UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

KhaBench: Un análisis comparativo de bases de datos multimodelo en ambientes distribuidos

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Edgar Cristóbal García Gutiérrez

DIRECTORA DE TESIS

Dra. María del Pilar Ángeles

Ciudad Universitaria, Cd. Mx., 2026

PROTESTA UNIVERSITARIA DE INTEGRIDAD Y HONESTIDAD ACADÉMICA Y PROFESIONAL
(Titulación con trabajo escrito)

De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado KHABENCH: UN ANALISIS COMPARATIVO DE BASES DE DATOS MULTIMODELO EN AMBIENTES DISTRIBUIDOS que presenté para obtener el titulo de INGENIERO EN COMPUTACIÓN es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

EDGAR CRISTOBAL GARCIA GUTIERREZ
Número de cuenta: 316028099

Índice

Agradecimientos.

A mis padres, cuyo amor, guía y fortaleza han sido el pilar más importante a lo largo de mi vida. Gracias por enseñarme con su ejemplo a nunca rendirme.

A mis amigos Eduardo Moreno y Alberto García, compañeros entrañables desde mis primeros años escolares. Su amistad, apoyo y presencia constante han hecho más ligero y memorable cada paso de este camino académico.

A Celeste Herrera, por acompañarme en la etapa final de esta tesis con paciencia, comprensión y un apoyo invaluable. Su impulso y confianza fueron determinantes para que este proyecto llegara a su conclusión.

Y, de manera muy especial, a la Dra. María del Pilar Ángeles, mi asesora y directora de tesis, por su guía, dedicación y exigencia académica. Su acompañamiento, experiencia y confianza fueron fundamentales para que este trabajo pudiera realizarse.

A todos ellos, mi más profundo reconocimiento y gratitud.

Resumen.

La creciente adopción de los sistemas gestores de bases de datos multimodelo (MMDBMS por sus siglas en inglés) en entornos empresariales plantea una pregunta fundamental para los equipos de desarrollo y arquitectura: ¿qué manejador ofrece el mejor rendimiento cuando los datos se distribuyen entre múltiples nodos? Si bien existen trabajos previos que comparan MMDBMS, estos se han limitado principalmente a entornos centralizados y a medir únicamente el tiempo de ejecución de consultas y transacciones, dejando de lado métricas igualmente relevantes como el uso de CPU, el consumo de memoria RAM y la consistencia observable en escenarios distribuidos. En este contexto, se plantea que el diseño de un *benchmark* específico para entornos distribuidos permitirá identificar diferencias significativas en el rendimiento de cada MMDBMS y orientar así la toma de decisiones sobre su adopción.

Este trabajo presenta KhaBench, un *benchmark* diseñado para evaluar el rendimiento de tres MMDBMS (ArangoDB, OrientDB y Couchbase) en un ambiente distribuido construido sobre nodos Raspberry Pi 5. El *benchmark* se desarrolla en el contexto de una red social de comercio electrónico, y contempla un esquema multimodelo que integra modelos de datos basados en relaciones, documentos, grafos y llave-valor. La metodología se estructura en cuatro fases: análisis y diseño de la base de datos distribuida, implementación y carga de datos con estrategias de fragmentación horizontal, ejecución de siete cargas de trabajo (consultas y transacciones locales, remotas y distribuidas), y análisis de métricas mediante percentiles de latencia (p50, p95 y p99) junto con el monitoreo de recursos y la consistencia tipo RYOW (Read Your Own Write).

Los resultados muestran que no existe un manejador óptimo para todos los escenarios: ArangoDB ofrece el comportamiento más estable y predecible bajo alta concurrencia, OrientDB resulta más adecuado para cargas híbridas de grafo y documento en entornos de baja concurrencia, y Couchbase sobresale en aplicaciones de lectura intensiva con datos frecuentemente almacenados en memoria. Estos hallazgos buscan apoyar a desarrolladores, administradores de bases de datos e investigadores en la toma de decisiones informadas sobre la adopción de tecnologías MMDBMS en entornos distribuidos.

1. Introducción y Estado del Arte.

Los sistemas gestores de bases de datos multimodelo o *Multi Model Database Management System* (MMDBMS por sus siglas en inglés) permiten el almacenamiento, el manejo y el análisis de diferentes modelos de datos en una sola plataforma [1]. Estos sistemas pueden operar con datos estructurados, semiestructurados y no estructurados, por ejemplo: documento, grafo, relaciones, llave-valor, entre otros.

El objetivo de los MMDBMS es proporcionar una solución de almacenamiento y gestión de datos más completa, eficiente y flexible [2] que pueda solucionar las necesidades de diversos sistemas y casos de uso, por ejemplo: aquellos que manejan grandes cantidades de datos, que requieren un alto rendimiento para las consultas y que manejan diferentes modelos de datos, entre otros [2].

En la actualidad, los MMDBMS son utilizados en una amplia variedad de aplicaciones, incluyendo la gestión de contenido web [3], la gestión de datos de redes sociales [3], la analítica empresarial [3], en internet de las cosas (IoT por sus siglas en inglés) [1], entre otras.

Algunos de los trabajos de comparación o *benchmarks* (término que en español puede traducirse aproximadamente como evaluación comparativa de rendimiento) de los MMDBMS se enfocan en medir el rendimiento, la escalabilidad y la eficiencia de estas bases de datos en diferentes escenarios; por ejemplo: M2Bench [4], UniBench [5], entre otros [6].

Actualmente algunos de los trabajos de comparación de MMDBMS [7], [8] se centran en analizar cómo se fusionan múltiples fuentes de datos en una sola instancia de una base de datos que permitan homogeneizar la información de manera efectiva, en examinar técnicas para realizar consultas más eficientes y precisas además de estudiar técnicas de análisis de datos [7], [8].

1.1. Hipótesis.

Si se diseña e implementa un *benchmark* que evalúe sistemas gestores de bases de datos multimodelo en un ambiente distribuido, considerando métricas de latencia, uso de CPU, memoria RAM y consistencia observable, entonces es posible identificar diferencias significativas en el rendimiento de cada sistema que permitan orientar la toma de decisiones sobre su adopción en entornos distribuidos reales.

1.2. Objetivos.

a) Objetivo General

Crear un marco de evaluación comparativo que permita medir el rendimiento de sistemas gestores de bases de datos multimodelo al implementar estrategias de fragmentación de datos que se distribuyan a lo largo de los nodos en un sistema de bases de datos multimodelo distribuido, con el fin de identificar diferencias en rendimiento, uso de recursos y consistencia entre los distintos sistemas evaluados, y que pueda aplicarse a cualquier MMDBMS arbitrario.

b) Objetivos específicos

- Proponer un esquema de base de datos distribuida multimodelo que integre diferentes modelos de datos mediante estrategias de fragmentación horizontal.
- Establecer un conjunto de métricas de evaluación que contemple latencia, uso de CPU, memoria RAM y consistencia observable en entornos distribuidos.
- Definir un conjunto de cargas de trabajo representativas que cubran operaciones locales, remotas y distribuidas sobre los MMDBMS evaluados.
- Validar el marco de evaluación mediante su aplicación en al menos tres MMDBMS, comparando su comportamiento bajo las mismas condiciones de carga y volumen de datos.
- Derivar conclusiones que orienten la toma de decisiones sobre la adopción de MMDBMS en entornos distribuidos reales.

1.3. Antecedentes.

Con el crecimiento de la información y la complejidad de las operaciones organizacionales, los sistemas computacionales se volvieron cada vez más importantes para el almacenamiento y el procesamiento de datos. Las personas responsables de gestionar esta información (denominadas encargados de la información) desarrollaron programas para procesar, almacenar y generar informes. Sin embargo, estos programas se volvieron difíciles de mantener debido a la gran cantidad de operaciones que los usuarios solicitaban, lo que provocó que la información comenzara a presentar inconsistencias [1] [10].

A mediados de los años sesenta, se desarrollaron los modelos de datos jerárquico y de red para atender el problema de la inconsistencia [10]. En el modelo jerárquico, los

[1] Inconsistencia se refiere a la situación en que el mismo dato aparece con valores distintos en diferentes partes del sistema, o en que los datos dejan de reflejar correctamente la realidad que representan, lo que compromete la confiabilidad de la información.

datos se organizan en estructuras de árbol con relaciones padre-hijo; en el modelo de red, los datos se representan mediante grafos que permiten relaciones más complejas entre registros. Los primeros sistemas manejadores de bases de datos (SGBD) o *Database Management System* (DBMS por sus siglas en inglés) utilizaban estos modelos para representar los datos [10].

Aunque estos modelos permitieron almacenar y organizar datos de forma más eficiente, presentaron algunos inconvenientes, como la fuerte dependencia entre los datos. Estas limitaciones llevaron al desarrollo del modelo relacional, propuesto por Edgar Frank Codd en 1969 [2]. Este modelo resolvió varios de los problemas anteriores al introducir la independencia física [12] y lógica [12] de los datos, y al fundamentarse en conceptos de lógica, teoría de conjuntos y álgebra relacional, lo que sentó las bases para el desarrollo del lenguaje SQL [10].

Es importante aclarar que la redundancia[2] sigue siendo un problema potencial en los sistemas de bases de datos, incluso en el modelo relacional. Para mitigar este problema se utilizan técnicas como la normalización [13].

Las bases de datos relacionales se convirtieron en el estándar de almacenamiento y procesamiento de datos debido a su simplicidad y capacidad para gestionar los datos estructurados [14]. Sin embargo, con el paso del tiempo la información empezó a estar conformada por datos estructurados, semiestructurados y no estructurados. Como consecuencia, la incapacidad de los DBMS relacionales para almacenar de forma nativa datos semiestructurados y no estructurados derivó en la necesidad de implementar procesos de transformación sobre los modelos de datos. Los procesos de transformación requerían un tiempo considerable, lo que repercutía negativamente en el rendimiento de los DBMS relacionales. Esta limitación en el manejo de datos semiestructurados y no estructurados constituyó uno de los principales factores que motivaron el surgimiento de las bases de datos NoSQL [15].

A mediados de los años 2000 surge el término NoSQL, abreviatura de *Not only SQL*. Estos sistemas no se basan en el modelo físico de tablas y relaciones, ni utilizan el lenguaje SQL para realizar consultas; optan por enfoques distintos para el almacenamiento y la recuperación de datos [16]. El detonador principal de su adopción fue la necesidad de mayor velocidad y escalabilidad para trabajar con conjuntos de datos masivos, aunque además ofrecen la ventaja de poder almacenar datos semiestructurados con menor tiempo de procesamiento [16].

La creciente adopción de los DBMS NoSQL impulsó un nuevo paradigma en la gestión de datos: la *persistencia políglota* [17]. Este enfoque reconoce que no existe un único DBMS que se adapte a todas las necesidades, y propone la integración de distintos DBMS en una arquitectura de datos, donde cada uno se especializa en un modelo

[2] El término *redundancia* consiste en la duplicación de datos con posibilidad de desincronización, es decir, que la misma información almacenada en múltiples lugares pueda actualizarse en un sitio y no en otro, generando inconsistencias.

específico (por ejemplo, Oracle Database para el modelo basado en relaciones y Neo4j para el modelo basado en grafos).

Si bien la persistencia políglota ofrece flexibilidad para la integración de datos, no está exenta de desventajas. La complejidad de administrar múltiples DBMS, la dificultad para garantizar la consistencia de los datos y la implementación de transacciones distribuidas son algunos de los desafíos que presenta [17]. A esto se suman la falta de herramientas maduras para su gestión, el incremento en los costos operativos y una pronunciada curva de aprendizaje asociada a las distintas tecnologías involucradas.

Para atender estas problemáticas, surgen los MMDBMS [1], los cuales proporcionan un medio para simplificar entornos donde coexisten múltiples modelos de datos. Los MMDBMS permiten almacenar distintos modelos (estructurados, semiestructurados y no estructurados) en una sola instancia, reduciendo así la complejidad y los costos propios de una arquitectura de datos heterogénea.

La necesidad de gestionar esta diversidad de modelos con el menor impacto posible responde, en gran medida, a la popularidad de internet, que detonó un crecimiento exponencial en la cantidad y variedad de datos generados diariamente [18]. El volumen, variedad, velocidad, valor y veracidad de estos datos impulsaron el desarrollo de nuevas herramientas y tecnologías para su gestión, dando lugar a lo que hoy se conoce como Big Data [19]. En el presente trabajo se abordará particularmente la dimensión de la variedad de los datos.

1.3.1. Tendencias en los modelos y manejadores de bases de datos.

Existen diversos tipos de manejadores de bases de datos que buscan mejorar el rendimiento, la escalabilidad y la flexibilidad. La Tabla 1 presenta una comparación de los principales tipos, sus ventajas, desventajas y ejemplos representativos.

Tabla 1. Comparativa de tipos de manejadores de bases de datos.

Tipo	Ventajas	Desventajas	Ejemplos
Distribuidas	Alta escalabilidad horizontal; reduce latencia distribuyendo datos cerca de los usuarios.	Los fallos de red pueden comprometer la consistencia e integridad en transacciones.	Microsoft SQL Server, Oracle Database.
NoSQL	Alta escalabilidad horizontal; menor necesidad de transformaciones al leer o escribir datos.	No siempre cumplen las propiedades ACID. Sus lenguajes de consulta son específicos a cada herramienta, a diferencia del ampliamente conocido SQL.	Apache Cassandra, Couchbase.

Multimodelo (MMDBMS)	Admiten datos estructurados, semiestructurados y no estructurados en una sola instancia; eliminan transformaciones complejas entre modelos.	Pueden presentar conflictos en las propiedades ACID. Sus lenguajes de consulta son propios de cada herramienta.	ArangoDB, OrientDB.
En la nube (modo de despliegue)	Escalabilidad elástica; infraestructura administrada por terceros; reducción de costos de inversión.	Riesgos de seguridad al no tener control total sobre la infraestructura.	Amazon Web Services, Microsoft Azure.

Si bien los manejadores de bases de datos que se mencionaron ostentan un lugar preponderante en el panorama actual, la constante evolución en la eficiencia, la escalabilidad y el procesamiento de datos exige realizar un análisis más profundo por cada tecnología. En este contexto, las MMDBMS se presentan como una alternativa innovadora que combina las ventajas de los diferentes modelos de datos en un único sistema. Este trabajo se enfocará en realizar un análisis a las bases de datos multimodelo.

1.3.2. Tendencias en el procesamiento de la información.

Las MMDBMS se perfilan como una de las tendencias más relevantes en el almacenamiento de datos, transformando la forma en que se procesa la información [29]. Las MMDBMS ofrecen una flexibilidad al poder albergar y gestionar una amplia variedad de modelos de datos sin necesidad de convertirlos a un único modelo canónico. Esta flexibilidad permite eliminar la necesidad de realizar procesos ETL (Extract, Transform, Load) para integrar y procesar los datos por separado lo que supone un ahorro de tiempo y esfuerzo considerable.

Además, las MMDBMS son altamente adaptables a las necesidades cambiantes de las empresas [29]. Permiten integrar una variedad de modelos de datos sin necesidad de realizar cambios drásticos en la infraestructura de la base de datos [29], lo que las convierte en una herramienta indispensable en un mundo donde los datos son cada vez más heterogéneos y dinámicos. Su capacidad para almacenar y gestionar diferentes modelos de datos en una única plataforma facilita enormemente el análisis local de datos. Los analistas pueden acceder a todos los datos que necesitan desde un único lugar, lo que reduce el tiempo y el esfuerzo necesarios para obtener insights [3] valiosos.

Sin embargo, para aprovechar al máximo las ventajas de las MMDBMS, es crucial elegir el modelo que mejor se ajuste a las cargas de trabajo, de almacenamiento, consulta y análisis para cada caso de uso específico. En este contexto, la evaluación

exhaustiva de las diferentes opciones disponibles en el mercado se ha convertido en un paso fundamental para elegir la mejor tecnología. Es, por tanto, que se han realizado diversas comparativas entre los manejadores de bases de datos existentes. Con base en el objetivo planteado, a continuación se describen brevemente los *benchmarks* que a la fecha se han realizado entre manejadores de bases de datos multimodelo.

1.4. Comparaciones de bases de datos multimodelo.

La evaluación del desempeño de los MMDBMS ha adquirido gran importancia debido a su capacidad de manejar diversos modelos de datos dentro de un único sistema. El *benchmarking* juega un papel fundamental en esta evaluación [5], permitiendo comparaciones estandarizadas y destacando las fortalezas y debilidades de las diferentes ofertas de MMDBMS.

En esta área han surgido varios esfuerzos de investigación; en este trabajo se analizarán algunos de los trabajos publicados en sitios de divulgación científica y tecnológica.

UniBench [5] se centra en la comparación de dos MMDBMS (OrientDB y ArangoDB) en la cual diseñan una serie de pruebas (consultas y transacciones) a ejecutar en los entornos de trabajo. El objetivo de estas pruebas es crear un marco de evaluación que pueda ser aplicado a cualquier MMDBMS, permitiendo una comparación objetiva y estandarizada de sus funcionalidades y rendimiento.

La problemática que aborda UniBench [5] radica en la falta de un método de comparación óptimo para los MMDBMS. Esta situación dificulta la elección del sistema más adecuado para cada caso particular, ya que no existe una referencia clara que permita evaluar las diferentes opciones de forma consistente y precisa. UniBench [5] propone un marco de trabajo para evaluar el rendimiento de los MMDBMS, el cual aborda dos desafíos principales:

- La generación de datos multimodelo.
- El diseño e implementación de cargas de trabajo.

Para medir el rendimiento UniBench [5] plantea 10 consultas (de lectura, ordenamiento, de funciones agregadas y de JOINS) que ejecutarán las MMDBMS. Adicional a las consultas se evaluará el rendimiento de 2 transacciones (inserción y actualización de datos) a través del uso del *commit* y del *rollback*.

Lo que se evalúa en UniBench [5] es el tiempo que tardan los MMDBMS en importar y generar los datos (en conjuntos de 1 GB, 10 GB y 30 GB), el tiempo que tardan en

realizar las consultas y las transacciones sobre esos datos. Los modelos de datos que se utilizaron son de tipo JSON, grafos, relacionales y pares llave-valor. Dado que cada modelo de dato tiene características propias, la forma en que se relacionan no es trivial y se requirió planificar estrategias para que interoperen con el menor impacto posible en el rendimiento.

UniBench [5] concluye que, de acuerdo con las pruebas realizadas, ArangoDB es el mejor MMDBMS entre ArangoDB y OrientDB. Sin embargo, existen algunas razones para ser cautelosos con esta conclusión. En primer lugar, la falta de detalle en la metodología de las pruebas dificulta la evaluación de su confiabilidad. Sería deseable contar con una mayor transparencia en cuanto a los datos utilizados, las métricas de evaluación y la configuración de ambos sistemas.

En segundo lugar, la ausencia de pruebas adicionales en las que OrientDB pueda destacar genera dudas sobre la imparcialidad del estudio. Se deberían realizar estudios independientes con metodologías transparentes que consideren las fortalezas y debilidades de ambos sistemas.

M. Macak, et. al. realizó un *benchmark* que compara el rendimiento de los MMDBMS contra las NoSQL [17]. La diferencia fundamental radica en que las bases de datos multimodelo pueden almacenar y procesar diversos modelos de datos en una única instancia, mientras que las NoSQL se especializan en un solo modelo de datos. Los autores del *benchmark* plantean la siguiente pregunta: ¿en qué casos es más adecuado usar una MMDBMS y una NoSQL? El *benchmark* destaca que, si bien existen estudios previos que comparan bases de datos NoSQL contra bases de datos multimodelo, ninguno de esos estudios lo había hecho en un entorno clusterizado. Los autores resaltan la importancia de realizar pruebas en un entorno clusterizado, ya que este se asemeja más a un escenario real de uso. En este contexto, las pruebas realizadas por los autores ofrecen una perspectiva más precisa del rendimiento de las bases de datos multimodelo en comparación con las NoSQL. El *benchmark* [17] compara el rendimiento de tres manejadores de bases de datos: OrientDB, MongoDB y Neo4j. Para ello, se diseñaron operaciones DML (Data Manipulation Language) específicas para cada par de manejadores, con el objetivo de evaluar su desempeño en diferentes modelos de datos. Las pruebas se dividieron en dos conjuntos:

- Comparación de MongoDB y OrientDB con modelo de datos grafo: se realizaron diversas operaciones DML para evaluar la capacidad de ambos manejadores para almacenar, actualizar y eliminar datos en estructuras de grafos. Dichas operaciones fueron de lectura, escritura y de funciones agregadas sobre un conjunto que contenía 22.3 GB de datos.
- Comparación de OrientDB y Neo4j con modelo de datos documento: se ejecutaron diferentes consultas para medir el rendimiento de ambos manejadores al trabajar con documentos JSON. Las consultas fueron de lectura, escritura y de funciones agregadas sobre un conjunto que contenía 18.3 GB de datos.

