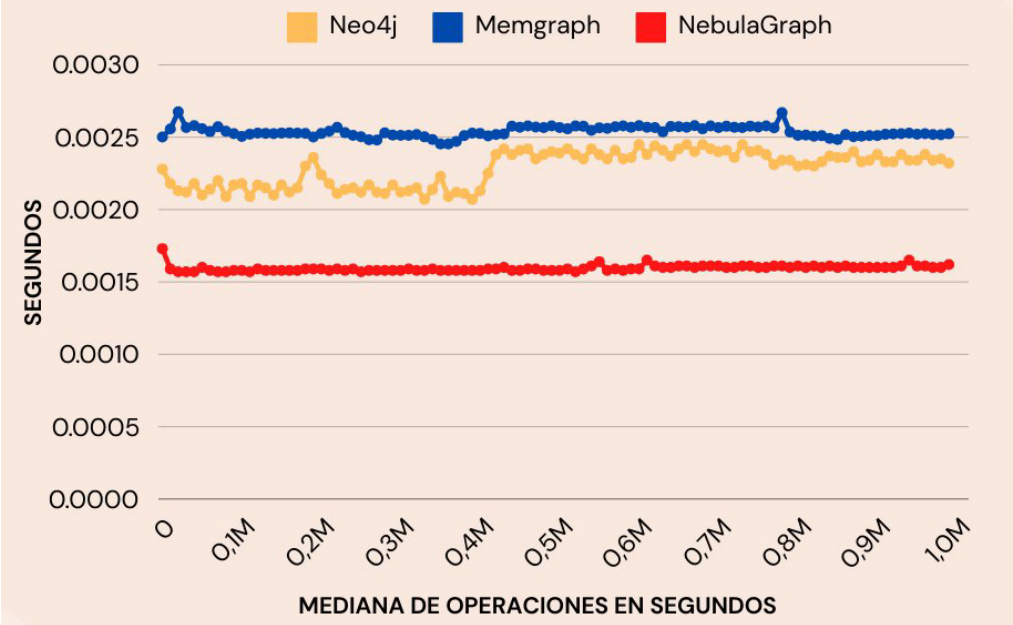

Figura 76. Evolución de la carga de los nodos que representan usuarios en los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph) en un cluster con 3 servidores.

En este caso, se puede observar que es general se mantiene un comportamiento regular para los 3 motores de bases de datos. No obstante, Neo4j es el único motor de base de datos que presenta algunas variaciones en sus tiempos.

Finalmente, se comparan los tiempos que demoró cada uno en completar la carga de datos en la figura 77.

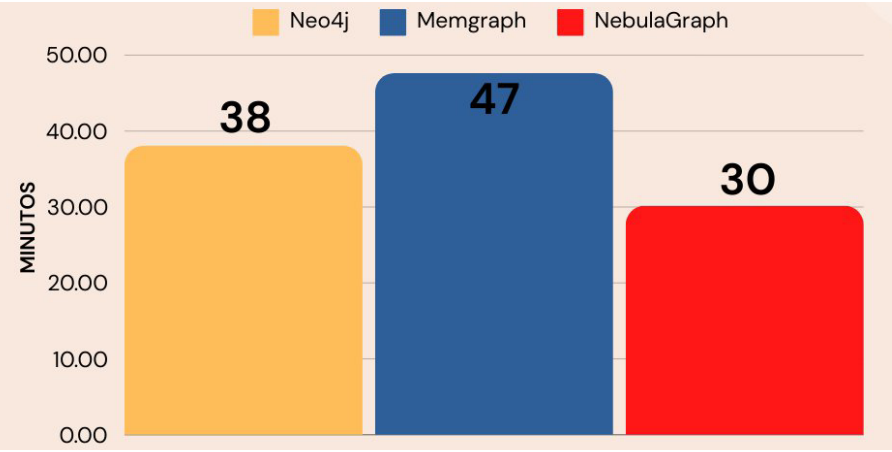

Figura 77. Tiempos totales para los 3 motores de base datos (Neo4j, Memgraph y NebulaGraph) para la generación de los nodos que representan usuarios en el grafo en un clúster con 3 servidores.

Podemos notar que al igual que en el caso anterior, todos tienen valores parecidos, es decir que no hay una gran diferencia en cuanto a tiempos por parte de ninguno.

5.3.9 Medianas - Recomendaciones - Tres servidores

A continuación, en la figura 78 la evolución de tiempos para Neo4j al importar recomendaciones con tres servidores.

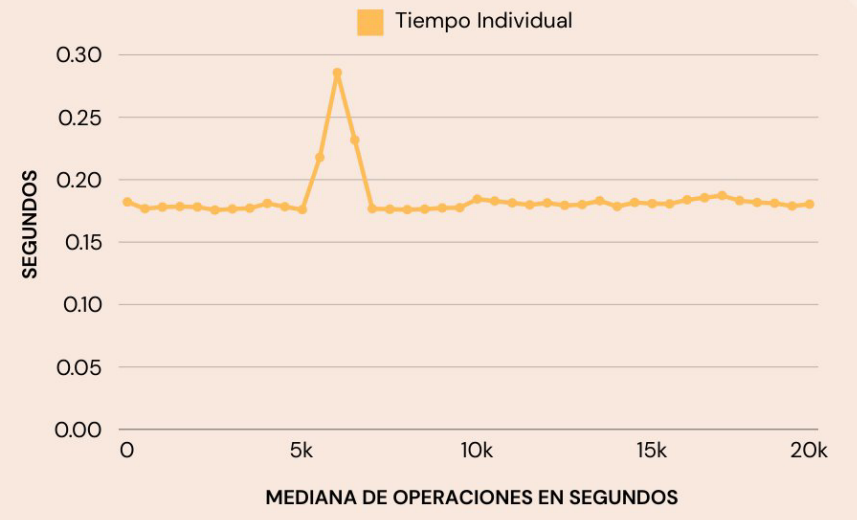

Figura 78. Evolución de Neo4j para la carga de las relaciones entre los nodos del grafo en un cluster con 3 servidores.

Neo4j es el único motor que presenta un pico en la performance, llegando a demorar casi 0,3 segundos para luego volver a estabilizarse por debajo de los 0,2, presentando el menor tiempo promedio de los tres motores.

A continuación en la figura 79 se presenta la evolución del tiempo al importar recomendaciones con tres servidores en Memgraph.

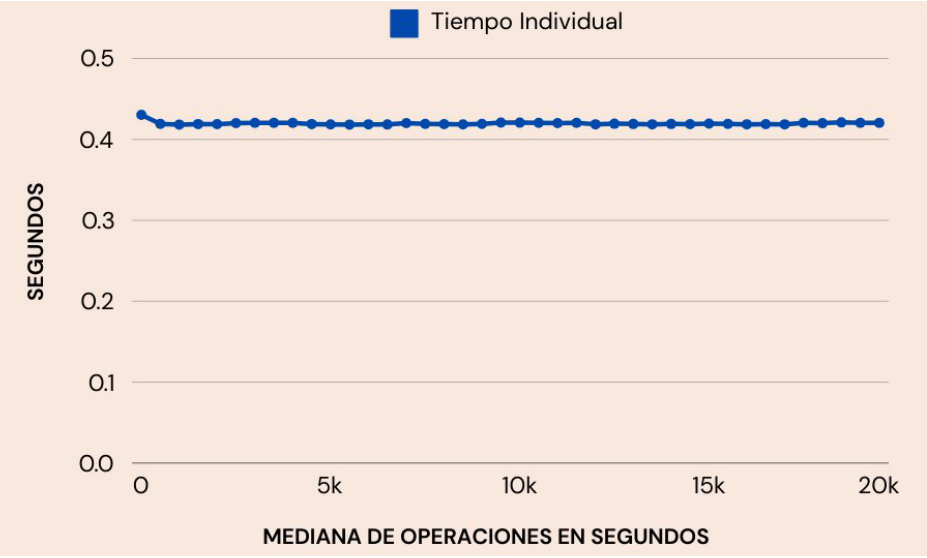

Figura 79. Evolución de Memgraph para la carga de las relaciones entre los nodos del grafo en un clúster con 3 servidores.

Se ven tiempos muy constantes, sin presentar casi ninguna variación a lo largo de la importación, manteniéndose ligeramente por sobre los 0,4 segundos.

En la siguiente figura 80, se presentan los tiempos obtenidos por NebulaGraph al importar recomendaciones con tres servidores.

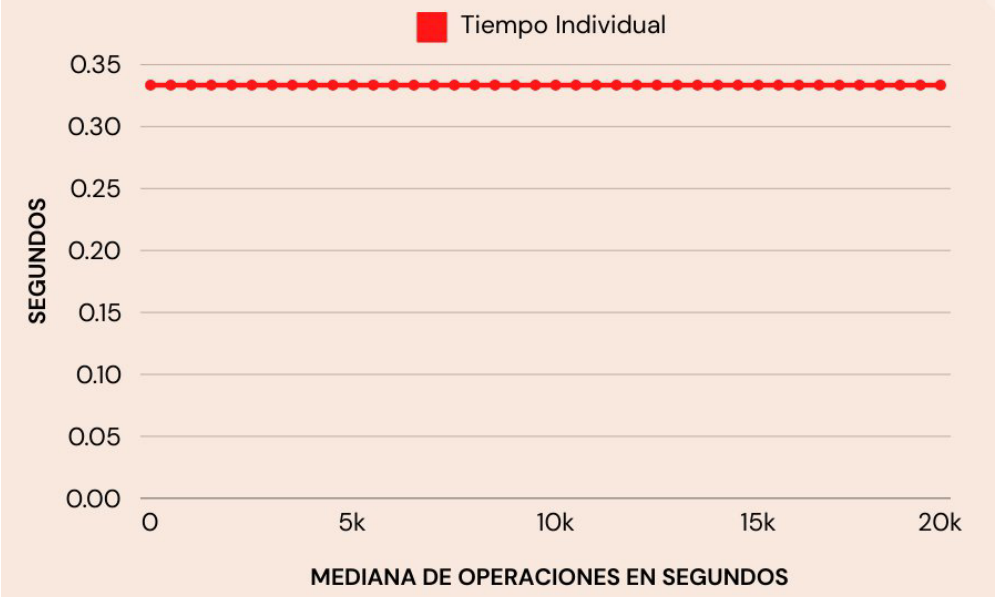

Figura 80. Evolución de NebulaGraph para la carga de las relaciones entre los nodos del grafo en un clúster con 3 servidores.

Al igual que con Memgraph, no se observan variaciones casi en los tiempos, manteniéndose ligeramente por sobre los 0,3 segundos, una diferencia de 0,1 segundos menos que Memgraph.