El objetivo principal del *benchmark* [17] es determinar qué manejador de bases de datos ofrece el mejor desempeño para generar y procesar datos específicos. Los resultados de las pruebas pueden ser de gran utilidad para los desarrolladores que necesitan elegir la base de datos más adecuada para sus proyectos.

Al igual que UniBench [5], la única métrica que se midió fue el tiempo de ejecución de algunas consultas sobre algunos *sets* de datos ya obtenidos previamente. Los resultados del *benchmark* mostraron que, en la mayoría de las pruebas, las NoSQL superaron a las MMDBMS, especialmente en tareas con cantidades de datos pequeñas. Sin embargo, derivado de que existen pruebas de que a medida que se incrementó el volumen de datos y la carga de trabajo, las MMDBMS empezaban a igualar e incluso superar a las NoSQL, especialmente en las pruebas OLAP. A partir de estos resultados, los autores del *benchmark* [17] proponen algunas conclusiones importantes:

- Las bases de datos multimodelo son una buena opción cuando no se conoce con certeza el volumen y la variedad de los datos que se van a procesar, debido a su sencillez en escalar horizontal y verticalmente.
- En el contexto específico de este *benchmark*, las bases de datos NoSQL mostraron un mejor desempeño que las MMDBMS para tareas con cantidades de datos pequeñas y bien definidas. Cabe aclarar que esta conclusión se refiere al desempeño relativo entre MMDBMS y NoSQL en las pruebas realizadas, y no a una caracterización general de NoSQL.

El *benchmark* M2Bench [4] compara los MMDBMS contra la técnica de persistencia políglota[3]. Los autores contextualizan el estudio destacando la evolución de las tecnologías de bases de datos, desde el predominio inicial de las bases de datos SQL, al auge de las NoSQL, la aparición de la persistencia políglota y, finalmente, el desarrollo de las bases de datos multimodelo. M2Bench [4] identifica una limitación en los *benchmarks* existentes: la mayoría se centran en evaluar a lo mucho dos modelos de datos. Para superar esta limitación, el estudio presentado explora el potencial de las bases de datos multimodelo en tres escenarios típicos: comercio digital, salud y desastres naturales. Además, emplea los cuatro modelos de datos más utilizados para análisis: relacional, documento, grafo y arreglo.

M2Bench [4] presenta un análisis comparativo de dos MMDBMS: ArangoDB y AgensGraph contra la implementación de cuatro DBMS: MySQL, MongoDB, Neo4j y SciDB como si fueran un solo sistema de base de datos, es decir, estos DBMS se integran en una sola infraestructura y realizan procesos de transformación entre modelos de datos. El objetivo de M2Bench [4] es ayudar a los usuarios a comprender cómo la elección del modelo de datos impacta en el rendimiento de un sistema de base de datos multimodelo.

[3] Técnica arquitectónica que consiste en usar múltiples motores de bases de datos, cada uno especializado en un modelo de datos particular dentro de una misma aplicación.

Los resultados obtenidos en las pruebas sugieren que en aquellas tareas en donde se ocuparon arreglos, la persistencia políglota permitió aprovechar el modelo de datos basado en arreglos al integrar SciDB, lo cual no es posible en ArangoDB y AgensGraph que no soportan arreglos de manera nativa. Sin embargo, para las tareas que involucraban grafos y relacionales tuvo un pésimo desempeño, ya que tardó demasiado en realizar las tareas. Por otra parte, ArangoDB tuvo un muy buen desempeño en cuanto a consultas donde se involucraban los grafos, y AgensGraph, a pesar de no tener un desempeño tan bueno como el de ArangoDB, tuvo un desempeño aceptable en cuanto al manejo de grafos, pero un desempeño superior que la persistencia políglota en el manejo de los relacionales.

M2Bench [4] concluye que las cargas de trabajo y los escenarios que propone son un avance para medir el desempeño de los MMDBMS, y que cada MMDBMS mostró un rendimiento diferente en función de las características específicas de cada tarea. En general, el estudio destaca la importancia de elegir el sistema de base de datos adecuado para cada caso particular. No existe una solución única para todas las necesidades, y el mejor sistema dependerá de las características específicas de la tarea que se desea realizar.

El siguiente y último *benchmark* holístico presentado en [8] que se analiza es la continuación de UniBench [5], ya que usan el mismo generador de datos (agregando algunos conjuntos de datos para enriquecer su generador), las mismas cargas de trabajo, pero en esta ocasión agregan a un MMDBMS en esta prueba, AgensGraph.

Las motivaciones y los objetivos siguen siendo los mismos que UniBench [5]. Sin embargo, al modificar algunos parámetros en los MMDBMS y al describir con mayor precisión las pruebas, se vuelve interesante volver a comparar los resultados. Aunque sean las mismas consultas, los autores decidieron modificar el tamaño de los conjuntos de datos por cada consulta a ejecutar; de esta manera se puede observar cómo es el rendimiento de los MMDBMS al variar el volumen del conjunto de datos. Los resultados mostraron que AgensGraph en general fue más rápido para el procesamiento de las consultas sobre un conjunto de datos pequeño, mientras que OrientDB es mejor para importación de datos y ArangoDB es más eficiente a medida que la cantidad de datos con la que operaba crecía.

El *benchmark* holístico [8] fue mucho más transparente que UniBench [5], y fueron mucho más detallados con las cargas de trabajo, las consultas y los conjuntos de datos: AgensGraph se destacó por su velocidad en la importación de datos, siendo 1.5 y 9 veces más rápido que ArangoDB y OrientDB; además, superó a los demás MMDBMS en la ejecución de consultas de agregación complejas, obteniendo una ventaja de 9 y 25 veces sobre ArangoDB y OrientDB respectivamente. En cuanto al procesamiento de consultas, OrientDB sobresale en consultas puntuales y consultas orientadas a rutas, siendo 3.6 y 36 veces más rápido que ArangoDB y AgensGraph respectivamente. Por otro lado, ArangoDB se posiciona como el mejor en filtrado de

documentos y unión de consultas, con una velocidad 4.5 y 5 veces superior a la de OrientDB y AgensGraph respectivamente.

El *benchmark* holístico [8] concluye que se necesitan optimizar sus consultas (como el uso de índices y planes de ejecución) para determinar si estas técnicas pueden ayudar a aumentar el rendimiento, y por lo tanto se necesitan más trabajos en esta área.

En la Tabla 2 se muestra un resumen de los artículos analizados en esta sección, destacando en cada columna las ventajas y desventajas que tienen por cada artículo.

Tabla 2. Resumen comparativo de artículos de MMDBMS.

Artículo	Ventaja	Desventaja
UniBench [5]	Describe con claridad la problemática actual en el *benchmarking* de MMDBMS. Compara los MMDBMS más populares y es útil para la toma de decisiones.	Carece de detalles técnicos de las pruebas, se enfoca solo en tiempos de respuesta y no compara otras métricas.
Macak et al. [17]	Ofrece una comparación entre MMDBMS y DBMS populares como OrientDB y MongoDB.	Solo evalúa tiempos de respuesta; omite CPU y RAM.
M2Bench [4]	Presenta escenarios analíticos realistas, explica las consultas y detalla resultados.	No considera métricas distintas al tiempo de respuesta.
Benchmark holístico [8]	Complementa a UniBench con mayor detalle.	Tiene potencial para enriquecerse con escenarios adicionales.

De los *benchmarks* anteriormente analizados, se observa que persisten varias limitaciones: la mayoría no contempla la implementación ni evaluación de los MMDBMS en ambientes distribuidos, no se abordan aspectos críticos como la integridad y consistencia de la información en aplicaciones transaccionales, y rara vez se considera el monitoreo del uso de recursos como CPU y memoria RAM durante las pruebas.

2. Planteamiento y Análisis del Problema.

La gran variedad que existe en los datos que participan en la vida cotidiana de las personas y en las operaciones de las empresas se ha convertido en un reto para los responsables de la información [32]. En entornos donde se tienen que procesar diferentes modelos de datos, los DBMS SQL y NoSQL deben procurar tener tiempos de respuesta rápidos para los distintos procesos con los diferentes modelos de datos que se necesitan en las operaciones de las organizaciones [3], situación que algunas veces puede repercutir en los costos de implementación y mantenimiento de nuevas tecnologías.

Las MMDBMS atienden el problema del almacenamiento de diversos modelos de datos, simplificando la arquitectura del sistema [3]. A diferencia de otros DBMS que manejan un único modelo de dato por instancia, las MMDBMS pueden manejar datos estructurados, semiestructurados y no estructurados en una única instancia, lo que reduce la complejidad de la arquitectura y resulta más eficiente que implementar la técnica de la persistencia políglota [34], [35].

2.1. Problemática.

La proliferación de diferentes tipos de MMDBMS ha creado un panorama complejo para la comunidad que busca una solución para sus necesidades. Cada tipo de MMDBMS tiene sus propias ventajas y desventajas [36], lo que dificulta la elección de la herramienta adecuada. Algunas medidas que pueden solucionar el problema de la elección de la herramienta adecuada son la creación de *benchmarks* estandarizados que evalúen el rendimiento de los MMDBMS en diferentes condiciones y la elaboración de trabajos que expliquen las ventajas y desventajas de cada tipo de MMDBMS.

En la actualidad se han realizado diversos *benchmarks* de MMDBMS [4], [5], [6], [7], [17]. Sin embargo, estos son un tanto limitados: como señalan algunos autores, *"existen algunos programas de benchmark que asumen múltiples modelos de datos, pero los modelos que soportan son bastante limitados" [4, p. 748, traducción propia]*. Además, muchos de estos *benchmarks* [3], [4], [5], [7] se realizan en un ambiente centralizado. Hoy en día muchas empresas usan ambientes distribuidos para sus operaciones [38], lo que hace necesario realizar comparativas bajo esta condición.

Además, algunos de estos *benchmarks* [3], [4], [5], [7] se realizan en un ambiente centralizado. Realizar comparaciones de MMDBMS en ambientes centralizados puede

generar sesgos que no reflejan la realidad de un ambiente distribuido. Esto se debe a que las condiciones de un ambiente centralizado no siempre son replicables en un escenario real, donde la distribución de datos, la carga de trabajo y la infraestructura pueden variar considerablemente [37].

Hoy en día muchas empresas usan ambientes distribuidos para sus operaciones [38]. Esto significa que los recursos informáticos, como servidores, aplicaciones y datos, se distribuyen en diferentes ubicaciones físicas, por lo que se vuelve necesario realizar *benchmarks* que trabajen bajo esta condición.

El presente trabajo propone KhaBench, un *benchmark* para evaluar y comparar el rendimiento de sistemas gestores de bases de datos multimodelo en entornos distribuidos, midiendo no solo tiempos de respuesta sino también uso de CPU, memoria RAM y consistencia observable, con el fin de proveer información que apoye la toma de decisiones informadas sobre la adopción de estas tecnologías.

2.2. Análisis del Problema.

Aunque la adopción de entornos distribuidos ha experimentado un notable crecimiento en los últimos años [38], persiste una carencia destacable de *benchmarks* específicos para MMDBMS en este contexto. Esta falta de herramientas dificulta una evaluación precisa del rendimiento del sistema, así como la identificación de áreas de mejora y la toma de decisiones informadas sobre inversiones en infraestructura y aplicaciones [39].

La evaluación del rendimiento y el uso de recursos en entornos distribuidos es un área de investigación que requiere mayor atención, dado que la mayoría de los *benchmarks* disponibles no están adaptados a este tipo de arquitecturas. Esta ausencia de *benchmarks* adecuados para MMDBMS en entornos distribuidos genera una carencia de información crítica para desarrolladores, administradores de bases de datos e investigadores.

Si bien los *benchmarks* que hemos podido analizar hoy en día han contribuido a resaltar la importancia de estas tecnologías, se han limitado principalmente a medir el tiempo de ejecución de consultas y transacciones. Sin embargo, en un contexto donde las empresas necesitan un análisis detallado del uso de recursos de sus sistemas, es esencial no solo medir los tiempos de respuesta, sino también otros aspectos como el uso de CPU y memoria RAM.

A partir de los planteamientos establecidos en la sección anterior, se puede observar que no existe detalle técnico en la generación y pruebas de cargas de trabajo y la forma en la que se realizaron las consultas en los MMDBMS en cuestión.

Adicionalmente, en las comparativas [5] y [17] se abordaron entornos centralizados. Sería muy interesante realizar *benchmarks* bajo entornos distribuidos. En un

ambiente distribuido se tiene la oportunidad de evaluar aspectos como las capacidades de fragmentación de datos en ambientes distribuidos y multimodelo y el procesamiento distribuido en transacciones y consultas. Las mediciones realizadas han sido principalmente de latencia en consultas y transacciones en ambientes centralizados, por ejemplo, en [4], [5], [8] y [17]. En un ambiente distribuido las métricas como el uso de memoria y CPU, y latencia de transacciones distribuidas son igualmente importantes.

Los *benchmarks* existentes no están diseñados para ejecutarse en entornos distribuidos, lo que limita su capacidad para evaluar el rendimiento y la escalabilidad de los MMDBMS en este contexto [40]. Por lo tanto, se ha detectado que hacen falta *benchmarks* que contemplen las capacidades de las tecnologías MMDBMS en entornos distribuidos.

Actualmente, las empresas que buscan implementar MMDBMS en ambientes distribuidos carecen de comparativas que evalúen métricas como latencia, uso de CPU y memoria RAM, y control de consistencia e integridad. Esta ausencia dificulta la toma de decisiones informadas sobre la adopción de estas tecnologías en entornos transaccionales y analíticos [41].

En este trabajo se abordará una evaluación de los MMDBMS en entornos distribuidos. Se incluirá el detalle de las consultas, transacciones y las métricas relevantes de análisis, con el fin de ayudar a las empresas a mejorar su rendimiento, reducir costos y tomar decisiones más informadas sobre la adopción de estas tecnologías.

3. Marco Teórico.

El presente trabajo busca evaluar el rendimiento de múltiples bases de datos multimodelo distribuidas mediante un *benchmark*. Para comprender este estudio, se abordarán conceptos como las bases de datos multimodelo, sus capacidades para operar en entornos distribuidos, los motores de almacenamiento que utilizan, así como los procesos de extracción, transformación y carga (ETL) involucrados.

3.1. Bases de datos multimodelo.

Los Sistemas Gestores de Bases de Datos Multimodelo o *Multi Model Database Management System* (MMDBMS por sus siglas en inglés) son plataformas tecnológicas que permiten almacenar y gestionar múltiples modelos de datos a través de una interfaz unificada y de un solo lenguaje de consulta [42].

Este tipo de herramientas tiene como objetivo resolver las problemáticas que presentan algunos manejadores de bases de datos, por ejemplo, tratar de almacenar un dato XML en un manejador puramente relacional [43], o el aumento en la complejidad de la administración y la disminución del rendimiento que implica implementar la persistencia políglota [34][35] sin escalar adecuadamente la infraestructura, lo que genera un aumento en los costos [3]. Una ventaja adicional de los MMDBMS es que, al mantener los datos en su forma nativa, también se evitan las inconsistencias que podrían surgir de múltiples procesos de transformación.

Los MMDBMS ofrecen una solución para gestionar la diversidad de modelos de datos en los sistemas de *Data Warehouse* (DWH, o almacén de datos), que son repositorios centralizados de información integrada proveniente de múltiples fuentes, fundamentales para las aplicaciones de Procesamiento Analítico en Línea (OLAP) y la inteligencia de negocios (BI por sus siglas en inglés) [43].

Anteriormente, la manera en que se manejaba la variedad en los datos en los proyectos de software era a través del uso de la *persistencia políglota*, en la cual se combinaban diferentes soluciones de DBMS de acuerdo con el modelo de dato a usar, es decir, usarían una plataforma para almacenar datos relacionales, otra plataforma para almacenar datos llave-valor y otra plataforma para almacenar datos basados en grafos, solo por mencionar algunos modelos de datos [44]. Ahora, con los MMDBMS, se pueden almacenar diferentes modelos de datos sin la necesidad de realizar complejos procesos de transformación y de integración [42], lo que permite que puedan afrontar los cambios a lo largo del tiempo de operación de manera más eficiente, menos costosa y evitando las inconsistencias.

De acuerdo con ArangoDB, los MMDBMS pueden almacenar diversos modelos de datos porque estos comparten características estructurales. Por ejemplo, los modelos de datos documento generalmente se almacenan con formato JSON. Utilizando este formato se pueden manejar los datos llave-valor debido a la naturaleza misma del formato. Y finalmente para transformar los datos a grafos, se necesitaría un documento JSON para obtener los vértices y otro documento JSON para obtener las aristas [45]. Finalmente se implementa un lenguaje de consulta que unifique estas relaciones entre los modelos de datos.

3.1.1. Características y tipos de bases de datos multimodelo.

En comparación con otros manejadores, los MMDBMS se popularizaron por un conjunto de características que pueden ofrecer un mayor beneficio a aquellos sistemas que requieren el manejo de diversos modelos de datos [42].

La primera característica en que se debe profundizar es aquella que la distingue de las demás, la del soporte de múltiples modelos de datos en una misma plataforma. Un MMDBMS tiene la particularidad de que a través del uso de un solo lenguaje de consulta (nativo al manejador en cuestión) puede unificar los procesos de consulta, transacción y escritura de datos en disco [43]. Esta unificación se debe a cómo está definida su arquitectura, que de manera muy general se explica a continuación.

La Figura 1 ilustra, a manera de arquitectura, las capas que componen un MMDBMS en general.

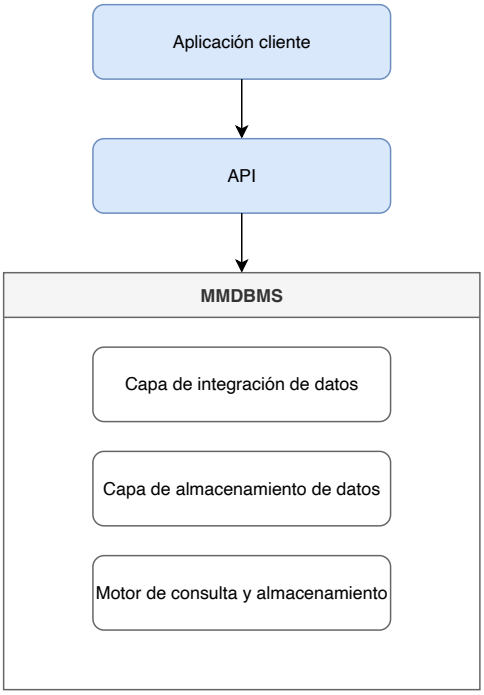


Fig. 1. Arquitectura por capas general de un MMDBMS.

La capa de presentación es un programa arbitrario que se ejecuta sobre un sistema de cómputo externo al MMDBMS y actúa como su cliente. Se comunica con el MMDBMS mediante un API (*Application Programming Interface*), que constituye la interfaz principal de acceso al manejador. Esta capa queda fuera de los límites del MMDBMS propiamente dicho.

La capa de integración de datos es responsable de la integración de las diferentes fuentes y modelos de datos en la base de datos. Puede incluir conectores para bases de datos relacionales, sistemas NoSQL, archivos CSV y XML, entre otros [46].

La capa de almacenamiento de datos define cómo se estructuran los datos dentro de los archivos en disco. Determina los formatos internos de representación, los mecanismos de indexación y la organización lógica de los datos, independientemente del dispositivo físico en que el sistema operativo los aloje.

El motor de consulta y almacenamiento es el núcleo de los MMDBMS. Se encarga de procesar las consultas de los usuarios y de realizar el proceso de indexación y almacenamiento de los datos mediante un lenguaje de consulta que aprovecha las similitudes entre los modelos para poder manipularlos [48][15]. Cualquier administrador de bases de datos (DBA) puede seleccionar y afinar cómo se realiza el almacenamiento.

Otra característica que se debe resaltar de los MMDBMS es que la manera en la que estos leen y almacenan los datos debe de ser lo más "natural" posible [42], por ejemplo, si se almacena un dato de tipo documento, se debe respetar ese modelo de dato al momento de almacenarlo, sin realizar algún procedimiento complejo de por medio. Los MMDBMS deben poder almacenar datos de manera nativa en al menos los siguientes formatos: JSON, texto, XML, RDF y binarios [42].

Los MMDBMS deben poder indexar los modelos de datos, es decir, la indexación debe ser un proceso completamente integrado en el software, sin necesidad de buscar algún complemento que permita indexar algún dato en un modelo diferente [42]. De manera similar se debe poder usar esquemas definidos por el manejador (por ejemplo, en un DBMS que trae un esquema por defecto), y definidos por un usuario (por ejemplo, XSD para datos XML) [42].