A continuación, en la figura 81, se presenta una comparativa con los tiempos obtenidos por los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph) para la carga en el grafo de las relaciones que representan las recomendaciones, en un cluster configurado con 3 servidores.

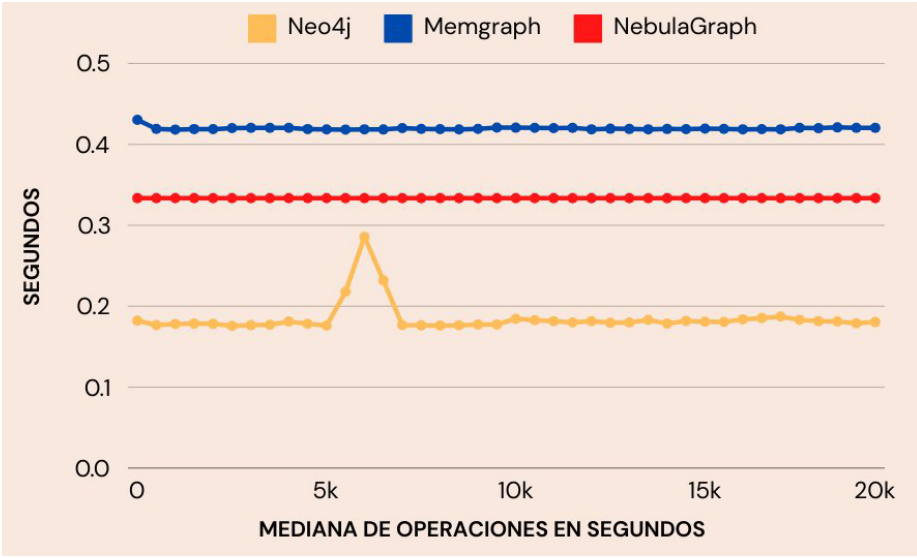

Figura 81. Evolución de la carga de los nodos que representan recomendaciones en los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph) en un cluster con 3 servidores.

NebulaGraph y Memgraph mantienen un comportamiento lineal, este comportamiento es el ideal ya que nos permitiría intuir que se mantendrá si se continúa

haciendo las consultas, a diferencia de Neo4j que su comportamiento nos da la pauta que se debe esperar otro salto en algún momento de la ejecución.

Finalmente en el gráfico 82 se aprecian los tiempos totales de los tres motores de forma comparativa.

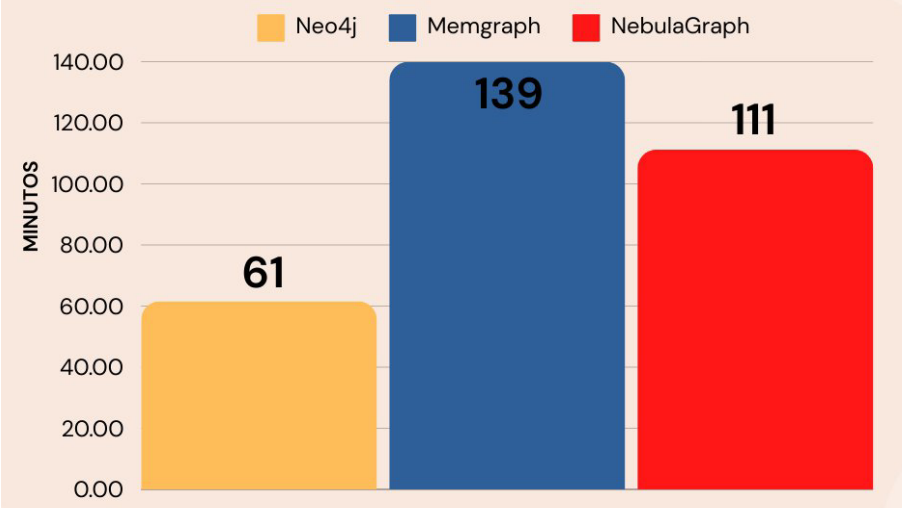

Figura 82. Comparación de tiempos (totales) de recomendaciones para NebulaGraph, Memgraph y Neo4j, en un cluster con 3 servidores

La significativa diferencia que tuvo Neo4j por el resto de los otros motores, es consistente con la ventaja que obtuvo al ejecutar el caso de estudio en un solo servidor, siendo el mejor performante a la hora de importar relaciones.

5.3.10 Ejecución de una consulta implementada en Cypher en los 3 motores de bases de datos - Tres servidores

A continuación, en la figura 83, se presenta la consulta desarrollada en Cypher para ejecutar en los 3 motores de bases de datos (Neo4j, Memgraph y NebulaGraph) en un clúster configurado con 3 servidores.

Esta consulta retorna los juegos que fueron recomendados por más de un usuario distinto.

```
MATCH (u1:User)-[:RECOMMENDS]->(g:Game)<-[:RECOMMENDS]-(u2:User)
WHERE u1.user_id <> u2.user_id
RETURN u1.user_id, u2.user_id, g.title
```

Figura 83. Consulta en Cypher utilizada para medir tiempos en cada motor de base de datos.

A continuación, en la figura 84, se presentan los tiempos expresados en segundos para la ejecución de la consulta en cada uno de los 3 motores de bases de datos, con tres servidores.

Base de datos	Tiempo
Neo4j	79 S
Memgraph	57,75 S
NebulaGraph	30,07 S

Figura 84. Tiempos en segundos para la ejecución de la consulta en cada motor de base de datos en un cluster configurado con 3 servidores.

En este caso, NebulaGraph obtuvo el mejor tiempo. Luego sigue Memgraph, demorando aproximadamente el doble que NebulaGraph y Neo4j obtuvo el peor tiempo de respuesta, siendo un 263% más lento que NebulaGraph.

A continuación, en la figura 85, se presenta una gráfica de los resultados obtenidos para la ejecución de la consulta en los 3 motores de bases de datos en el clúster configurado con 3 servidores.

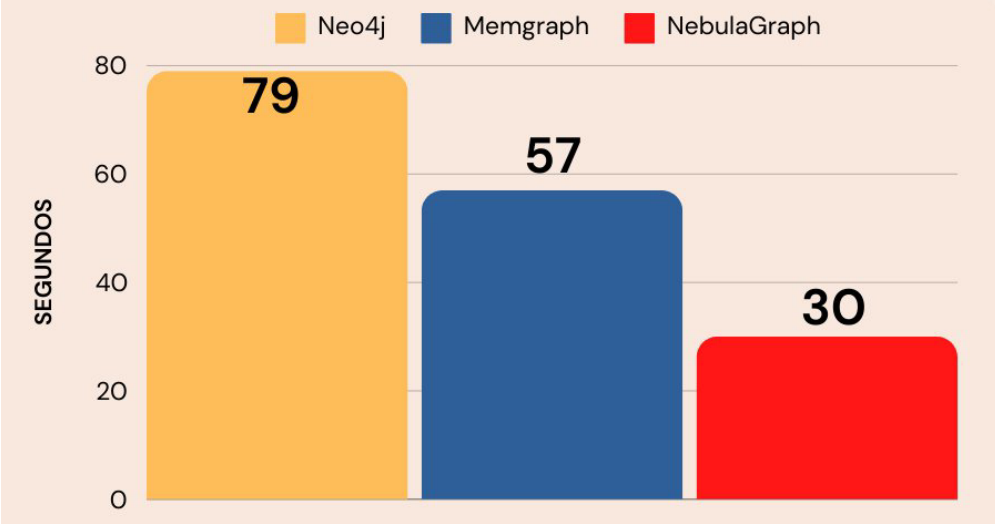

Figura 85. Gráfica comparativa para la ejecución de la consulta en los 3 motores de bases de datos utilizando un clúster con 3 servidores.

Estos resultados son consistentes con los obtenidos al ejecutar la consulta utilizando un solo servidor, en donde se obtienen el mismo orden de tiempos entre los tres motores de bases de datos. Sin embargo, se destaca que el tiempo de respuesta se degradó en relación a la ejecución de la consulta en un solo servidor. Esto se puede deber a que, al tener tres servidores, la consulta tiene que propagarse a los dos servidores réplica en lugar de sólo en el principal.

5.3.11 Análisis comparativo del caso de estudio 2

A continuación, en la figura 86 se presenta una comparativa en los tiempos de carga totales para la configuración con un servidor y 3 servidores.

Base de datos	Un servidor	Tres servidores
Neo4j	81 MIN	100 MIN
Memgraph	161 MIN	183 MIN
NebulaGraph	129 MIN	142 MIN

Figura 86. Comparación de los tiempos de carga para los 3 motores de bases de datos con un sólo servidor

En este caso se puede observar la diferencia de Neo4j con respecto de los otros 2 motores de bases de datos. En la sección 5.3.1, es más del doble respecto de Memgraph y de un 50% con NebulaGraph. En la sección 5.3.6 continúa la tendencia pero se reduce para ambos motores

Además, los 3 motores de bases de datos, bajan su performance cuando se configuran para un cluster de 3 servidores. Aunque no es drástico.

A continuación, en la figura 87 se presenta una comparación de los tiempos de consulta propia del dominio para los 3 motores de bases de datos con un servidor y tres servidores

Base de datos	Un servidor	Tres servidores
Neo4j	18,1 S	79 S
Memgraph	15,7 S	57,75 S
NebulaGraph	6,2 S	30,07 S

Figura 87. Comparación de los tiempos de consulta propia del dominio para los 3 motores de bases de datos con un servidor y tres servidores

Si bien todos los motores sufrieron un incremento de sus tiempos, el más perjudicado es NebulaGraph el cual aumenta en un 485% su demora y en un segundo lugar Neo4j con un 438%.

A continuación, en la figura 88, se presenta un gráfico comparativo, en donde se visualizan tanto los tres motores, como el tiempo que obtuvo cada uno con la configuración de uno y tres servidores.

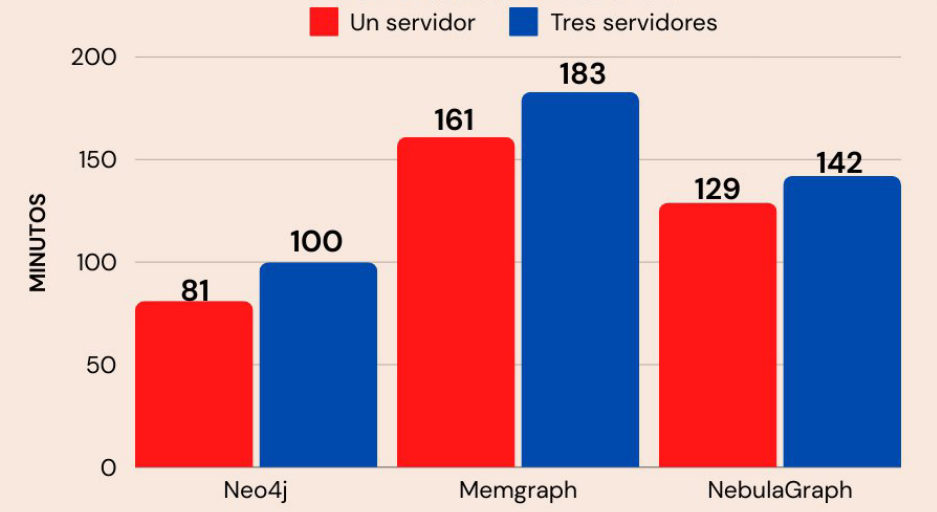

Figura 88. Comparación de los tiempos de carga para los 3 motores de bases de datos con un servidor y tres servidores