Finalmente, un MMDBMS debe mantener un alto rendimiento para consultas en todos los modelos de datos, y mejorar la escalabilidad y tolerancia a fallos (algunos MMDBMS como MarkLogic implementan almacenamiento en caché y clusterización, lo que puede ayudar a cubrir estos puntos) [42].

Los MMDBMS ofrecen cualidades interesantes que los han posicionado en una situación en donde es necesario profundizar más en el potencial que pueden ofrecer, sin embargo, es necesario analizar las diferencias respecto a otros manejadores de bases de datos, para validar si el costo/beneficio de migrar a estas tecnologías es alto.

3.1.2. Procesos de extracción, transformación y carga en MMDBMS y DBMS.

En la presentación de Luca Garulli titulada "NoSQL Adoption - What's the Next Step", durante la conferencia "NoSQL Matters" en el año 2012, se asocia por primera vez el término multimodelo al área de las bases de datos [42]. En un principio se asociaba que un sistema era multimodelo cuando este integraba diferentes soluciones de bases de datos en su arquitectura, lo que hoy conocemos como persistencia políglota [42]. Sin embargo, con el surgir de los MMDBMS nativos se volvió indispensable comparar las diferencias que existen entre un MMDBMS con los DBMS que se siguen utilizando hoy en día (SQL y NoSQL principalmente).

El mayor desafío de la persistencia políglota no es el almacenamiento en sí, sino la integración de los datos. Cuando una organización maneja múltiples esquemas y modelos, surge la necesidad de implementar procesos ETL (*Extract, Transform, Load*, o Extracción, Transformación y Carga), que actúan como el puente que permite consolidar información de fuentes heterogéneas para centralizarla en un *Data Warehouse* (DWH, o almacén de datos), el cual sirve como repositorio unificado para el análisis estratégico. Estos procesos se componen de tres etapas:

Extraer (*Extract*): Consiste en obtener los datos de distintas fuentes, por ejemplo, un archivo local en disco, una interfaz de programación de aplicaciones (API por sus siglas en inglés) [56] o un sistema de planificación de recursos empresariales (ERP por sus siglas en inglés) [57]. Los datos extraídos son almacenados en su forma original, es decir, no hay alguna alteración en ellos.

Transformar (*Transform*): Consiste en procesar los datos de forma que sean congruentes a las necesidades de una organización [56]. Durante la fase de transformación los datos previamente extraídos (estructurados, semiestructurados o no estructurados) son procesados para obtener información de ellos. Estos procesos de transformación de datos son congruentes a las reglas de negocio de la organización, es decir, los distintos modelos de los datos se transforman a un modelo canónico para ser almacenados [57].

Cargar (*Load*): Consiste en almacenar los datos previamente transformados en un sistema de almacenamiento, como el mencionado DWH [56][57].

En la Figura 2 se ilustra, de manera general, cómo funcionan los procesos ETL.

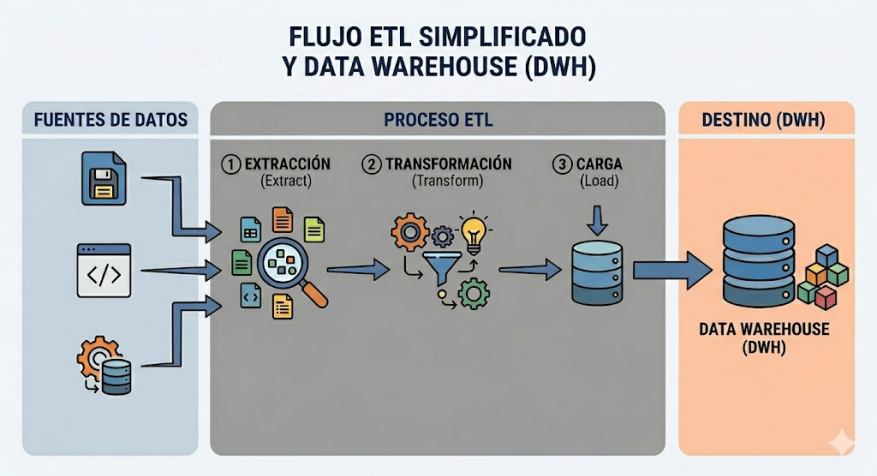


Fig. 2. Procesos ETL.

La distinción entre los DBMS tradicionales y los MMDBMS radica precisamente en el manejo de estos procesos. Mientras que en la persistencia políglota los tiempos de respuesta dependen de procesos ETL que suelen ser costosos y lentos, los MMDBMS eliminan gran parte de esta intermediación [43]. Al permitir el almacenamiento de múltiples modelos de datos de forma nativa, un MMDBMS reduce los costos operativos y computacionales.

Además, el manejo de diferentes esquemas en un MMDBMS permite que el procesamiento y análisis de datos tengan un valor agregado [43], ya que un MMDBMS es capaz de garantizar un soporte eficaz a la evolución e integración de diferentes esquemas.

Dado que un MMDBMS reduce los tiempos de tratamiento de distintos modelos de datos, a continuación se describen los tres manejadores evaluados en este trabajo: ArangoDB, OrientDB y Couchbase.

3.1.3. Descripción de los manejadores de bases de datos multimodelo a comparar.

A continuación, se describen los tres MMDBMS utilizados en este trabajo. Las secciones 3.1.4 a 3.1.7 presentarán, para cada uno de estos tres manejadores, distintos ángulos de análisis: sus motores de almacenamiento y su manejo de los modelos de datos de grafo, llave-valor y documentos, respectivamente.

A) ArangoDB

Es un MMDBMS de código abierto lanzado en el año 2011 y desarrollado en C++, puede manejar y almacenar los modelos de datos de documentos, grafos y llave-valor en un único sistema [61]. ArangoDB utiliza un lenguaje de consulta declarativo nativo llamado AQL (*ArangoDB Query Language*) que permite a los desarrolladores realizar

consultas complejas en sus datos, incluyendo consultas que abarcan múltiples modelos de datos. ArangoDB permite implementar bases de datos distribuidas mediante la conformación de un clúster con múltiples nodos [61].

B) OrientDB

OrientDB es un MMDBMS de código abierto que soporta los modelos de datos de documentos, grafos y clave-valor en un único sistema. OrientDB utiliza un lenguaje de consulta declarativo similar a SQL, lo que facilita a los desarrolladores familiarizados con SQL realizar consultas en sus datos [62]. Además, OrientDB permite la implementación de una base de datos distribuida, soporta la fragmentación de los datos, maneja algunos de los niveles de transparencia y ofrece soporte para la consistencia ACID de las transacciones [63].

C) Couchbase

Couchbase es un sistema MMDBMS de código abierto que combina en un solo entorno modelos de datos de documentos y clave-valor, usado en entornos distribuidos y aplicaciones que operan con datos semiestructurados [64][65]. Basado en una arquitectura de clúster escalable y *memory-first*, facilita bajos tiempos de respuesta con alto rendimiento [64][65]. Su lenguaje de consulta N1QL (SQL++), inspirado en SQL para JSON, permite escribir consultas complejas como si se tratase de un lenguaje de consulta relacional [64][65]. El sistema soporta transacciones ACID, incluso distribuidas a través de múltiples nodos [64].

3.1.4. Características generales de los motores de almacenamiento en los MMDBMS.

Los motores de almacenamiento son una característica fundamental para elegir la solución adecuada, ya que son el núcleo de los MMDBMS, responsables de la organización, acceso y gestión eficiente de los datos. A continuación se describen las estrategias de almacenamiento, las estructuras de datos y los algoritmos que emplean ArangoDB, OrientDB y Couchbase.

A) Motor de almacenamiento de ArangoDB

El motor de almacenamiento de ArangoDB se basa en RocksDB [66], un motor de almacenamiento llave-valor desarrollado originalmente por Facebook, el cual permite inserciones constantes incluso cuando la cantidad de datos supera la capacidad de la memoria principal [67]. Su arquitectura basada en bloqueos a nivel de documento permite las escrituras concurrentes sin afectar el rendimiento de las lecturas. Sin embargo, puede presentar conflictos de escritura al modificar simultáneamente un mismo dato; por ello tiene un límite de tamaño para las transacciones, lo que significa que las consultas AQL que modifican una gran cantidad de datos deben dividirse en transacciones más pequeñas.

Adicionalmente, para la gestión de páginas en disco, ArangoDB emplea un sistema de almacenamiento local paginado [67]. Este enfoque, basado en disco, utiliza un modelo de página para trabajar con la información y opera en conjunto con un registro de escritura anticipada (*Write-Ahead Log*, WAL).

B) Motor de almacenamiento de Couchbase

Couchbase emplea un motor de almacenamiento basado en memoria optimizado para entornos distribuidos y de baja latencia. Adopta una arquitectura *memory-first*, en la que todos los datos activos residen en memoria y se escriben de forma asíncrona al disco mediante un proceso de escritura diferida [68].

El sistema utiliza *buckets* como unidades lógicas de almacenamiento, que internamente emplean un diseño tipo *append-only* junto con mecanismos de replicación cruzada entre nodos (XDCR, *Cross Datacenter Replication*). El motor incorpora un sistema de *checkpointing* y de W*rite-Ahead Log* (WAL)

Para la escritura en disco, Couchbase utiliza el motor *Couchstore* (en versiones Community) y *Magma* (en versiones Enterprise) [69]. Cabe señalar que Couchbase no implementa la caché de disco de dos niveles propia de ArangoDB y OrientDB.

C) Motor de almacenamiento de OrientDB

OrientDB emplea un sistema de almacenamiento local paginado para albergar sus datos [70]. Los archivos se dividen en páginas, lo que permite que las operaciones sean atómicas a nivel de página. La caché de disco de dos niveles posibilita a OrientDB almacenar en caché las páginas de acceso frecuente, diferenciándolas de las de acceso esporádico, minimizando así la cantidad de búsquedas del cabezal del disco durante las escrituras de datos [70]. La caché de disco se compone de dos partes: caché de lectura y caché de escritura.

3.1.5. Manejo de datos basados en grafos en OrientDB y ArangoDB.

Los modelos de datos basados en grafos representan los datos como una red de entidades (nodos) y sus relaciones (aristas). Los MMDBMS OrientDB y ArangoDB tienen soporte para este modelo de dato, por otro lado Couchbase no lo implementa.

Los modelos de datos basados en grafos representan los datos como una red de entidades (nodos) y sus relaciones (aristas). Los MMDBMS OrientDB y ArangoDB tienen soporte para este modelo de dato, por otro lado Couchbase no lo implementa.

A) ArangoDB

ArangoDB manipula los grafos a través de sus propiedades. De este modo un nodo o una arista son un objeto JSON compuesto con un identificador único. Al ser un objeto JSON cada grafo puede tener un número indefinido de propiedades [59]. Refiérase a

[71] para más información sobre sus propiedades. Además, ArangoDB permite etiquetar nodos y aristas para relacionarlos a través de una colección.

Una colección puede ser nombrada y etiquetada bajo cualquier contexto que el sistema requiera. En ArangoDB, se tienen ciertas limitaciones en el uso de las aristas, por ejemplo, éstas siempre son dirigidas y cuentan con atributos especiales llamados "*_from*" y "*_to*" [72] para hacer referencia a otros nodos. Este modelo de grafos dirigidos implica que las relaciones creadas con aristas son unidireccionales, pero se puede generar una arista inversa para poder crear una relación bidireccional [59].

B) OrientDB

Al igual que ArangoDB, OrientDB manipula los grafos a través de sus propiedades, sin embargo, no representa a los nodos y aristas como un objeto JSON, sino que los representa como clases [73]. OrientDB implementa algunos de los pilares de la programación orientada a objetos (abstracción, herencia y polimorfismo) [74] para el manejo de los grafos, es decir, OrientDB ya tiene incluido dentro de su software acciones como crear clases, heredar clases, declarar atributos y métodos.

Cuando se requiere crear un nodo, se debe declarar una clase que va a heredar de la clase V (V de Vertex que significa vértice en inglés), de igual manera cuando se requiere crear una Arista se debe declarar una clase que va a heredar de la clase E (E de *Edge* que significa arista en inglés). Al igual que ArangoDB, las aristas son unidireccionales, pero se puede generar una arista inversa para poder crear una relación bidireccional [75].

3.1.6. Manejo de llave-valor en ArangoDB, Couchbase y OrientDB.

Los modelos de datos llave-valor se basan en un paradigma de almacenamiento simple y eficiente. Se componen de dos partes: la llave, que es un identificador único, y el valor, que puede ser cualquier tipo de dato. Cabe destacar que en todos los manejadores utilizados en este trabajo, los datos de llave-valor son un subconjunto de documentos (JSON principalmente).

A) ArangoDB

ArangoDB manipula los modelos de dato llave-valor como documentos, ya que cada documento tiene un atributo inmutable [59] denominado "*_key*" que permite identificar cualquier documento que esté almacenado dentro de la base de datos. Para generar este atributo se puede utilizar la función lambda incluida en el software (un algoritmo hash) [76], o bien se puede asignar manualmente, pero una vez generado ya no se podrá modificar. Como se mencionó en la sección "3.1.5. Manejo de datos basados en grafos en OrientDB y ArangoDB", ArangoDB puede agrupar objetos JSON a través de colecciones, es decir, cada colección tiene un índice en el atributo "*_key*" (índice primario), lo que permite identificar de manera muy rápida los datos en los

documentos de la manera <colección>/<índice>, todo esto gracias a la forma en la que está conformado un documento JSON [59].

B) OrientDB

OrientDB maneja los modelos de datos de tipo llave-valor mediante la creación de índices sobre las entidades, declarando el índice único sobre una clase [78]. Adicionalmente, OrientDB implementa un API REST que permite realizar operaciones sobre los pares llave-valor mediante peticiones HTTP estándar (GET, PUT, DELETE) [77], [79]. Un mensaje de respuesta 204 indica que la operación fue exitosa.

C) Couchbase

Couchbase implementa el modelo de llave-valor como una de sus funcionalidades principales, ya que su arquitectura original se deriva de Memcached, un sistema de almacenamiento en caché basado exclusivamente en pares llave-valor [64]. En Couchbase, cada documento JSON almacenado dentro de un *bucket* está asociado a una llave única inmutable, que actúa como identificador primario del documento [82]. Esta llave puede ser generada por el desarrollador o por el sistema, pero una vez creada, no puede modificarse. El acceso a los documentos puede hacerse directamente a través de la API de alto rendimiento del motor de datos (Data Service), la cual permite realizar operaciones GET, INSERT, REPLACE y REMOVE sobre los pares llave-valor [65]. Estas operaciones se ejecutan de forma eficiente gracias al diseño en memoria de Couchbase y a su sistema de replicación basado en buckets y nodos.

3.1.7. Manejo de datos basados en documentos en ArangoDB, Couchbase y OrientDB.

Los modelos de datos basados en documentos pueden representar datos no estructurados o semiestructurados en forma de objetos JSON, XML, YAML, entre otros. La principal característica de este tipo de modelo de dato es la flexibilidad para definir la estructura. En este caso, todos los MMDBMS tienen soporte para estos modelos de datos.

A) ArangoDB

ArangoDB maneja los modelos de datos documento como objetos JSON [59], cada objeto JSON tiene métodos que permiten su manipulación de una manera sencilla, por ejemplo, Marco Teórico un ordenamiento por algún criterio, una iteración entre los datos del documento [83], entre otros. La manera en la que ArangoDB recupera e ingresa datos en un documento JSON se asemeja a la de un dato llave-valor, en donde se proporciona la clave para conseguir el valor del dato específico, aunque también se puede manipular los datos a través de sus atributos. Los documentos en ArangoDB se acomodan por colecciones, cada colección es un contenedor lógico de documentos

[85]. Los documentos en una colección pueden tener diferentes estructuras y campos, lo que proporciona flexibilidad en el almacenamiento de datos.

B) OrientDB

OrientDB almacena documentos como estructuras dinámicas en las que cada campo funciona como un par llave-valor. Cada documento, permite almacenar cualquier valor (primitivos, otros documentos, arreglos, entre otros). Los documentos se agrupan en colecciones, por medio de las clases y *clústeres* [84]. Las clases introducen tipado para un mejor control de datos, mientras que los *clústeres* permiten la distribución horizontal en entornos de gran volumen. Un aspecto diferenciador de OrientDB es el concepto de *link*. Un *link* es un vínculo en donde múltiples documentos que pertenecen a una colección, pueden hacer referencia a un documento que pertenece a otra colección. Esto es similar a las llaves foráneas en una base de datos relacional [84]. Sin embargo, la diferencia en las llaves foráneas respecto a los *links* es en los tiempos de ejecución. Para hacer búsquedas usando llaves foráneas se necesitan realizar operaciones JOIN [85] (que implican un alto costo en tiempo), mientras que para realizar búsquedas usando links se necesitaría realizar recorridos [85], los cuales son mucho más eficientes que los JOINS.

C) Couchbase

Couchbase maneja los modelos de datos tipo documento utilizando estructuras JSON como su unidad básica de almacenamiento [82]. Cada documento representa un objeto JSON completo, compuesto por pares llave-valor que pueden incluir valores primitivos, objetos anidados o listas. Estos documentos se agrupan lógicamente en buckets, y cada uno es identificado por una clave única [64]. Couchbase permite insertar, consultar y modificar documentos. Además, permite acceder a atributos específicos de un documento utilizando notación punto (.), realizar filtros, ordenar resultados y agrupar por propiedades internas del JSON. Asimismo, permite almacenar documentos con estructuras variables dentro del mismo bucket, sin necesidad de definir un esquema fijo.

3.2. Bases de datos distribuidas.

Las bases de datos distribuidas son una colección de múltiples bases de datos lógicamente interrelacionadas a través de una red de datos [86]. Las bases de datos se encuentran ubicadas en diferentes nodos (no necesariamente a grandes distancias), pero a pesar de estar dispersas, funcionan como si fueran una única base de datos lógica [86], lo que significa que cualquier usuario conectado a cualquier nodo puede acceder a todos los datos como si estuvieran almacenados en un mismo lugar.

Los MMDBMS como ArangoDB, OrientDB y Couchbase soportan bases de datos distribuidas mediante la creación de diferentes instancias que se interconectan para

actuar como una sola base de datos. Esto permite aprovechar el procesamiento paralelo y la redundancia de datos, junto con el soporte de diversos modelos de datos en donde cada nodo no necesita realizar procesos de transformación, logrando así un rendimiento optimizado [61][63][65].

Para comprender de manera integral las capacidades de los MMDBMS en entornos distribuidos, las subsecciones siguientes abordan los aspectos técnicos que tienen mayor impacto en el rendimiento y la consistencia: el procesamiento distribuido de consultas, las técnicas de optimización y particionamiento, los mecanismos de garantía de propiedades ACID, la replicación y los sitios, la fragmentación y los niveles de transparencia, la transaccionalidad en los MMDBMS, y los mecanismos de desempeño y consistencia. Estos temas son relevantes porque, en conjunto, determinan el comportamiento de los sistemas evaluados en el *benchmark* KhaBench.

3.2.1. Procesamiento distribuido de consultas.

El procesamiento distribuido de consultas es una técnica utilizada en los sistemas de bases de datos tradicionales para ejecutar consultas de manera eficiente en un ambiente distribuido. En lugar de ejecutar consultas en una única base de datos centralizada, el procesamiento distribuido de consultas permite distribuir la carga de trabajo entre múltiples nodos o servidores que forman parte de la infraestructura distribuida [87]. En este enfoque, cuando se emite una consulta, el sistema distribuido puede dividir la consulta en subconsultas más pequeñas que pueden ser ejecutadas en paralelo en diferentes nodos de la red. Posteriormente, los resultados de estas subconsultas pueden ser combinados y procesados para producir el resultado final de la consulta original [88]. Para optimizar consultas, los esquemas globales y locales de fragmentación junto con el análisis de los planes de ejecución se vuelven herramientas recurrentes, ya que por medio del álgebra relacional, recorridos basados en costo y la indexación, se pueden optimizar las consultas.

Todos los MMDBMS utilizados en este trabajo cuentan con herramientas para analizar planes de ejecución. La manera en la que lo implementan es muy similar entre los tres MMDBMS, ejecutando la sentencia explain [89][90][91]. El presente trabajo busca medir las capacidades de procesamiento de consultas distribuidas en los MMDBMS.

3.2.2. Técnicas de optimización de consultas y estrategias de particionamiento de datos.

Las técnicas de optimización de consultas y las estrategias de particionamiento de datos son fundamentales para maximizar el rendimiento de cualquier manejador de bases de datos.