Se puede observar que los tres motores vieron una degradación de tiempos al escalar de uno a tres servidores en la ejecución, siendo la diferencia proporcional más grande la sufrida por Neo4j, la cual constituye un 123% al pasar a tres servidores. También es notable la poca diferencia que existen entre los tres ya que en Memgraph y NebulaGraph fue de 113% y 110% respectivamente.

En la siguiente figura 89, se puede observar de forma comparativa los resultados obtenidos por el motor Neo4j, tanto en el primer caso de estudio (centrado en concurrencia) como en el segundo (secuencial).

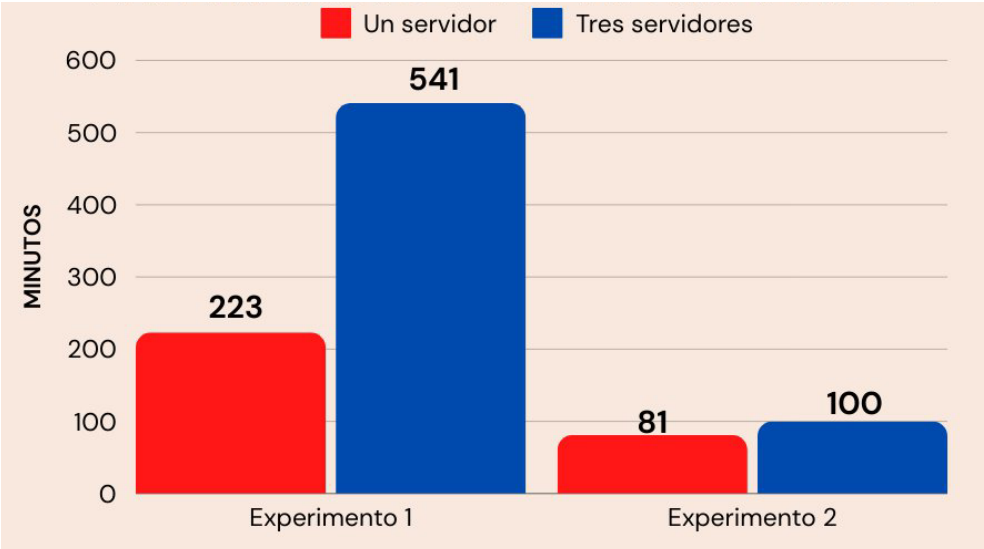

Figura 89. Resultados comparativos para Neo4j de la ejecución del caso de estudio 1 y 2

En la figura 90, de la misma forma, se observa los resultados comparativos para el caso de estudio 1 y 2 para el motor Memgraph

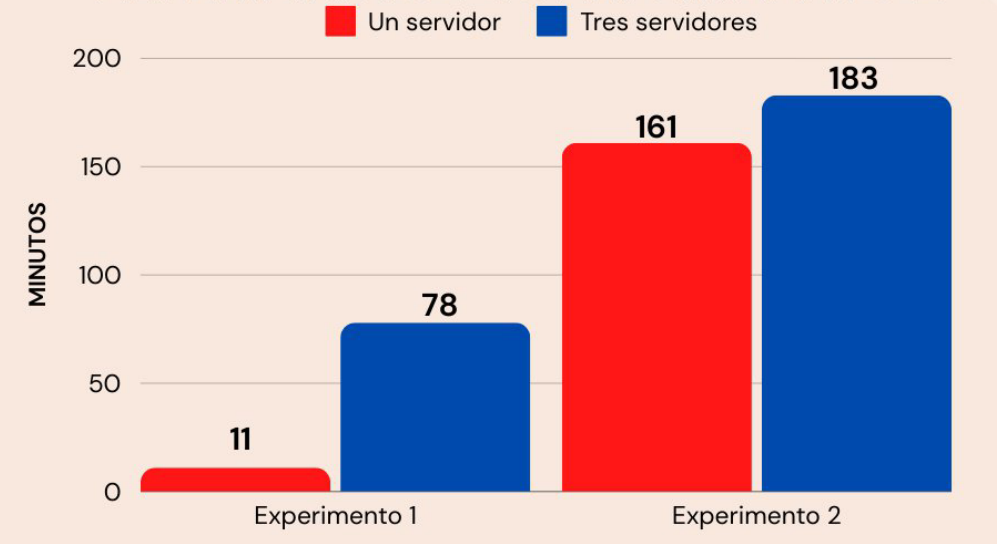

Figura 90. Resultados comparativos para Memgraph de la ejecución del caso de estudio 1 y 2

Finalmente, en la figura 91, se observan los resultados comparativos para el caso de estudio 1 y 2 para el motor NebulaGraph

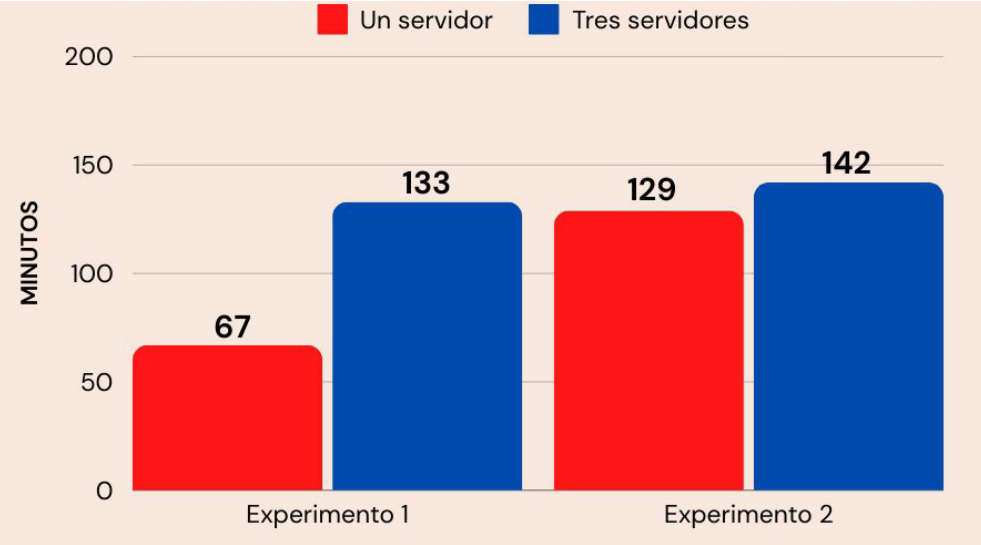

Figura 91. Resultados comparativos para NebulaGraph de la ejecución del caso de estudio 1 y 2

Capítulo 6 (Conclusiones y Trabajo futuro)

6.1 Conclusiones

En este trabajo se han abordado conceptos sobre bases de datos, principalmente bases de datos relacionales y bases de datos no relacionales. En este contexto, se estudian detalladamente las bases de datos no relacionales orientadas a grafos. Posteriormente, se plantean un conjunto de casos de estudio prácticos en 3 bases de datos orientadas a grafos, Neo4j, Memgraph y NebulaGraph. El objetivo principal es realizar una serie de mediciones comparativas de performance entre los 3 motores de bases de datos.

En el primer caso de estudio se realizaron operaciones (lecturas y escrituras) para evaluar el acceso concurrente a los 3 motores de bases de datos. Además, se realizaron pruebas para evaluar cómo impacta escalar horizontalmente en cada uno de los motores de bases de datos estudiados.

En el segundo caso de estudio se realizó una importación masiva de datos y luego se realizó una consulta del dominio de grafos con el objetivo de obtener métricas de performance ante un gran volumen de datos.

En el primer caso de estudio (sección 5.2), el motor que posee mejor performance para las consultas realizadas fue el motor de bases de datos Memgraph, seguido del motor de bases de datos NebulaGraph. Ambos motores de bases de datos, han logrado mejor performance con respecto al motor de bases de datos Neo4j.

Esto se puede deber a que Memgraph utiliza almacenamiento en memoria principal en vez de memoria secundaria (disco rígido). Por lo tanto, intenta cargar la mayor parte del grafo en la memoria principal con el objetivo de agilizar las consultas. Además, cada cierto tiempo se realiza una captura del estado de la base de datos para guardar en memoria secundaria y garantizar las propiedades ACID [103].

Otro motivo que puede justificar la diferencia de performance en las consultas planteadas, es el lenguaje en el que está desarrollado cada uno de los motores de bases de datos utilizados. Ambos motores, Memgraph y NebulaGraph están desarrollados en C++, esta característica brinda efectividad en el manejo a bajo nivel de la memoria, así como mayor compatibilidad con el lenguaje de programación elegido para realizar las pruebas (Python). En el caso de Neo4j, utiliza el lenguaje Java, que es un lenguaje que brinda portabilidad, pero con menor optimización de performance que C y C++ en términos generales.

Además, la estructura de cada nodo involucra pocos atributos, esto es algo que puede contribuir a obtener una mejor performance en alguno de los motores de bases de datos utilizados [112].

Memgraph y NebulaGraph poseen un buen desempeño al realizar un procesamiento en tiempo real [63, 64], esto es algo que no sucede con Neo4j. Por otro

lado, en un entorno de concurrencia NebulaGraph es el único motor de base de datos que no baja su performance.

En este primer caso de estudio, se pueden realizar dos análisis, uno respecto a operaciones solo de escrituras de forma concurrente y el otro respecto a la evaluación de las operaciones de recuperación de datos (lecturas) sobre el grafo generado.

En lo que respecta a las operaciones de escritura de forma concurrente, Neo4j ofrece los mejores tiempos, ya sea instalado y configurado en un solo servidor, como escalando en un cluster de 3 servidores.

En el segundo caso de estudio (sección 5.3), el desempeño de las pruebas realizadas en los 3 motores de bases de datos (Memgraph, NebulaGraph y Neo4j) es similar en un solo servidor que al realizar escalamiento horizontal en un cluster de 3 servidores. Además, en los 3 motores de bases de datos se presenta una baja de performance al plantear una consulta cuando se utiliza escalamiento horizontal. No obstante, en el caso de las cargas masivas de manera secuencial, escalar horizontalmente no afecta la performance de las operaciones realizadas.