En el ámbito de los DBMS relacionales, la optimización de consultas se centra en el uso eficiente de índices, la reescritura de consultas complejas, la reducción de uniones y el uso Marco Teórico de sentencias para analizar los planes de ejecución [92].

Además, un DBMS relacional permite la división lógica de conjuntos de datos en partes más pequeñas y manejables. Estos conjuntos de datos se distribuyen en diferentes nodos de acuerdo con las necesidades y las reglas de negocio [93].

Mientras que en los DBMS no relacionales la optimización de consultas se centra en la indexación de los datos [94], el uso de procesos de agregación [95], el uso de sentencias para analizar los planes de ejecución y el uso de filtros específicos para cada modelo de datos (documentos, grafos, llave-valor). En el presente trabajo se usan estas estrategias de particionamiento durante la implementación de la base de datos distribuida multimodelo.

3.2.3. Algoritmos para garantizar las propiedades ACID en una transacción distribuida.

Los algoritmos para garantizar las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) en transacciones distribuidas son diseñados para mantener la integridad y coherencia de los datos en sistemas distribuidos, donde las transacciones pueden abarcar múltiples nodos o bases de datos.

- **Protocolo *two phase commit* (2PC):** Este es uno de los algoritmos más comunes para garantizar la atomicidad en transacciones distribuidas [108]. En este protocolo, un nodo maestro coordina la ejecución de una transacción entre múltiples nodos participantes. En la primera fase, el nodo maestro envía una solicitud de preparación a todos los nodos participantes. Cada nodo realiza las operaciones necesarias y luego responden al maestro indicando si están listos para confirmar (preparados) o abortar la transacción. En la segunda fase, el coordinador decide si confirmar o abortar la transacción en función de las respuestas de los nodos participantes. Couchbase es el único MMDBMS que implementa este protocolo [97], lo que le permite ofrecer garantías fuertes de consistencia en transacciones distribuidas. ArangoDB y OrientDB no implementan 2PC, por lo que no ofrecen este nivel de garantía.
- **Protocolo de compensación:** En lugar de coordinar la ejecución de la transacción, cada nodo participante ejecuta la transacción localmente y registra las acciones realizadas en un registro de cambios. Si ocurre un fallo en algún nodo, se ejecuta un proceso de compensación para deshacer las acciones realizadas en otros nodos y restaurar la consistencia de forma eventual. Todos los MMDBMS que se usan en este trabajo implementan un protocolo que sigue este principio [98], [63], [99], aunque con garantías más débiles que las que ofrece el 2PC.
- **Mecanismos de replicación sincrónica/asincrónica:** Para garantizar la disponibilidad y la consistencia en sistemas distribuidos, se utilizan mecanismos de replicación de datos. La replicación síncrona garantiza que los datos se escriban en múltiples nodos antes de que se confirme una transacción, mientras que la replicación asíncrona permite que los datos se repliquen después de que se haya confirmado una transacción, lo que puede mejorar el rendimiento, pero introduce un estado de consistencia eventual [115], en el que las réplicas pueden diferir temporalmente hasta que la actualización se propague a todos los nodos. Todos

los MMDBMS que se usan en este trabajo soportan configuración de réplicas [100], [101], [102].

El *benchmark* de este trabajo tiene como objetivo revisar el comportamiento de las propiedades ACID [117] (con soporte nativo o externo) en transacciones distribuidas para poder mostrar su rendimiento y sus capacidades de conservación de integridad y consistencia de los datos.

3.2.4. Sitios y replicación de una base de datos distribuida.

Un sitio o réplica se refiere a los diferentes nodos o ubicaciones físicas donde se almacenan los datos [103]. Cada sitio puede contener una porción de los datos totales y puede tener su propio conjunto de transacciones y operaciones de base de datos [104]. La replicación en una base de datos distribuida se refiere al proceso de mantener copias idénticas de los datos en múltiples sitios. Estas copias replicadas pueden ubicarse en diferentes nodos dentro de la misma red o en ubicaciones geográficas separadas. La replicación se realiza por varias razones, por ejemplo:

1. Al tener copias de los datos en múltiples sitios, se pueden distribuir las cargas de consulta y reducir la latencia al acceder a los datos desde ubicaciones cercanas a los usuarios [105].
2. Si un sitio falla, los usuarios aún pueden acceder a los datos desde otras réplicas, lo que mejora la disponibilidad del sistema [105]. No obstante, cuando dos sitios sufren una desconexión por un período prolongado, es necesario contar con un mecanismo de sincronización que permita reconciliar el estado de los datos entre ambos sitios una vez que la conexión se restablece, evitando así inconsistencias derivadas de las modificaciones realizadas durante el período de desconexión.
3. Ayuda a proteger contra la pérdida de datos en caso de fallo de hardware o de red en uno de los sitios [105].

3.2.5. Fragmentación y tipos de niveles de transparencia de una base de datos distribuida.

La fragmentación en una base de datos distribuida se refiere a la división de la base de datos en partes más pequeñas llamadas *shards*, que pueden ser gestionados de manera independiente y almacenados en diferentes nodos del sistema [106]. La fragmentación puede mejorar el rendimiento de la base de datos al distribuir la carga de trabajo y el almacenamiento de datos entre múltiples nodos.

Existen tres tipos principales de fragmentación: horizontal, vertical e híbrida. Estos pueden aplicarse tanto a sistemas DBMS relacionales como no relacionales, a pesar de que la lógica de su aplicación es distinta.

En un DBMS relacional, la fragmentación horizontal consiste en dividir los datos por registros enteros, almacenando diferentes subconjuntos de registros en distintos nodos. Esta distribución introduce un grado de partición en el modelo de datos, ya que al dispersar los registros se pierden las garantías relacionales nativas (como la integridad referencial o la eficiencia de los joins) entre tablas que no residen en el mismo nodo o *shard*. En un DBMS no relacional la fragmentación horizontal se conoce como *sharding* [96], la cual consiste en distribuir los datos a lo largo de los nodos en un clúster (por ejemplo, para particionar una colección de documentos JSON, se tendría que distribuir a lo largo de los nodos los documentos pertenecientes a esa colección). Cabe destacar que el *sharding* tiene como consecuencia un grado de partición en el modelo de datos, pues deja de haber garantías relacionales entre tablas relacionadas que no estén alojadas en el mismo *shard*.

En un DBMS relacional, la fragmentación vertical divide los datos por atributos, manteniendo diferentes subconjuntos de registros con ciertos atributos en nodos separados. En un DBMS no relacional se puede aplicar este tipo de fragmentación a través de la desnormalización [107]. Por ejemplo, en lugar de tener todos los atributos en un solo documento JSON, en un grafo o en pares llave-valor, se puede dividir los atributos en múltiples documentos, colecciones y subgrafos.

La fragmentación híbrida en un DBMS relacional combina ambos enfoques, dividiendo los datos tanto por registro completo, como por registro parcial. Mientras que en un DBMS no relacional se involucra la combinación de sharding para distribuir los datos entre diferentes nodos y la desnormalización para separar diferentes partes de esos datos en colecciones o grafos distintos.

3.2.6. Niveles de transparencia en una base de datos distribuida.

Los niveles de transparencia en una base de datos distribuida son cruciales para ocultar los detalles complejos de la fragmentación y otros aspectos técnicos de los usuarios finales y desarrolladores [86]. Los niveles de transparencia son:

Transparencia de distribución: Asegura que los usuarios puedan realizar consultas y transacciones sin tener que saber en qué fragmento o la ubicación física en donde se encuentran los datos específicos [86]. Esto simplifica la interacción con el sistema, ya que los usuarios trabajan con una única base de datos lógica, mientras que el sistema maneja internamente la distribución y acceso a los fragmentos.

Transparencia de transacción: Asegura la integridad y el estado consistente de los datos, en especial cuando se ejecutan diversas sentencias de manipulación en varios sitios. Para ello los MMDBMS implementan algoritmos o protocolos de seguridad, por ejemplo, el protocolo *two phase commit* (2PC) [108].

Transparencia de replicación: Implica que si un elemento o la base de datos completa están replicados, los procesos de sincronización y aseguramiento de la consistencia entre las réplicas se gestionan de manera automática, de forma que el usuario final no necesita preocuparse por estos detalles técnicos, ya que quedan ocultos tras el funcionamiento del sistema [86].

Transparencia de desempeño: Indica que cuando hay fragmentación, el sistema maneja la conversión de consultas sobre relaciones globales a consultas definidas sobre fragmentos [86].

Transparencia de falla: Permite que el sistema continúe operando si un nodo presenta algún error [86].

Transparencia de heterogeneidad: Permite la integración de otros manejadores diferentes a un esquema global común [86]. Un esquema global común [109] se refiere a una representación unificada y consistente de la estructura de la base de datos que abarca todos los nodos o sitios donde la base de datos está distribuida, permitiendo a los usuarios interactuar con ella como si fuera una única base de datos.

3.2.7. Transaccionalidad en los MMDBMS.

En el contexto de las bases de datos multimodelo (MMDBMS), las propiedades ACID garantizan la integridad de los datos, especialmente en sistemas distribuidos. Sin embargo, la forma en que estas propiedades se implementan varía significativamente entre cada manejador, dependiendo de su arquitectura, soporte para transacciones y mecanismos de replicación. A continuación, se describe qué tan bien se cubren en ArangoDB, OrientDB y Couchbase.

ArangoDB:

- La atomicidad se garantiza en transacciones locales. En entornos distribuidos, su alcance suele limitarse a las colecciones que no están fragmentadas [98].
- La consistencia se mantiene adecuadamente en operaciones locales. En clústeres distribuidos depende en gran medida del criterio de fragmentación y de la configuración de las claves asociadas [98].
- El aislamiento se proporciona de forma completa en el contexto local. En sistemas distribuidos puede verse comprometido cuando no existe un consenso global sobre el estado de la transacción [98].
- La persistencia se asegura mediante el almacenamiento persistente en disco y el uso de mecanismos de recuperación ante fallos [67].

ArangoDB no implementa el protocolo 2PC y limita las transacciones distribuidas a colecciones con la misma clave de fragmentación [98][110].

OrientDB:

- La atomicidad se encuentra soportada tanto en transacciones locales como en entornos distribuidos, siempre que el sistema cuente con mecanismos nativos para manejar ambas modalidades [63].
- La consistencia se mantiene mediante el uso de un sistema de quórum (mecanismo en el que una operación solo se confirma cuando una mayoría de nodos del clúster la aprueban) y técnicas de replicación síncrona, lo que permite conservar un estado uniforme entre las réplicas del sistema [101]
- El aislamiento ofrece distintos niveles configurables, entre los que destacan "Read Committed" y "Repeatable Read", los cuales pueden seleccionarse según los requisitos del entorno transaccional [75].
- La persistencia se garantiza mediante procesos de replicación y mecanismos de recuperación, diseñados para preservar los datos aún en presencia de fallos del sistema [101].

OrientDB no utiliza el protocolo 2PC tradicional, pero implementa un mecanismo de consenso distribuido basado en quorum.

Couchbase:

- La atomicidad se garantiza mediante transacciones distribuidas que pueden abarcar múltiples documentos, lo que permite preservar esta propiedad incluso en operaciones complejas [112].
- La consistencia se mantiene a través de un modelo de consistencia fuerte y el uso de aislamiento por instantáneas, lo que asegura una visión coherente de los datos durante la ejecución de la transacción [112].
- El aislamiento se proporciona mediante técnicas como MVCC y bloqueo optimista, las cuales permiten controlar el acceso concurrente a los datos sin afectar la integridad del sistema [112].
- La persistencia se asegura mediante la escritura distribuida en múltiples nodos y la confirmación de escritura, evitando la pérdida de información ante fallos del sistema [112].

Couchbase implementa una versión moderna del protocolo 2PC, integrada en su Transaction Manager, disponible en los SDK oficiales [97].

La Tabla 3 resume qué manejadores multimodelo (MMDBMS) aplican las propiedades ACID y si cuentan con soporte para el protocolo de confirmación en dos fases (2PC):

Tabla 3. MMDBMS que aplican propiedades ACID.

Manejador	Atomicidad	Consistencia	Aislamiento	Durabilidad	Soporte 2PC
ArangoDB	Parcialmente	Parcialmente	Parcialmente	Aplica	No aplica
OrientDB	Aplica	Aplica	Aplica	Aplica	No aplica
Couchbase	Aplica	Aplica	Aplica	Aplica	Aplica

3.2.8. Mecanismos de desempeño y consistencia en bases de datos distribuidas.

En los sistemas distribuidos, la caché y el modelo de consistencia influyen directamente en el equilibrio entre rendimiento y fiabilidad. No siempre se pueden optimizar ambos aspectos al mismo tiempo, por lo que cada decisión técnica implica un compromiso. Se le llama caché fría (*cold cache*) a la situación en que los datos aún no se han cargado o no se han consultado recientemente. En ese estado, las primeras solicitudes deben recuperar la información desde el almacenamiento en disco o desde otros nodos, lo que aumenta la latencia y el costo computacional. Cuando la caché ya está caliente (*warm cache*), las consultas acceden a datos almacenados en memoria y los tiempos de respuesta disminuyen de forma notable en contraposición a traerlos del disco [113][114].

4. Diseño de la propuesta de solución.

En esta sección se presenta KhaBench, un *benchmark* para la comparación de sistemas de bases de datos multimodelo en ambientes distribuidos.

4.1. Benchmark KhaBench.

KhaBench (el prefijo deriva de la raíz semítica **'Khā'** (*Especialidad*), haciendo referencia a un marco de evaluación diseñado especialmente para bases de datos) es el nombre del *benchmark* propuesto en el presente trabajo y surge con la idea de ser una herramienta de evaluación para diversos MMDBMS dentro de un ambiente distribuido, en el contexto de una red social de una aplicación de comercio electrónico. KhaBench presenta el diseño de una base de datos distribuida multimodelo, así como las consultas y transacciones a ejecutar para la evaluación de la consistencia, el uso de CPU y de memoria RAM dentro de un ambiente distribuido.

En la Figura 3 se ilustran, a manera de diagrama, las fases que componen KhaBench.

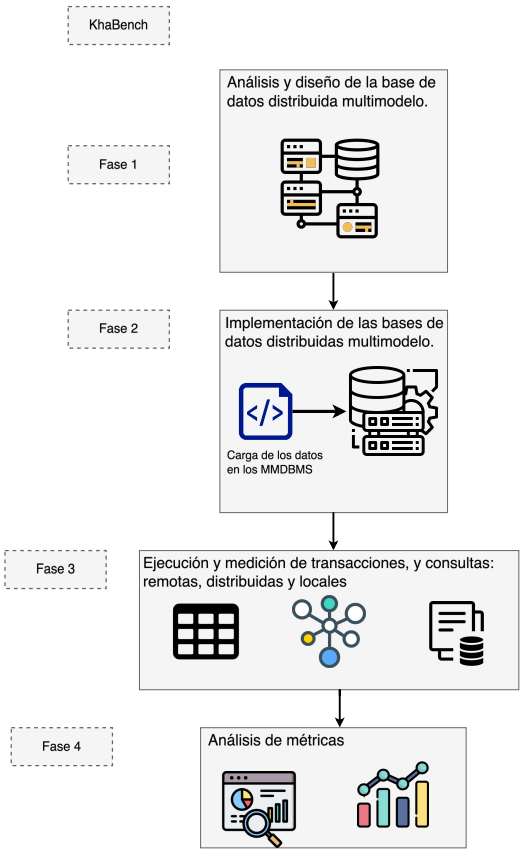


Fig. 3. Metodología de KhaBench.

La primera fase de KhaBench se enfoca en el análisis, diseño y creación de una base de datos distribuida en cada MMDBMS. Durante esta fase, se establece el caso de uso que se considerará para el diseño del esquema global multimodelo. Además, se definen las estrategias para la fragmentación de los datos (fragmentación horizontal) y los criterios que se utilizarán para distribuir los fragmentos entre los nodos.

La segunda fase de KhaBench consiste en la implementación de la base de datos distribuida multimodelo diseñada en la fase anterior. En esta etapa, se describen las características del conjunto de datos que serán cargados en la base de datos distribuida. Se instalan los MMDBMS en cada nodo y se cargan los fragmentos correspondientes. Este proceso se lleva a cabo mediante un script que carga el conjunto de datos de UniBench [5], disponible de forma abierta.

La tercera fase de KhaBench se centra en el diseño, ejecución y medición de las cargas de trabajo sobre la base de datos distribuida multimodelo previamente implementada en los MMDBMS.

La ejecución de consultas distribuidas permite evaluar cómo se comportan los MMDBMS con cargas de trabajo que involucran el llamado a nodos remotos.

Al concluir esta fase, se obtienen las métricas que ayudan a determinar el rendimiento de cada uno de los MMDBMS en las tareas ejecutadas. Estas métricas abarcan el uso de recursos (CPU y memoria RAM) y la consistencia.

En la cuarta y última fase, se analizan las métricas obtenidas de la fase anterior y se identifica qué MMDBMS ha demostrado el mejor rendimiento en general.

4.2. Primera fase: Análisis y diseño de la base de datos distribuida multimodelo.

Con el fin de poder evaluar un MMDBMS en un caso de estudio ampliamente representativo en bases de datos, se usa el caso de estudio de una red social de una aplicación de comercio electrónico, debido a la propia naturaleza del caso de uso en operar con diferentes modelos de datos.

4.2.1. Esquema global.

El esquema global (o simplemente el esquema de la base de datos multimodelo) se toma de M2Bench [4]. En este trabajo se desea usar un esquema bien definido como el de M2Bench [4] para poder dar un enfoque a los aspectos del *benchmark*, como la creación de los esquemas de fragmentación y las fórmulas para obtener las métricas.

El caso de uso abarca la definición de las siguientes entidades:

- **VENDOR** (vendor_id, country, industry)
- **CUSTOMER** (customer_id, first_name, last_name, gender, birthday, create_date, location_ip, browser_used, place)
- **PRODUCT** (product_id, title, price, img_url, sku, vendor_id)
- **ORDER** (order_id, customer_id, order_date, total_price, orderline) - FEEDBACK (product_id, customer_id, rate, review)
- **INVOICE** (order_id, customer_id, order_date, total_price, product)
- **PERSON** (person_id, first_name, last_name, gender, birthday, create_date, location_ip, browser_used, place)
- **TAG** (tag_id, title)
- **POST** (post_id, create_date, image_file, location_ip, browser_used, content, length)

Nota: La diferencia entre Person y Customer es el modelo de datos (Customer es relacional y Person es grafo), sin embargo, ambas entidades comparten los mismos atributos.

La Figura 4 presenta el esquema global de la base de datos distribuida multimodelo. Para su correcta interpretación, se deben considerar los siguientes elementos:

- El color del encabezado de cada entidad indica el MMDBMS en el que se almacena: el color naranja corresponde a OrientDB, el verde a ArangoDB y el rojo a Couchbase. En los casos donde una entidad es compartida entre dos MMDBMS (como ORDER, que se fragmenta entre OrientDB y ArangoDB), el encabezado muestra ambos colores.
- La parte superior de la figura funciona como leyenda, indicando qué modelo de dato (relacional, grafo, llave-valor, JSON o XML) es soportado por cada MMDBMS dentro de este esquema.
- Las relaciones entre entidades siguen las convenciones estándar de un diagrama entidad-relación: las líneas con doble barra indican cardinalidad obligatoria y las flechas indican dirección de la relación. Las relaciones de grafo (como *Knows*, *has interest*, *create* y *has*) conectan las entidades del modelo de grafo almacenadas en ArangoDB.

4.2.2. Esquema multimodelo.

Dado que se van a emplear diferentes modelos de datos en cada MMDBMS, el esquema multimodelo propuesto se muestra en la Figura 4.

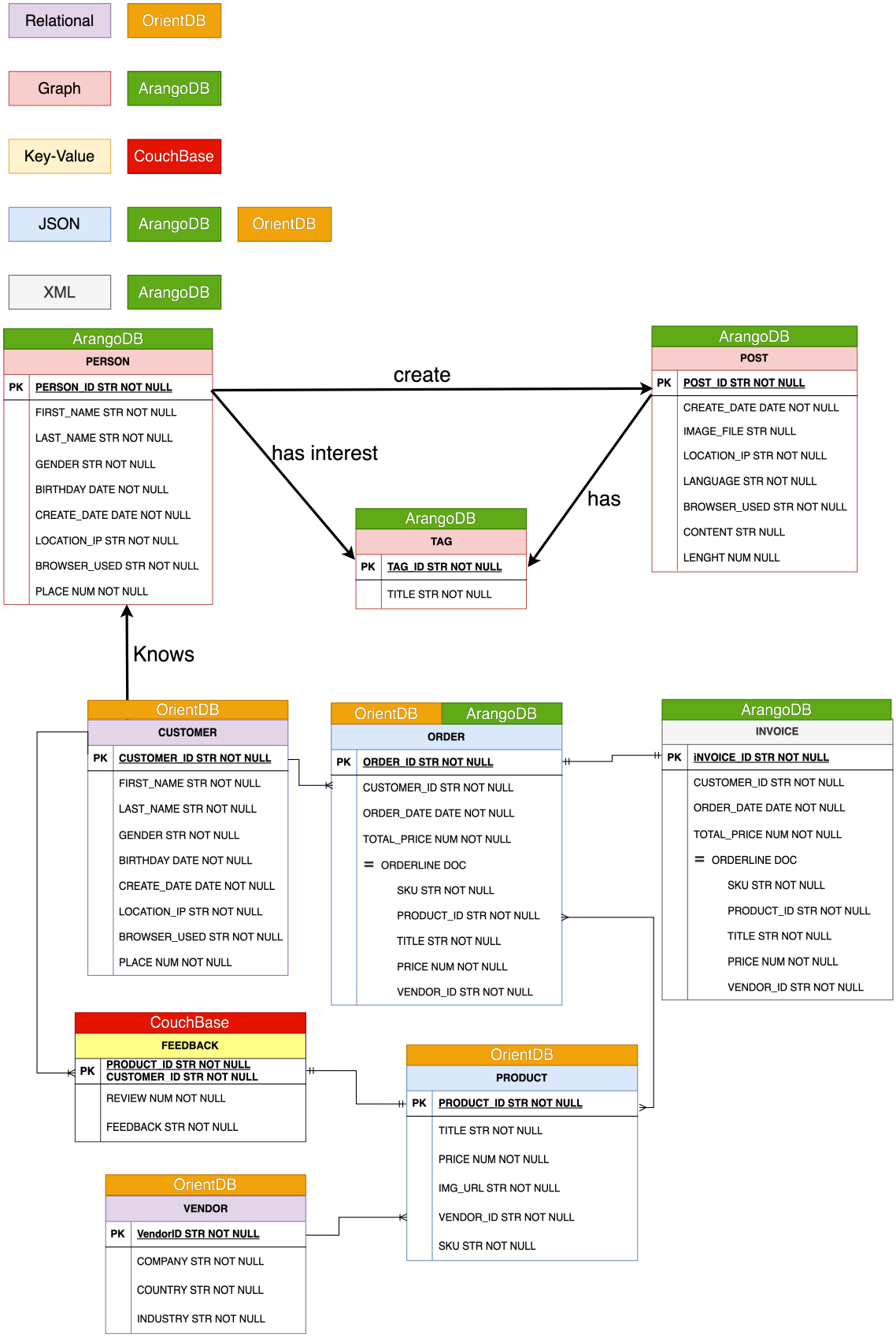


Fig. 4. Esquema global de cada base de datos distribuida multimodelo.

La parte relacional de la base de datos distribuida multimodelo está compuesta por las entidades CUSTOMER y VENDOR, que, dentro del contexto de una red social de una aplicación de comercio electrónico, tienen como objetivo gestionar la información de los clientes y proveedores que venden en la aplicación, debido a que el modelo relacional permite conservar la integridad de los datos y los atributos que pueden componer a un cliente y a un producto están bien definidos.

La parte documental de la base de datos distribuida multimodelo está compuesta por las entidades ORDER, PRODUCT, FEEDBACK e INVOICE. Dentro del contexto de la aplicación, su objetivo es administrar la parte transaccional de los productos que los clientes compran y las reseñas que emiten sobre dichos productos. Debido a que las características y el comportamiento que pueden presentar estas entidades varían, su esquema no es completamente estructurado.

La parte definida por grafos de la base de datos está compuesta por las entidades PERSON, POST y TAG, que, dentro del contexto de la aplicación, tienen como objetivo representar y analizar las relaciones y conexiones entre usuarios y el contenido que consumen en la red social, lo cual depende en gran medida de los productos que los usuarios (o clientes) adquieren.

4.2.3. Diseño de la base de datos distribuida multimodelo.

El diseño de la base de datos distribuida se genera a partir del esquema multimodelo global. A pesar de que para los modelos de datos de documento y grafos no exista un equivalente exacto al álgebra relacional, en estos modelos sí están definidas las operaciones de proyección y selección, las cuales son útiles para definir cómo se va a fragmentar la base de datos.

Las reglas de negocio empleadas para fragmentar los datos son las siguientes:

- Los clientes deben ser particionados de acuerdo con su localización geográfica, para mejorar temas de logística y entrega de productos. Esto mismo aplica para perfiles de las personas que viven en el sur.

  Nodo 1 OrientDB:

  $$\boldsymbol{CUSTOMER_{SOUTH}} = \sigma_{place \geq 1 \; and \; place \leq 500}(\boldsymbol{CUSTOMER})$$

  Nodo 2 OrientDB:

  $$\boldsymbol{CUSTOMER_{NORTH}} = \sigma_{place > 500 \; and \; place \leq 1000}(\boldsymbol{CUSTOMER})$$

  Nodo 3 OrientDB:

  $$\boldsymbol{CUSTOMER_{CENTER}} = \sigma_{place > 1000}(\boldsymbol{CUSTOMER})$$

  Nodo 4 ArangoDB:

$$\boldsymbol{PERSON_{SOUTH}} = \sigma_{place \geq 1 \; and \; place \leq 500}(\boldsymbol{PERSON})$$

- Los productos deben ser organizados de acuerdo con su precio, para distinguir cuáles son los productos costosos.

  Nodo 2 OrientDB:

$$\boldsymbol{PRODUCT_{EXPENSIVE}} = \sigma_{price \geq 250}(\boldsymbol{PRODUCT})$$

- Se desean clasificar aquellas órdenes que fueron hechas antes y después de la pandemia.

  Nodo 1 OrientDB:

$$\boldsymbol{ORDERS_{PRE\ PANDEMIC}} = \sigma_{ORDER\ DATE \leq "2020-03-11"}(\boldsymbol{ORDERS})$$

  Nodo 4 ArangoDB:

$$\boldsymbol{ORDERS_{POST\ PANDEMIC}} = \sigma_{ORDER\ DATE > "2020-03-11"}(\boldsymbol{ORDERS})$$

Las entidades restantes que no se fragmentan, se cargarán en los siguientes nodos y MMDBMS:

- Nodo 2 Couchbase - FEEDBACK
- Nodo 4 OrientDB - VENDOR e INVOICE
- Nodo 1 ArangoDB - PERSON, POST y TAG

4.3. Segunda fase: Implementación de las bases de datos distribuidas multimodelo.

Una vez creadas las bases de datos distribuidas multimodelo, se utilizará el conjunto de datos proporcionado por Unibench en [5], ya que es de libre uso y se acopla perfectamente con el caso de uso.

Las características de este conjunto de datos son las siguientes:

- La entidad de clientes y usuarios cuenta con 9 950 registros, los cuales forman un total de 1 MB.
- La entidad de productos cuenta con 9 962 registros, los cuales forman un total de 1.4 MB.
- La entidad de marca cuenta con 9 694 registros, los cuales forman un total de 250 KB.
- La entidad de órdenes cuenta con 142 257 registros, los cuales forman un total de 119.2 MB.

- La entidad de reseñas cuenta con 142 257 registros, los cuales forman un total de 221.5 MB.
- La entidad de posteos cuenta con 150 000 registros, los cuales forman un total de 221.5 MB.

4.3.1. Entornos nativos en Raspberry Pi 5.

Cada nodo del sistema distribuido corresponde a una Raspberry Pi 5 en la que se instaló de forma nativa el software del MMDBMS correspondiente.

Cada Raspberry Pi 5 tiene la misma configuración de recursos: 4 GB de RAM, 2 núcleos de CPU y 10 GB de almacenamiento, lo que garantiza condiciones uniformes de evaluación entre los MMDBMS.

Una vez configurados los entornos en cada dispositivo, se utilizó un script orquestador para cargar los datos en los respectivos nodos. Este script accede al conjunto de datos UniBench [5] y distribuye los fragmentos entre los dispositivos físicos según la estrategia de particionamiento previamente definida.

Si bien los MMDBMS evaluados ofrecen distintos niveles de transparencia de distribución (como se describió en la sección 3.2.5), se optó por un script orquestador externo debido a que ArangoDB y OrientDB no implementan el protocolo 2PC de forma nativa, lo que limita las garantías de consistencia en operaciones distribuidas entre distintos motores.

El script permite controlar de manera uniforme la distribución de los fragmentos bajo las mismas condiciones operativas para los tres MMDBMS, asegurando así una comparación justa entre ellos.

4.4. Tercera Fase: Ejecución y medición de consultas y transacciones distribuidas.

Una vez implementadas las bases de datos en los distintos MMDBMS a lo largo de los nodos, se ejecutan las consultas y transacciones distribuidas en cada MMDBMS para identificar cuál presenta un mejor rendimiento.

Las cargas de trabajo propuestas constan de siete tareas, descritas en la Tabla 4. Cada tarea fue diseñada para aprovechar la estrategia de fragmentación definida previamente, con el fin de identificar qué MMDBMS ofrece el mejor desempeño en cada escenario. Para cada tarea se especifica su descripción, el nodo en que se ejecuta, el tipo de operación (consulta o transacción) y el MMDBMS involucrado.

Tabla 4. Descripción de las Cargas de Trabajo.

ID	Nombre de la tarea	Descripción	MMDBMS	Tipo
C1	Validar órdenes post-pandemia	Buscar **órdenes** realizadas después de la pandemia por **clientes** ubicados en el norte.	ArangoDB, OrientDB	Consulta Distribuida
T2	Modificación de precio de producto	Modificar el precio (aumento del 10 %) de los **productos** costosos. Además, actualizar el total de las **órdenes** post-pandemia al aumento.	ArangoDB, OrientDB	Transacción Distribuida
T3	Eliminación de órdenes de pre-pandemia	Eliminar **órdenes** generadas antes de la pandemia, pero conservar las **facturas** relacionadas para futuras aclaraciones fiscales.	ArangoDB, OrientDB	Transacción Distribuida
T4	Modificación de Posts	Modificar los **posts** y los **tags** que hayan creado las **personas** del sur (has_creator y has_tag).	ArangoDB	Transacción Local
T5	Modificación de feedbacks	Modificar la calificación y reseñas de los feedbacks que tienen una calificación menor a 3.	Couchbase	Transacción Remota
C6	Consultar feedbacks modificados	Mostrar el contenido modificado de los feedbacks de la transacción anterior.	Couchbase	Consulta Remota
C7	Validar productos con orden alfabético	Mostrar los **productos** cuyo nombre esté entre la M y la Z.	OrientDB	Consulta Local

4.4.1. Plan de experimentación.

Con las transacciones y consultas definidas (T2–T5, C1, C6 y C7) se diseñó un plan de experimentación orientado a medir tres aspectos principales del desempeño de los motores evaluados: el consumo de CPU, el uso de memoria RAM y la consistencia

observable tipo Read Your Own Write (RYOW), es decir, el tiempo que tarda una escritura en hacerse visible para lecturas posteriores del mismo cliente.

Para cada tarea se ejecutarán distintos escenarios de prueba que combinan tres variables de control:

- Tamaño del dataset, con tres escalas: S (10^4 registros), M (10^5 registros) y L (10^6 registros), que permiten observar el impacto del volumen de datos.
- Nivel de concurrencia, con cargas de 1, 4, 16 y 32 hilos en paralelo, que simulan diferentes números de usuarios simultáneos y revelan la capacidad de cada motor para escalar horizontalmente.
- Adicionalmente, se contemplan dos condiciones del sistema: *warmup*, cuando la caché ya está precalentada y el sistema refleja un estado estable de operación, y *cold cache*, cuando el sistema está sin datos precargados, lo que permite medir el impacto del primer acceso a disco, construcción de índices en memoria y propagación de metadatos.

5. Análisis de resultados experimentales.

Esta sección tiene como objetivo evaluar el comportamiento de los motores multimodelo bajo distintas condiciones de carga y volumen de datos, a partir de las cargas de trabajo definidas en el plan de experimentación. Mediante el monitoreo de CPU y memoria RAM se analiza la forma en que cada sistema gestiona sus recursos y mantiene la consistencia observable (RYOW) durante la ejecución, considerando distintos tamaños del conjunto de datos y diferentes estados de caché, tal como se describió en las secciones 4.4.1 y 3.2.8.

Adicionalmente, este apartado presenta e interpreta los resultados obtenidos, lo que permite comparar la eficiencia, la escalabilidad y la estabilidad de cada motor, tanto en el desempeño promedio como en los casos de mayor exigencia reflejados en los percentiles de latencia (p50, p95 y p99).

5.1. Cuarta fase: análisis de métricas.

A continuación, se muestra el análisis respectivo de cada transacción y consulta, acompañados de un diagrama de secuencia.

5.1.1. Consulta Distribuida C1.

La consulta C1 valida cuántas órdenes post-pandemia están asociadas a los clientes de cada región (norte, centro y sur). Para ello, obtiene primero desde OrientDB los identificadores de los clientes fragmentados por región y, a continuación, utiliza ArangoDB para contabilizar las órdenes correspondientes a esos clientes.

De este modo, la consulta integra información proveniente de dos motores distintos, lo que permite obtener una visión comparativa de la distribución de órdenes por región y evidencia el comportamiento de cada base de datos al operar de forma complementaria en un entorno multimodelo y distribuido.

La Figura 5 presenta el diagrama de secuencia correspondiente, donde se ilustran las operaciones ejecutadas durante la consulta y los datos intercambiados entre ambos sistemas: la obtención de identificadores desde OrientDB y el conteo de órdenes en ArangoDB.

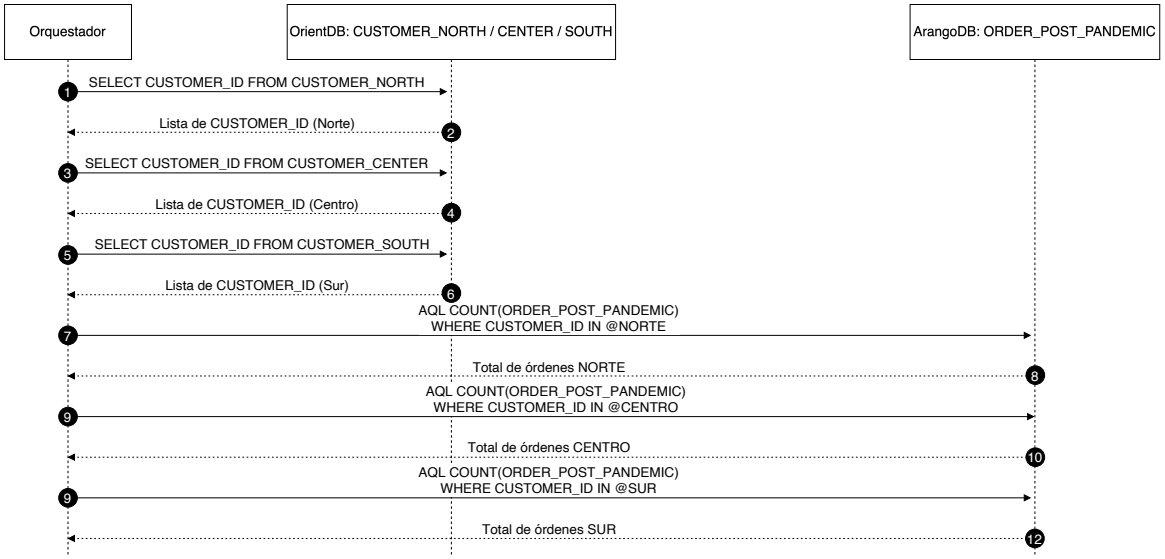
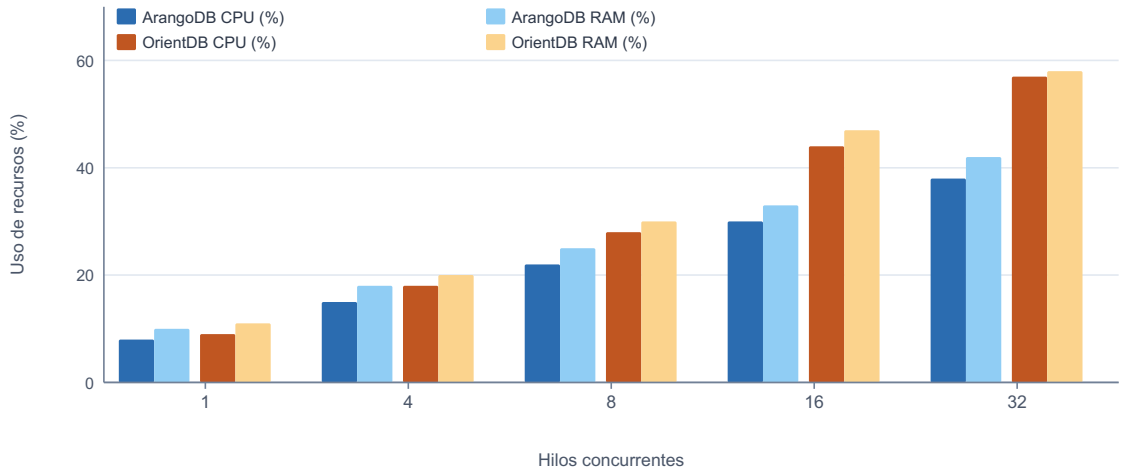


Fig. 5. Diagrama de secuencia de consulta C1.

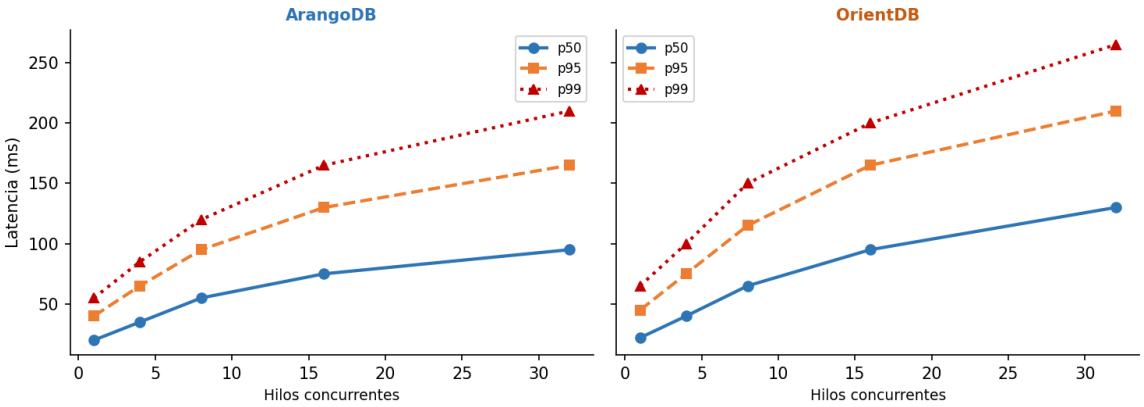
Para analizar las métricas obtenidas en esta consulta se emplean las Gráficas 1 y 2. En la Gráfica 1, el eje X muestra el número de hilos concurrentes utilizados durante la ejecución (1, 4, 8, 16 y 32), mientras que el eje Y representa el porcentaje promedio de uso de CPU y memoria RAM registrado en cada motor. La Gráfica 2 utiliza igualmente el número de hilos en el eje X; el eje Y muestra los tiempos de respuesta medidos en milisegundos, expresados a través de los percentiles p50, p95 y p99, que reflejan el comportamiento típico, el alto y el extremo de las demoras observadas durante la ejecución concurrente.

En la Gráfica 1 se aprecia que ambos motores incrementan progresivamente el uso de CPU y RAM conforme aumenta la concurrencia, lo cual es esperable en escenarios de carga. Sin embargo, ArangoDB mantiene un escalamiento más moderado, mientras que OrientDB muestra un crecimiento más pronunciado a partir de 16 y 32 hilos, con un consumo de CPU y memoria superior al de ArangoDB.



Gráfica 1. Consumo de recursos en OrientDB vs ArangoDB (Consulta C1).

En la Gráfica 2 se aprecia que bajo cargas bajas (1 y 4 hilos) los tiempos de respuesta en los percentiles p50, p95 y p99 son similares en ambos motores. No obstante, conforme crece la concurrencia las diferencias se vuelven notorias: ArangoDB sostiene latencias más bajas y estables, mientras que OrientDB experimenta una degradación más rápida, alcanzando valores de p99 superiores a 250 ms en escenarios de 32 hilos. Los resultados muestran que, aunque ambos motores son adecuados en escenarios pequeños, ArangoDB ofrece mejores resultados bajo alta concurrencia.



Gráfica 2. Latencia de Consulta C1 en ArangoDB vs OrientDB.