En este segundo caso de estudio Neo4j presenta un buen desempeño para la carga de datos de forma secuencial. Además, existe una diferencia notoria en minutos entre el mejor tiempo de respuesta obtenido y el peor tiempo de respuesta obtenido. Esto último sucede cuando se utiliza un solo servidor y cuando se escala horizontalmente en un cluster de 3 servidores.

Para el caso de la ejecución de una consulta propia del dominio, Neo4j ha obtenido el tiempo más alto entre los 3 motores de bases de datos seleccionados. En resumen, Neo4j posee un buen desempeño en escenarios donde la prioridad no es tener una respuesta inmediata. En el caso de Memgraph, se obtiene una performance aceptable para ambientes concurrentes, es decir, evaluando múltiples consultas en simultáneo.

6.2 Trabajo Futuro

Como trabajo a futuro se propone ampliar este trabajo incorporando nuevas configuraciones para escalar horizontalmente, como puede ser aumentar la cantidad de servidores del cluster y/o aplicar diferentes combinaciones para la sincronización de los servidores del cluster.

Además, se pretende realizar pruebas similares para otros tipos de motores de base de datos NoSQL, como pueden ser bases de datos NoSQL documentales, bases de datos orientadas a familia de columnas, bases de datos clave-valor, entre otras.

También, se pretende expandir los casos de estudio para incluir nuevos motores de bases de datos orientados a grafos y probar nuevas consultas específicas para motores de bases de datos NoSQL orientados a grafos.

Bibliografía

[1] Date, C. J. *An Introduction to Database Systems*. Pearson/Addison Wesley, 2003.

[2] Auer, David, et al. *Database Concepts*. Pearson, 2017.

[3] Elmasri, R., et al. Fundamentals of Database Systems. Addison-Wesley, 2019

[4] Abiteboul, S., Quass, D., McHugh, J. *et al.* The Lorel query language for semistructured data. *Int J Digit Libr* 1, 68–88 (1997). https://doi.org/10.1007/s007990050005

[5] Olle, T. William. *The Codasyl approach to database management*. Wiley, 1978.

[6] https://www.ibm.com/products/ims

[6] Scheuermann, Peter, ed. 2000. *Proceedings, 1999 Workshop on Knowledge and Data Engineering Exchange (KDEX '99): November 7, 1999, Chicago, Illinois*. N.p.: IEEE Computer Society.

[7] Serge Abiteboul and Richard Hull. 1988. Restructuring hierarchical database objects. Theor. Comput. Sci. 62, 1–2 (December 1988), 3–38. https://doi.org/10.1016/0304-3975(86)90010-1

[8] Taylor, Allen G. *Database Development For Dummies*. Wiley, 2000.

[9] IBM Products. https://www.ibm.com/products/ims
[10] RDM Database. Overview of Raima Database Manager (RDM) Solutions & Architecture
[11] HPL. http://www.hpl.hp.com/hpjournal/pdfs/IssuePDFs/1986-12.pdf
[12]C. W. Bachman, "The Origin of the Integrated Data Store (IDS): The First Direct-Access DBMS," in IEEE Annals of the History of Computing, vol. 31, no. 4, pp. 42-54, Oct.-Dec. 2009, doi: 10.1109/MAHC.2009.110.
[13]RDB.oracle.com/technetwork/database/database-technologies/rdb/dbms-131998.pdf
[14] E. F. Codd. 1970. A relational model of data for large shared data banks. Commun. ACM 13, 6 (June 1970), 377–387. https://doi.org/10.1145/362384.362685

[15] Tomasz Imielinski and Witold Lipski. 1982. A systematic approach to relational database theory. In Proceedings of the 1982 ACM SIGMOD international conference on Management of data (SIGMOD '82). Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/582353.582356

[16] PostgreSQL. PostgreSQL

[17] MySQL. MySQL

[18] MariaDB. MariaDB Foundation

[19] OracleDB. Database | Oracle

[20] ODBMS. ODBMS.org

[21] Kim, Won. *Introduction to Object-oriented Databases*. MIT Press, 1990.

[22] Delphi. Delphi: IDE Software Overview - Embarcadero

[23] Ruby. Ruby

[24] Java. Java Software | Oracle

[25] Perst. Perst Open Source Object Oriented Database System - McObject LLC

[26] Object-store. Object Store v2 Overview | MuleSoft Documentation

[27] ObjectDB. ObjectDB

[28] DB-Engines. DB-Engines Ranking - popularity ranking of database management systems

[29] MongoDB. MongoDB

[30] Redis. https://redis.io/docs/

[31] Neo4j. Neo4j.

[32] Cassandra. Apache Cassandra

[33] Uses of Neo4j. Graph Databases Uses by Neo4j Customers

[34] Case studies for Redis. Redis Reviews & Customer Case Studies

[35] Customers, MongoDB. Our Customers | MongoDB

[36] SQL Language. Structured Query Language (SQL)

[37] ANSI. American National Standards Institute

[38] ISO. ISO

[39] ISO 9075. ISO/IEC 9075-1:2023 - Information technology — Database languages SQL

[40] Chamberlin, D. (2009). SQL. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_1091],[Chatham, Mark (2012). *Structured Query Language By Example - Volume I: Data Query Language*

[41] What is NoSQL? What Is NoSQL? NoSQL Databases Explained | MongoDB

[42] Google Big Table. Bigtable Documentation | Google Cloud

[43] Santhosh Kumar Gajendran, A Survey on NoSQL Databases, 2017

[44] CAP Theorem. What Is the CAP Theorem? | IBM

[45] What is Cloud Storage? What is Cloud Storage & How Does it Work? | Google Cloud

[46] Schütt, T., Schintke, F., & Reinefeld, A. (2008, septiembre). Scalaris: Reliable transactional p2p key/value store. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG* (pp. 41-48).