5.1.2. Transacción Distribuida T2.

En la Transacción T2 (actualización de precios en productos y órdenes) se ejecuta una operación coordinada entre OrientDB y ArangoDB, donde cada motor interviene sobre distintos modelos de datos.

T2 es una transacción multimodelo, ya que opera sobre estructuras documentales diferentes (productos individuales frente a órdenes compuestas), y multi-motor, al coordinar la ejecución entre dos manejadores independientes. La sincronización entre ambos incide directamente en la consistencia RYOW: la actualización del precio está sujeta a las reglas de escritura y lectura de OrientDB, mientras que la propagación del nuevo total depende del nivel de durabilidad (writeConcern) configurado en ArangoDB.

La Figura 6 muestra un diagrama de secuencia que ilustra las acciones realizadas en T2.

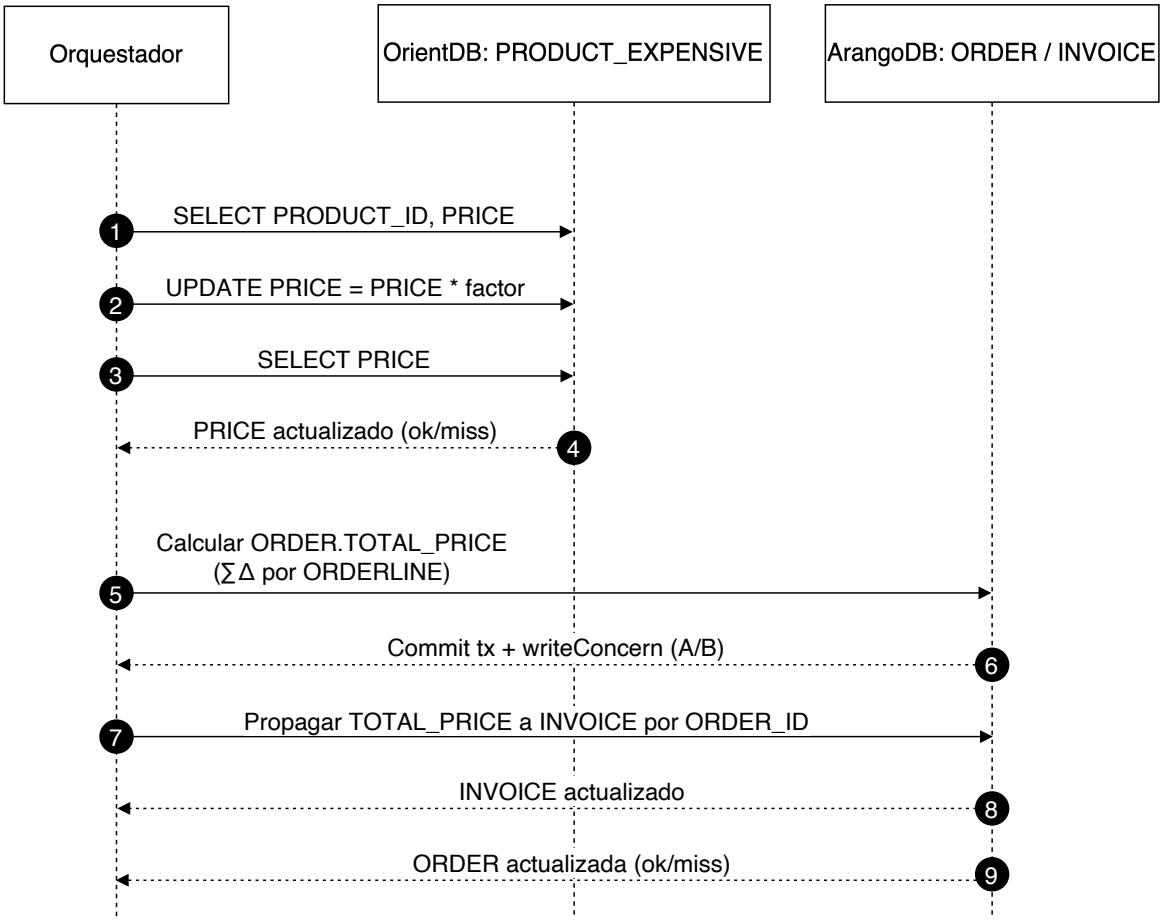
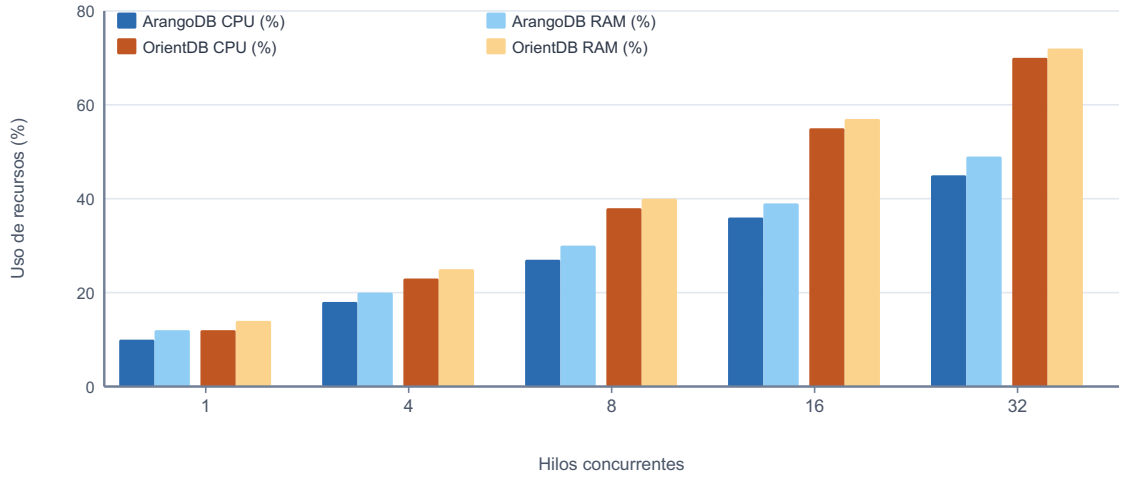


Fig. 6. Diagrama de secuencia de transacción T2.

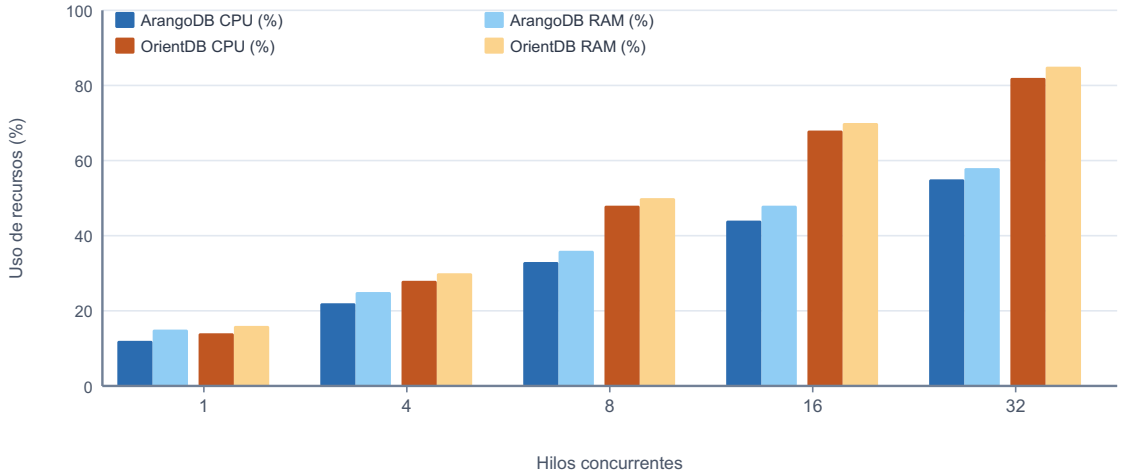
Las Gráficas 3 a 8 presentan los resultados de la Transacción T2 bajo diferentes condiciones de carga, tamaño del conjunto de datos, número de hilos y estado de caché. Las Gráficas 3 y 4 corresponden a ejecuciones con caché caliente (*warmup*), con índices y estructuras de datos ya cargadas en memoria, mientras que la Gráfica 5 refleja un escenario de caché fría (*cold cache*). En las Gráficas 6 a 8 se evalúa la consistencia RYOW con la latencia en percentiles p50, p95 y p99.

En la Gráfica 3 se observa que, al aumentar la concurrencia en el conjunto de datos S, ambos motores elevan su uso de recursos. OrientDB mantiene un consumo sistemáticamente mayor, lo que indica menor eficiencia al crecer la carga de trabajo.



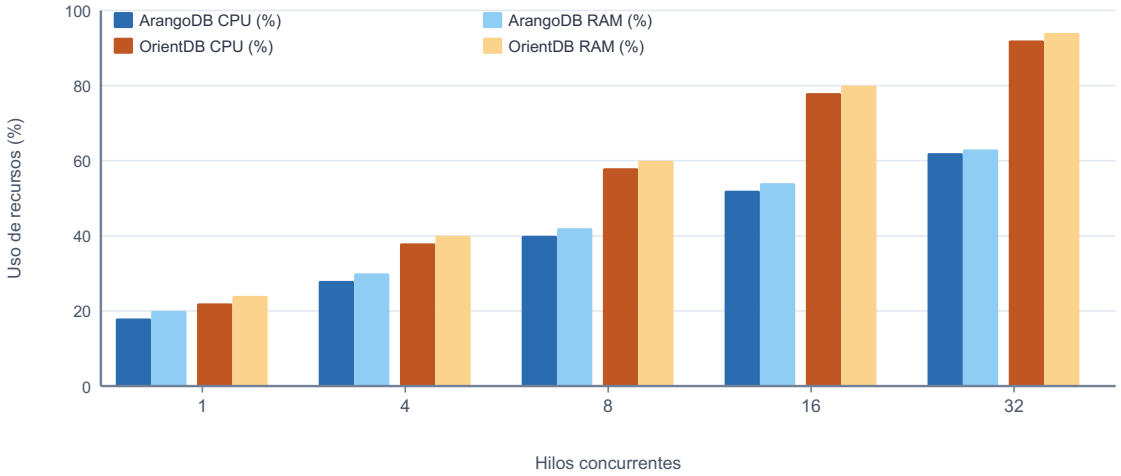
Gráfica 3. Consumo de recursos en OrientDB vs ArangoDB (Transacción T2, Conjunto de datos S).

La Gráfica 4 muestra el comportamiento durante la ejecución con el conjunto de datos M (10^5 registros). OrientDB presenta un incremento más pronunciado en el uso de CPU y memoria conforme aumenta el número de hilos. ArangoDB, por su parte, muestra un aumento más gradual y estable, lo que refleja una mejor capacidad de adaptación y control del consumo bajo una carga equivalente.



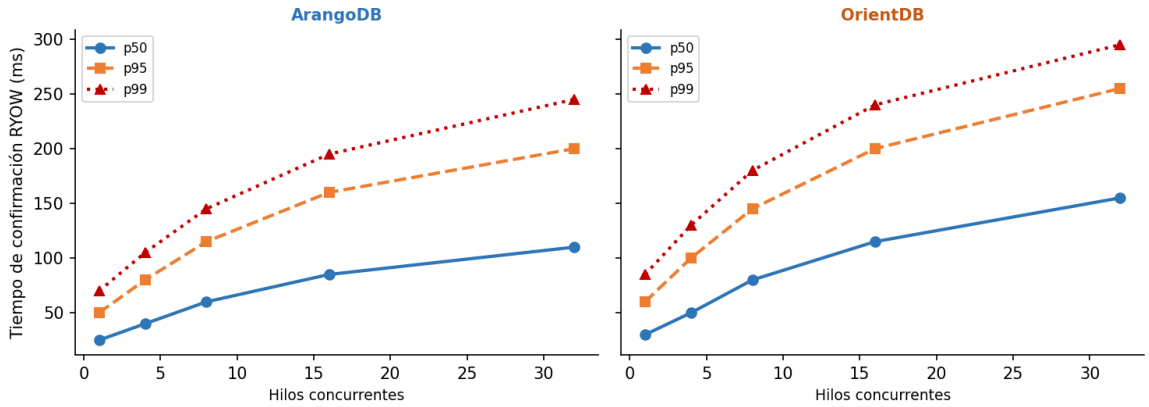
Gráfica 4. Consumo de recursos en OrientDB vs ArangoDB (Transacción T2, Conjunto de datos M).

La Gráfica 5 muestra el sobrecosto inicial asociado al escenario de caché fría, donde ambos motores incrementan temporalmente su consumo de CPU y RAM al iniciar las operaciones sin datos precargados; ArangoDB contiene mejor este pico.



Gráfica 5. Consumo de recursos en OrientDB vs ArangoDB (Transacción T2, Conjunto de datos M, Caché fría).

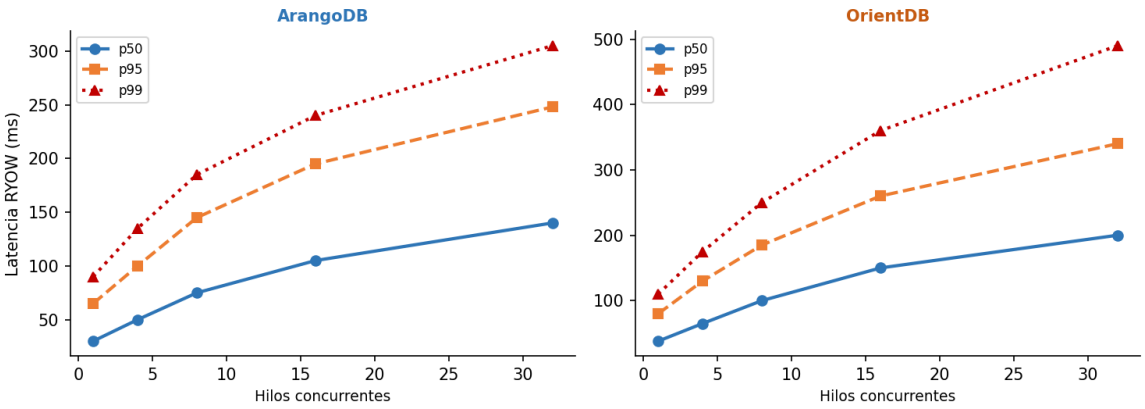
La Gráfica 6 evidencia que, bajo cargas ligeras, los valores p50 permanecen bajos en ambos motores, pero los percentiles p95 y p99 aumentan con mayor dispersión en OrientDB. Esto hace menos predecible la visibilidad inmediata de los cambios en OrientDB frente a ArangoDB.



Gráfica 6. Consistencia RYOW en OrientDB vs ArangoDB (Transacción T2, Conjunto de datos M).

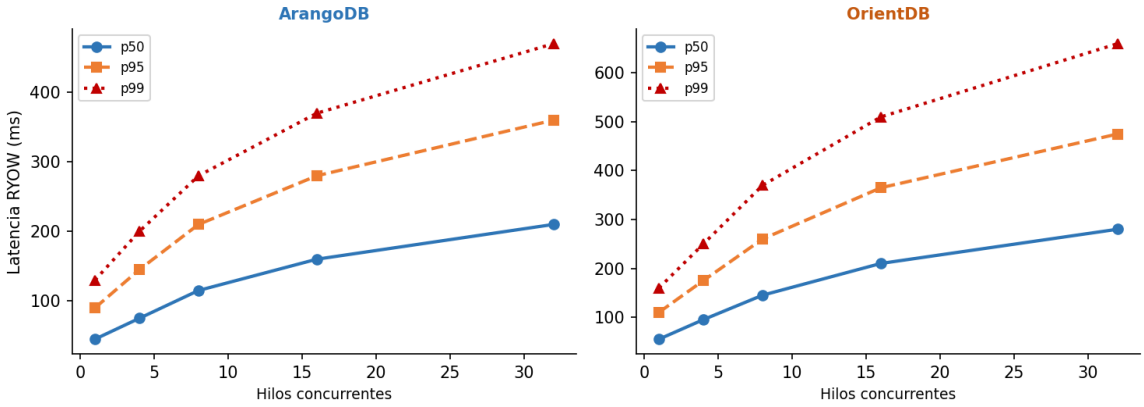
La Gráfica 7 confirma la brecha entre ambos motores: con 32 hilos, el percentil p99 en OrientDB alcanza valores cercanos a 600 ms, mientras que en ArangoDB se mantiene más estable y predecible. Este incremento en OrientDB se asocia con una mayor contención de locks y la retención temporal de cambios no confirmados.

ArangoDB, en cambio, utiliza transacciones distribuidas con confirmaciones más eficientes y control explícito de durabilidad.



Gráfica 7. Consistencia RYOW en OrientDB vs ArangoDB (Transacción T2, Conjunto de datos M).

La Gráfica 8 muestra que, sin calentamiento de caché, los percentiles de latencia se deterioran en ambos motores, aunque ArangoDB controla mejor la dispersión de valores. Este resultado se explica por la diferencia en el modelo de datos de cada motor: OrientDB, con mayor dependencia de estructuras enlazadas, requiere más lecturas internas tras cada escritura.



Gráfica 8. Consistencia RYOW en OrientDB vs ArangoDB (Transacción T2, Conjunto de datos M, Caché fría).

Al ser una operación distribuida, T2 implica la coordinación simultánea entre OrientDB y ArangoDB mediante un script orquestador. Los resultados muestran que ArangoDB logra manejar mejor esta complejidad, manteniendo un consumo equilibrado de recursos y una latencia más estable. OrientDB, en contraste, presentó mayor dispersión, lo que pudiera deberse a una mayor sensibilidad a la sincronización inter-nodo en la propagación de escrituras.

5.1.3. Transacción Distribuida T3.

La Transacción T3 es una operación distribuida que involucra la coordinación entre diferentes motores ubicados en nodos distintos. Primero se eliminan órdenes en OrientDB y, a continuación, el cambio se propaga en ArangoDB mediante la anulación del campo ORDER_ID dentro de los documentos de factura (INVOICE). El borrado completo de registros en OrientDB impone una mayor carga sobre CPU y memoria, generando presión de CPU. La actualización parcial de un atributo en ArangoDB resulta menos costosa gracias a sus transacciones ACID, que permiten modificar múltiples documentos manteniendo la consistencia sin necesidad de reconstruir índices completos.

La Figura 7 muestra un diagrama de secuencia que detalla las acciones realizadas en T3.