[47] Memcached. Memcached

[48] ETCD. etcd

[49] Ehcache. Ehcache Documentation

[50] DynamoDB.Amazon DynamoDB - AWS Mobile SDK

[51] JSON. JSON

[52] XML. XML introduction - XML: Extensible Markup Language | MDN

[53] CouchDB. Apache CouchDB

[54] Abadi, D. J., Boncz, P. A., & Harizopoulos, S. (2009). Column-oriented database systems. Proceedings of the VLDB Endowment, 2(2), 1664-1665

[55] Apache Hbase. Apache HBase

[56] Paredaens et al. (1995). G-Log: A graph-based query language. IEEE Transactions on Knowledge and Data Engineering, 7(3), 436-453.

[57] Kurant, M., Markopoulou, A., & Thiran, P. (2010, September). On the bias of BFS (breadth first search). In 2010 22nd International Teletraffic Congress (LTC 22) (pp. 1-8). IEEE.

[58] Depth First Search. 12.3 Graph Traversal

[59] Single-Source Shortest Path. Undirected single-source shortest paths with positive integer weights in linear time | Journal of the ACM

[60] All-Pairs Shortest Path. Floyd Warshall Algorithm - GeeksforGeeks

[61] Degree Centrality. Degree Centrality

[62] OrientDB. OrientDB.

[63] Memgraph. Memgraph

[64] NebulaGraph. Nebula Graph

[65] Elastics search. Elastic documentation | Elastic

[66] Microsoft Azure Search.Introduction to Azure AI Search - Azure AI Search | Microsoft Learn

[67] Amazon Cloud Search. What Is Amazon CloudSearch? - Amazon CloudSearch

[68] Splunk. What Is Splunk & What Does It Do? A Splunk Intro | Splunk

[69] Use Cases – Elastic Search. Use Cases - ElasticSearch Customers

[70] Vector databases. Vector Database | Microsoft Learn

[71] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[72] Vector Database (AWS). What is a Vector Database?

[73] Vector Database (IBM) What Is a Vector Database? | IBM

[74] Qdrant. Qdrant

[75] Chroma. Chroma

[76] Pinecone. https://www.pinecone.io

[77] Keogh, E., Lonardi, S., & Chiu, B. Y. C. (2002, July). Finding surprising patterns in a time series database in linear time and space. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 550-556).

[78] Chen Wang, Xiandong Huang, Jialin Qiao, et al. Apache IoTDB: time-series database for internet of things. https://dl.acm.org/doi/10.14778/3415478.3415504

[79] Prometheus. Prometheus

[80] Graphite. Overview — Graphite 1.1.10 documentation

[81] Cloud Databases: A Paradigm Shift in Databases, Indu Arora et al

[82] Grossman, R. L. (2009). The case for cloud computing. IT professional, 11(2), 23-27.

[83] MongoDB Atlas. MongoDB Atlas Database | Multi-Cloud Database Service

[84] Apache Cassandra Cloud. Apache Cassandra

[85] Aura DB. Fully Managed Graph Database Service | Neo4j AuraDB

[86] Redis Cloud. Redis Cloud – Fully Managed Cloud Service

[87] Microsoft Azure.Microsoft Azure

[88] Amazon Web Service. Amazon AWS

[89] Google Cloud. Google Cloud

[90] Azure Cosmos DB. Azure Cosmos DB - NoSQL and Relational Database

[91] PostgreSQL. PostgreSQL

[92] Agrawal, D., El Abbadi, A., Das, S., & Elmore, A. J. (2011, April). Database scalability, elasticity, and autonomy in the cloud. In *International conference on database systems for advanced applications* (pp. 2-15). Berlin, Heidelberg: Springer Berlin Heidelberg.

[93] Jaroslav Pokorný, NoSQL Databases: a step to database scalability in Web environment, 2011

[94] Cypher. openCypher

[95] Neo4j licensing. Neo4j Licensing

[96] Data consistency models.Data Consistency Models: ACID vs. BASE Explained

[97] 10 Reasons why Neo4j is the best graph database for your project. 10 Reasons Why Neo4j Is the Best Graph Database for Your Project

[98] Neo4j GraphQL. Neo4j GraphQL Library

[99] Neo4j Bloom. Neo4j Bloom

[100] C++. CPlusPlus.com

[101] BSL license. Business Source License (BSL)

[102] Memgraph.Memgraph Storage memory usage

[103] Apache license 2.0. Apache License, Version 2.0

[104] OpenCypher. openCypher

[105] Apache Spark. Apache Spark

[106] NebulaGraph exchange. GitHub - vesoft-inc/nebula-exchange

[107] Docker. Docker

[108] Python. Python.org

[109] GitHub. GitHub Docs

[110] Mediana. Mediana (estadística)

[111] Third time's the charm for NebulaGraph. Third Time Is The Charm For Nebula Graph Database

[112] Memgraph vs NebulaGraph. Memgraph vs NebulaGraph

[113] Graph database performance comparison. Graph Database Performance Comparison: Neo4j vs NebulaGraph vs JanusGraph