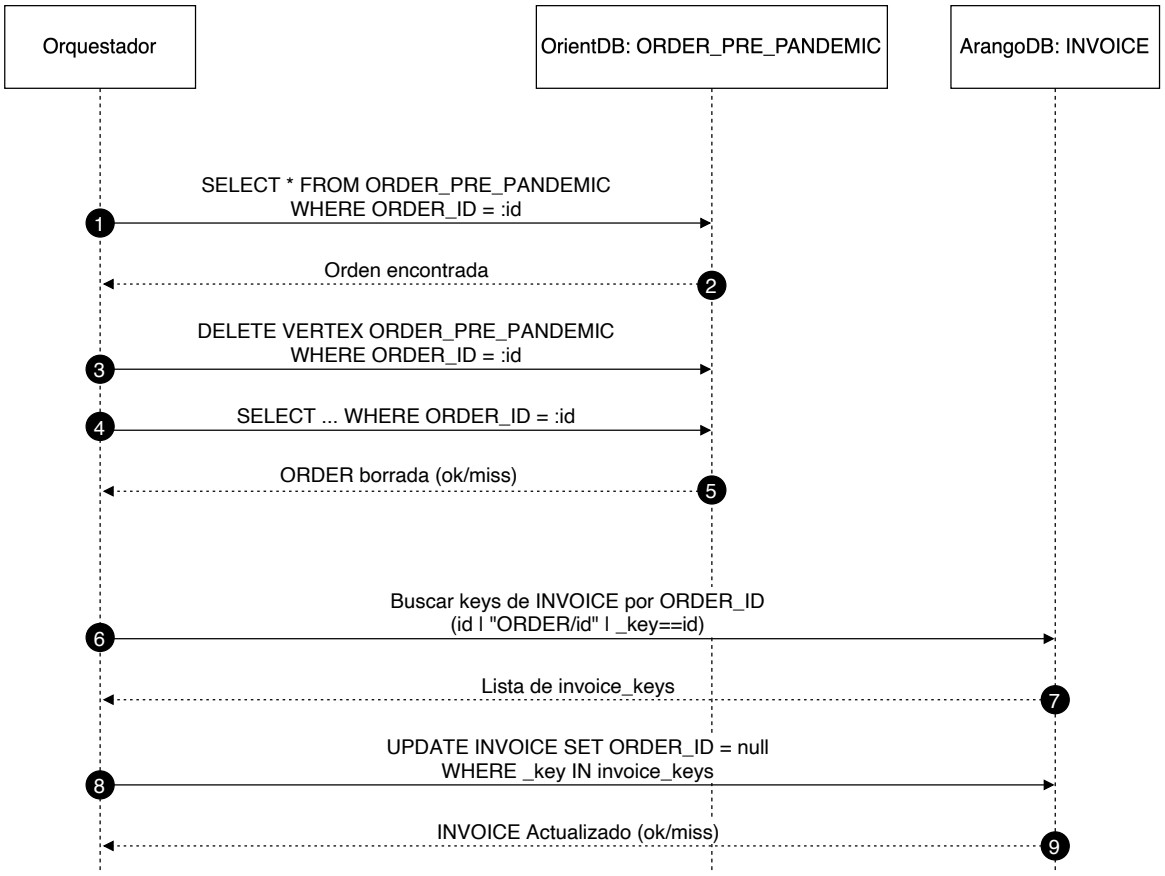
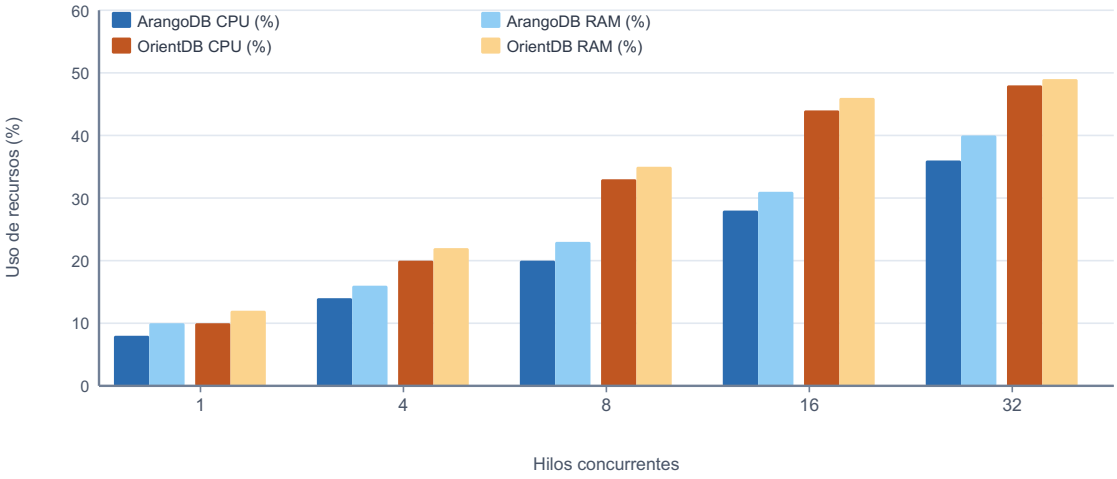


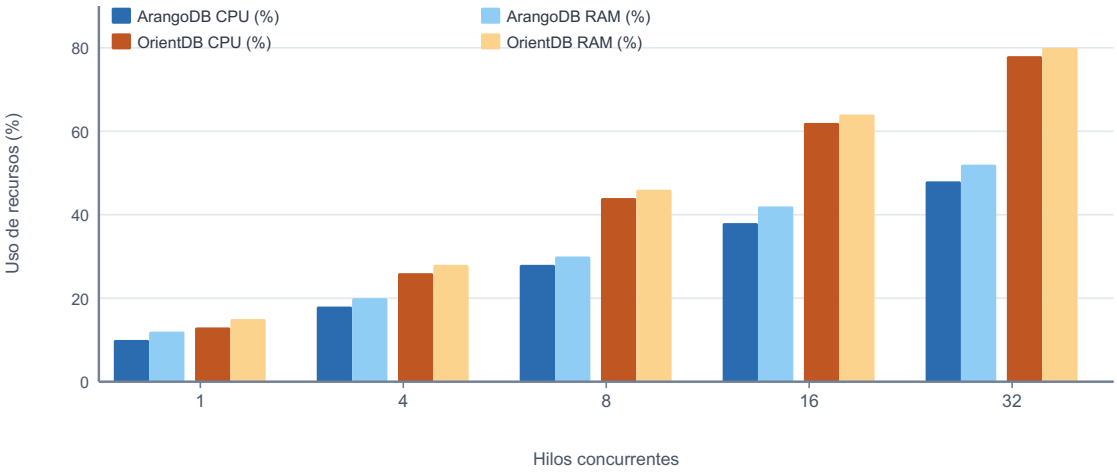
Fig. 6. Diagrama de secuencia de transacción T3.

En la Gráfica 9 se observa que el incremento de hilos eleva el consumo de ambos recursos, aunque OrientDB aumenta su demanda en mayor proporción. ArangoDB mantiene una pendiente de crecimiento más gradual, consistente con su arquitectura transaccional para operaciones documentales.



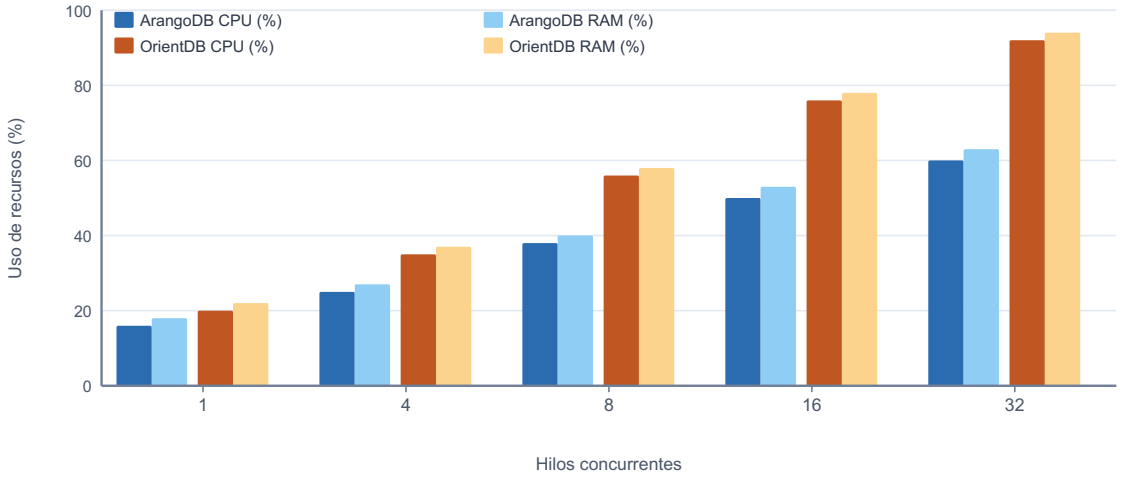
Gráfica 9. Consumo de recursos en OrientDB vs ArangoDB (Transacción T3, conjunto de datos S).

La Gráfica 10 refuerza esta tendencia: OrientDB presenta picos de memoria más altos en escenarios de 16 a 32 hilos, reflejo de la acumulación de estructuras intermedias durante los borrados masivos, mientras que ArangoDB conserva un uso de memoria más estable al modificar únicamente campos específicos.



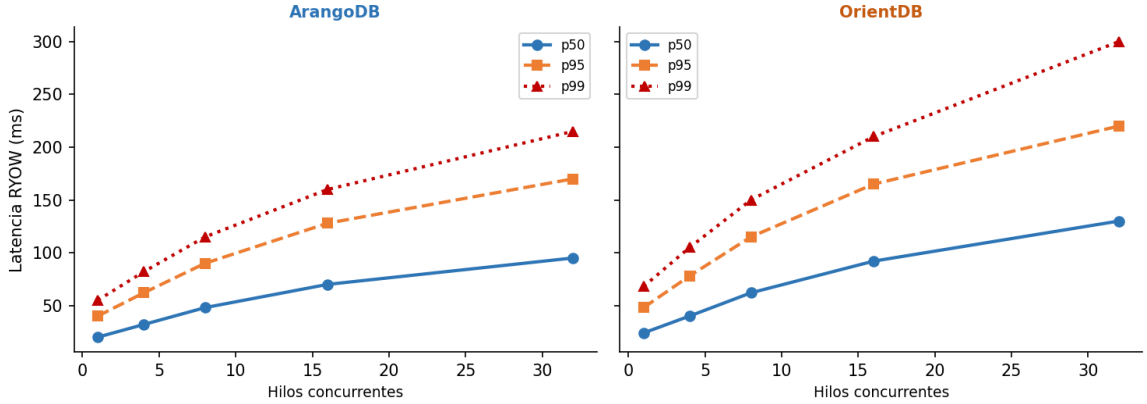
Gráfica 10. Consumo de recursos en OrientDB vs ArangoDB (Transacción T3, conjunto de datos M).

La Gráfica 11 introduce la condición de caché fría, donde ambos motores enfrentan un sobrecosto inicial por la falta de datos precargados. ArangoDB controla mejor el consumo de recursos durante el arranque.



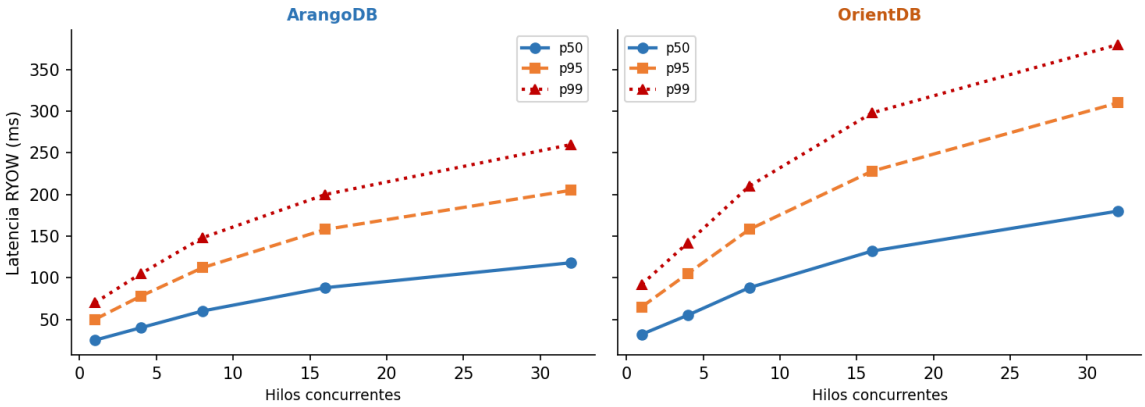
Gráfica 11. Consumo de recursos en OrientDB vs ArangoDB (Transacción T3, Conjunto de datos M, Caché fría).

La Gráfica 12 muestra que bajo cargas ligeras (1 a 8 hilos) los valores de latencia p50, p95 y p99 se mantienen relativamente bajos y cercanos entre ambos motores. A partir de 16 hilos se produce un aumento considerable, lo que indica que en escenarios de baja concurrencia ambos motores logran una visibilidad rápida y consistente de las escrituras.



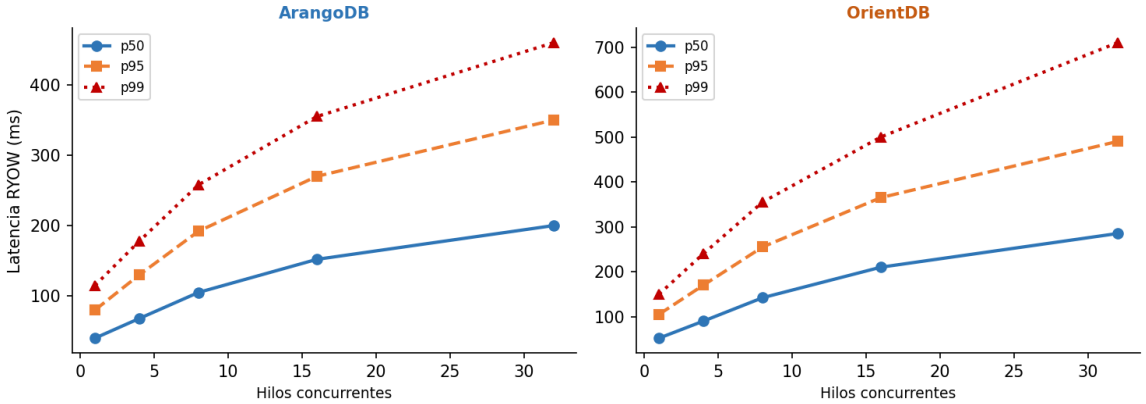
Gráfica 12. Consistencia RYOW en OrientDB vs ArangoDB (Transacción T3, Conjunto de datos S).

La Gráfica 13 muestra que a medida que aumenta la concurrencia y el tamaño del conjunto de datos, OrientDB presenta un incremento más pronunciado en los percentiles altos (p95 y p99). ArangoDB conserva un crecimiento más gradual y estable.



Gráfica 13. Consistencia RYOW en OrientDB vs ArangoDB (Transacción T3, Conjunto de datos M).

La Gráfica 14 presenta el escenario de caché fría: los tiempos de respuesta aumentan notablemente, en especial en OrientDB, donde el p99 supera los 700 ms. ArangoDB logra mantener un comportamiento más controlado y predecible, con menor dispersión entre ejecuciones.



Gráfica 14. Consistencia RYOW en OrientDB vs ArangoDB (Transacción T3, Conjunto de datos M, Caché fría).

La naturaleza distribuida de T3 generó un sobrecosto de coordinación que afectó directamente la latencia y el uso de recursos. OrientDB experimenta una mayor presión en CPU y memoria bajo alta concurrencia y caché fría, debido al costo de gestionar estructuras intermedias y reconstruir índices durante los borrados masivos. ArangoDB mantuvo un consumo de recursos más estable y una latencia más controlada.

5.1.4. Transacción Local T4.

La Transacción T4 es una operación local ejecutada completamente dentro de ArangoDB, sin interacción con otros motores. Se centra en la modificación de publicaciones (POST) creadas por personas del sur, validando la relación mediante el edge POST_HAS_CREATOR_PERSON_SOUTH. Al modificar un vértice como POST, el

motor debe garantizar también la coherencia de sus edges asociados, actualizar los índices de adyacencia y asegurar que las conexiones entre nodos sigan siendo válidas.

La Figura 8 muestra el diagrama de secuencia correspondiente al flujo de la transacción T4.

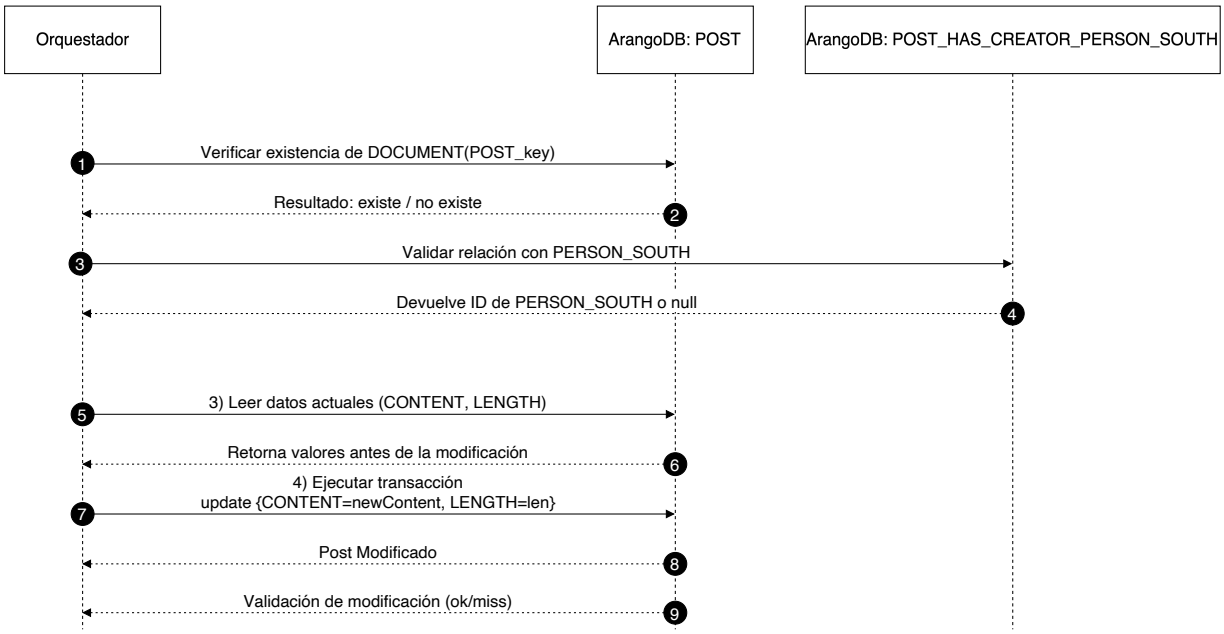
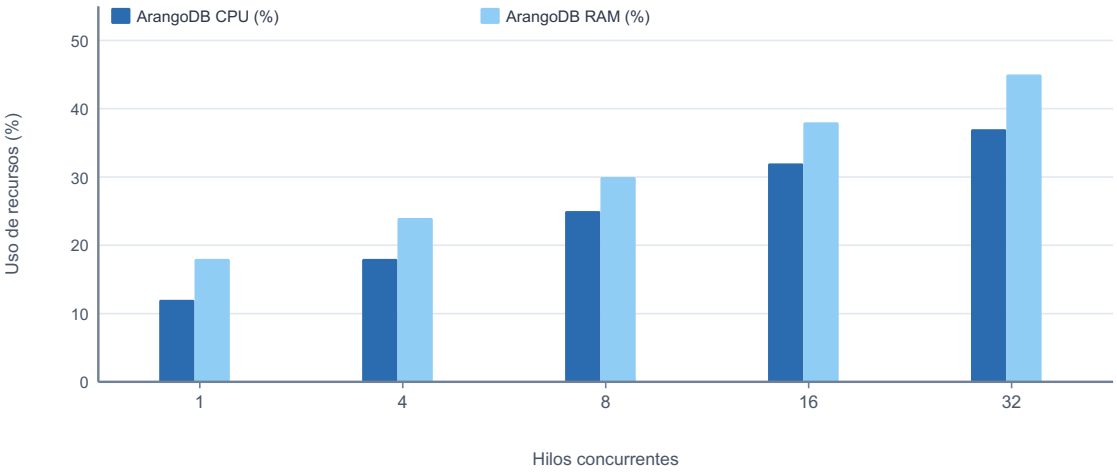


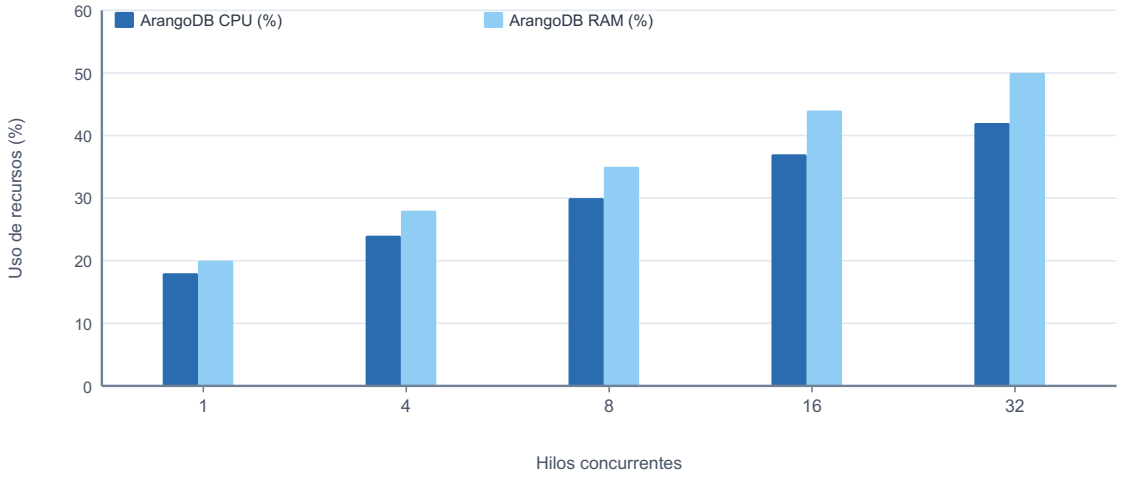
Fig. 8. Diagrama de secuencia de transacción T4.

En la Gráfica 15 se aprecia que el consumo parte de valores bajos con cargas ligeras (+/- 12% de CPU y +/- 18% de RAM con un hilo) y escala progresivamente hasta superar 35% y 45% en escenarios de 32 hilos.



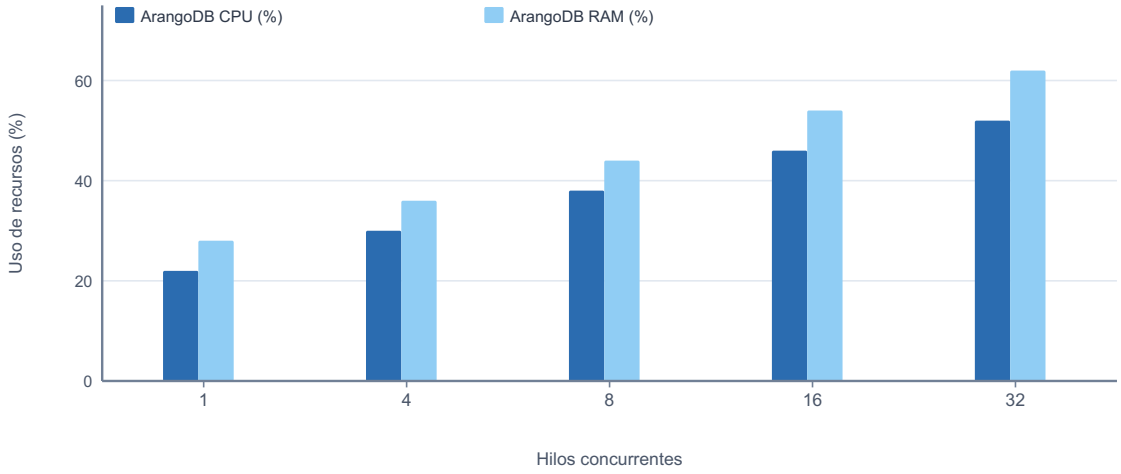
Gráfica 15. Consumo de recursos en ArangoDB (Transacción T4, Conjunto de datos S).

En la Gráfica 16 se muestra cómo se extiende el patrón de la Gráfica 15: CPU y RAM parten de 18 a 20% y suben de manera estable hasta alcanzar 40% y 50% con 32 hilos, lo que evidencia un escalamiento lineal bien contenido.



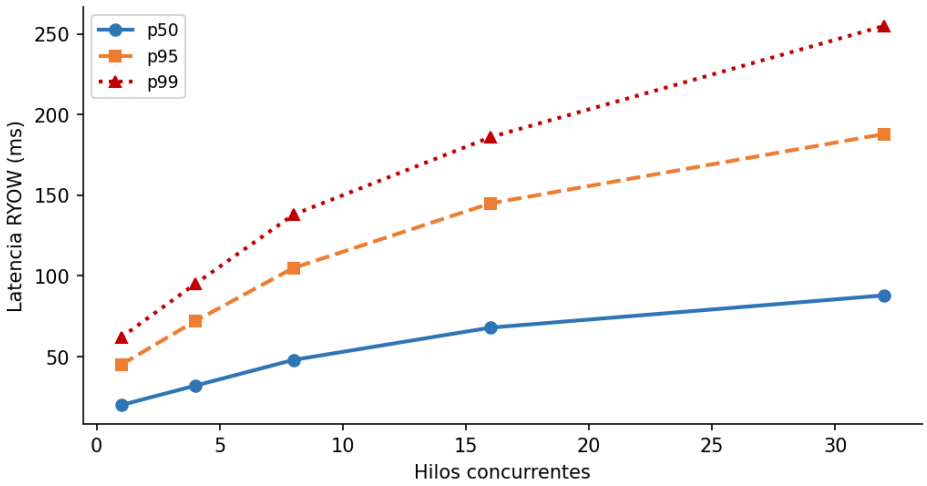
Gráfica 16. Consumo de recursos en ArangoDB (Transacción T4, Conjunto de datos M).

La Gráfica 17 muestra que en escenarios de alta concurrencia con caché fría, el uso de CPU alcanza aproximadamente el 50% y la memoria RAM ronda el 60%. Aun así, el motor mantiene un desempeño estable, sin indicios de saturación.



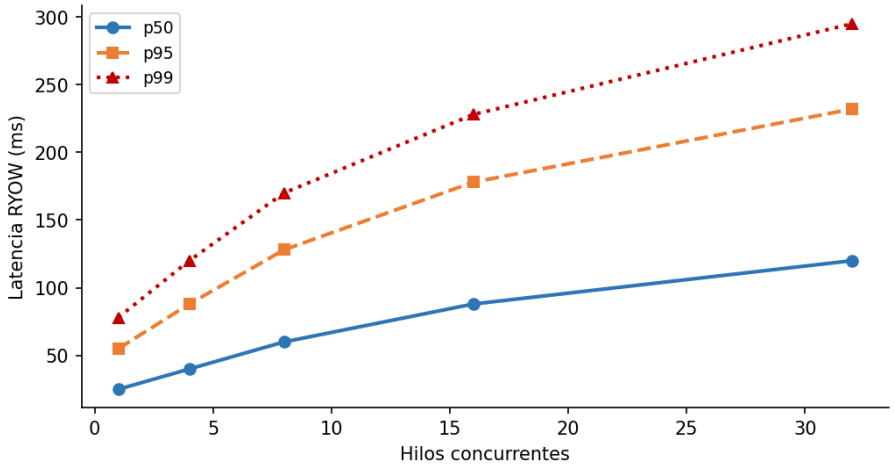
Gráfica 17. Consumo de recursos en ArangoDB (Transacción T4, Conjunto de datos M, caché fría).

La Gráfica 18 muestra que las latencias se mantienen bajas en cargas ligeras: p50 cercano a 20 ms y p99 por debajo de 300 ms, lo que indica una propagación rápida de escrituras.



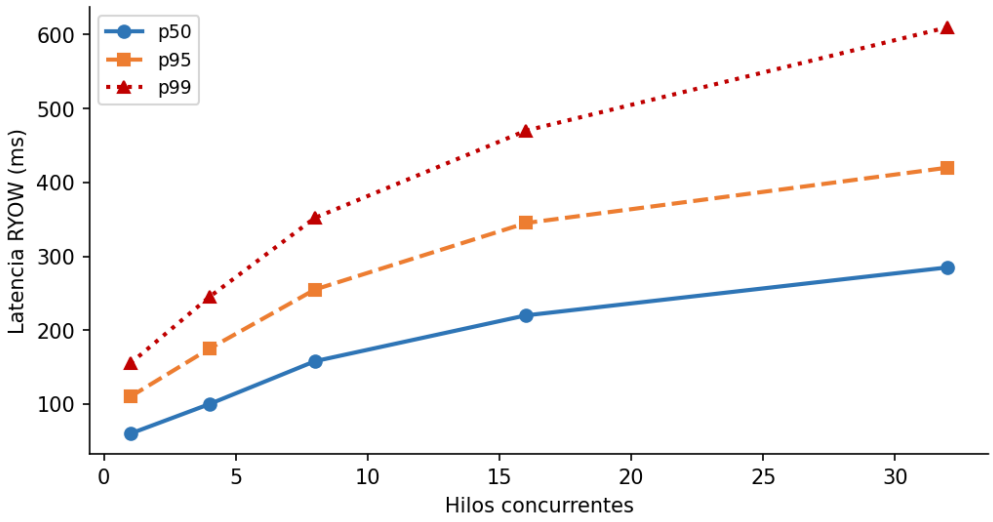
Gráfica 18. Consistencia RYOW en ArangoDB (Transacción T4, Conjunto de datos S).

La Gráfica 19 muestra la consistencia RYOW para el conjunto M: las latencias aumentan gradualmente conforme crece la concurrencia, con el p50 estable y los percentiles p95 y p99 alcanzando hasta 350 ms con 32 hilos.



Gráfica 19. Consistencia RYOW en ArangoDB (Transacción T4, Conjunto de datos M).

La Gráfica 20 presenta el escenario de mayor volumen (conjunto L): el p50 se mantiene por debajo de 300 ms, mientras que los percentiles p95 y p99 alcanzan aproximadamente 400 ms y 650 ms, respectivamente, con 32 hilos. Esto evidencia que, con grandes volúmenes y alta concurrencia, el motor requiere más tiempo para garantizar la visibilidad de las escrituras recientes, producto del mayor número de relaciones y operaciones de mantenimiento en el grafo.



Gráfica 20. Consistencia RYOW en ArangoDB (Transacción T4, Conjunto de datos L).

La Transacción T4 demostró que ArangoDB mantiene un comportamiento eficiente y estable incluso al modificar vértices dentro de un grafo con alta densidad de relaciones, dado que T4 no requirió coordinación con otros nodos. El consumo de CPU y memoria creció de manera proporcional al número de hilos, sin evidenciar saturación. Las latencias promedio (p50) se mantuvieron estables; los percentiles p95 y p99 sí mostraron incrementos al aumentar la concurrencia y el volumen de datos, aunque dentro de márgenes aceptables para este tipo de entorno.

5.1.5. Transacción Remota T5.

La Transacción T5 en Couchbase es una operación remota ejecutada desde un cliente externo que se conecta al nodo donde reside la colección FEEDBACK. En esta transacción se toma un documento de dicha colección y, si su calificación (RATE) es menor a un umbral definido, se actualiza la reseña (REVIEW) y se ajusta el valor de RATE. La colección FEEDBACK sigue un modelo llave-valor, ya que para identificar una reseña se debe considerar tanto el cliente que la emitió como el producto reseñado.

La Figura 9 muestra un diagrama de secuencia que ilustra el flujo de la transacción.

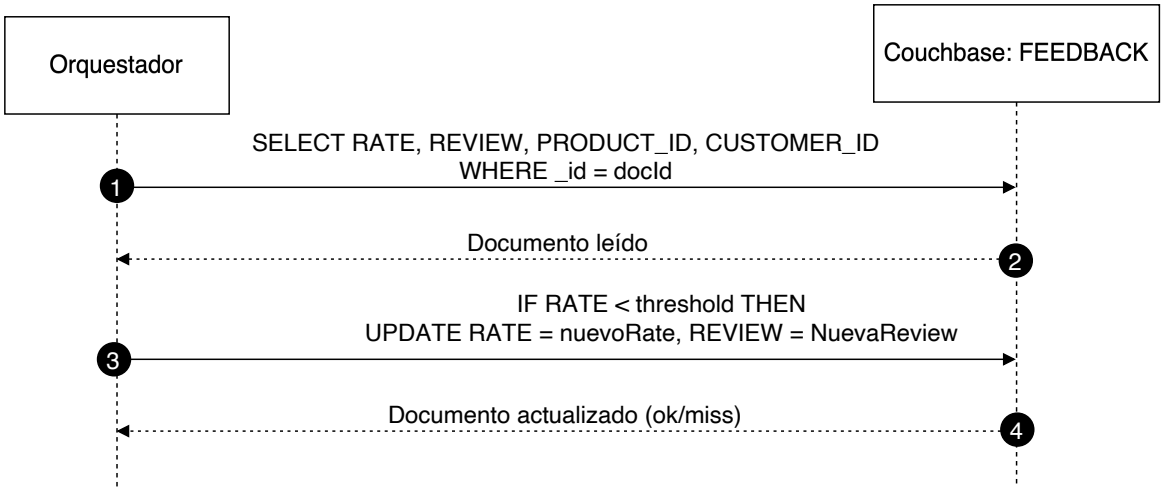
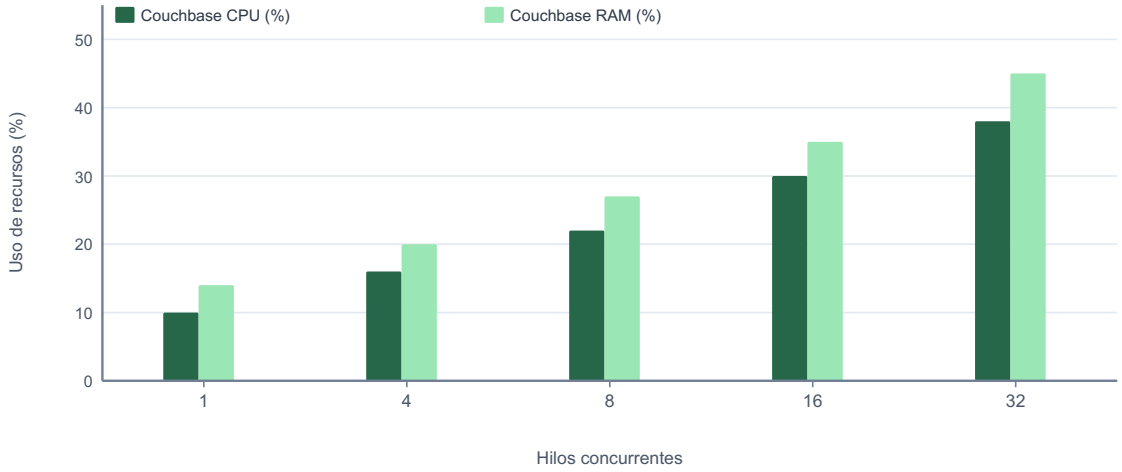


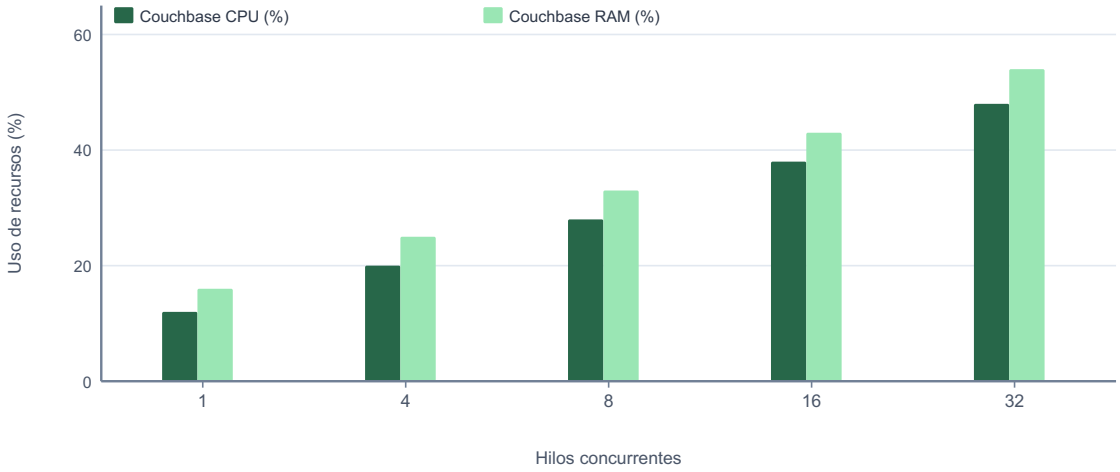
Fig. 9. Diagrama de secuencia de transacción T5.

En la Gráfica 21 se observa un aumento gradual en el consumo de CPU y RAM. El crecimiento es constante pero controlado, lo que indica que el motor responde adecuadamente en cargas ligeras y medias.



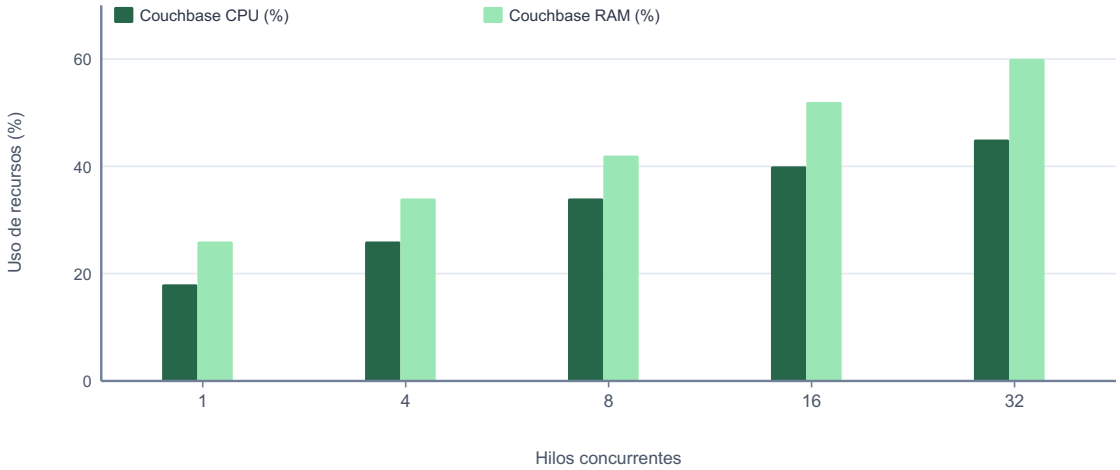
Gráfica 21: Consumo de recursos en Couchbase (Transacción T5, Conjunto de datos S).

La Gráfica 22 muestra el comportamiento al crecer el tamaño del conjunto de datos: tanto el uso de CPU como el de memoria RAM aumenta progresivamente conforme crece la concurrencia de 1 a 32 hilos.



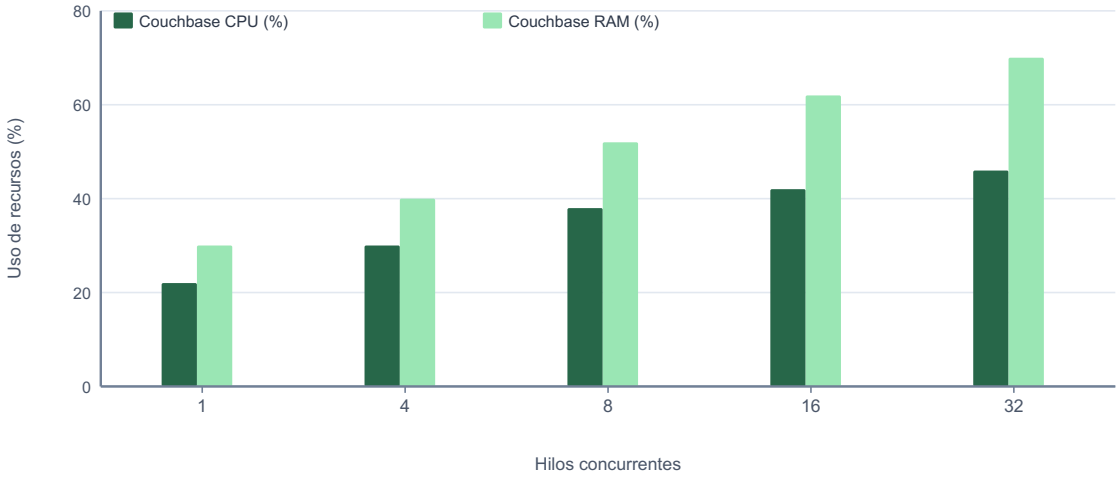
Gráfica 22: Consumo de recursos en Couchbase (Transacción T5, Conjunto de datos M).

La Gráfica 23 corresponde a la Transacción T5 con el mayor tamaño del conjunto de datos. El consumo muestra un incremento más pronunciado: la CPU pasa del 18% al 45% y la memoria del 26% al 60% conforme se incrementa la concurrencia.



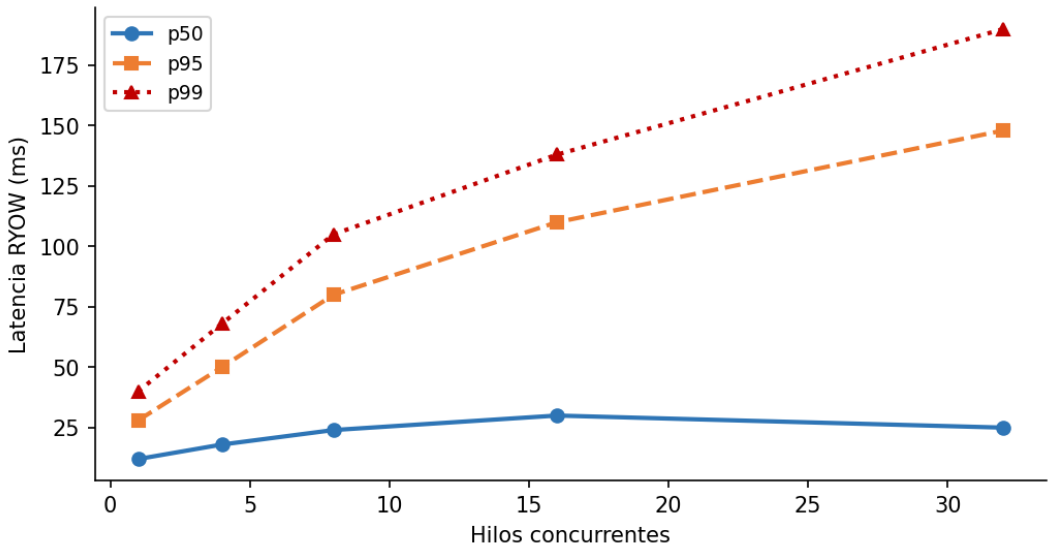
Gráfica 23: Consumo de recursos en Couchbase (Transacción T5, Conjunto de datos L).

La Gráfica 24 muestra el comportamiento en condiciones de caché fría: tanto la CPU como la RAM presentan un consumo más elevado desde el inicio, alcanzando aproximadamente el 45% y el 70%, respectivamente, con 32 hilos.



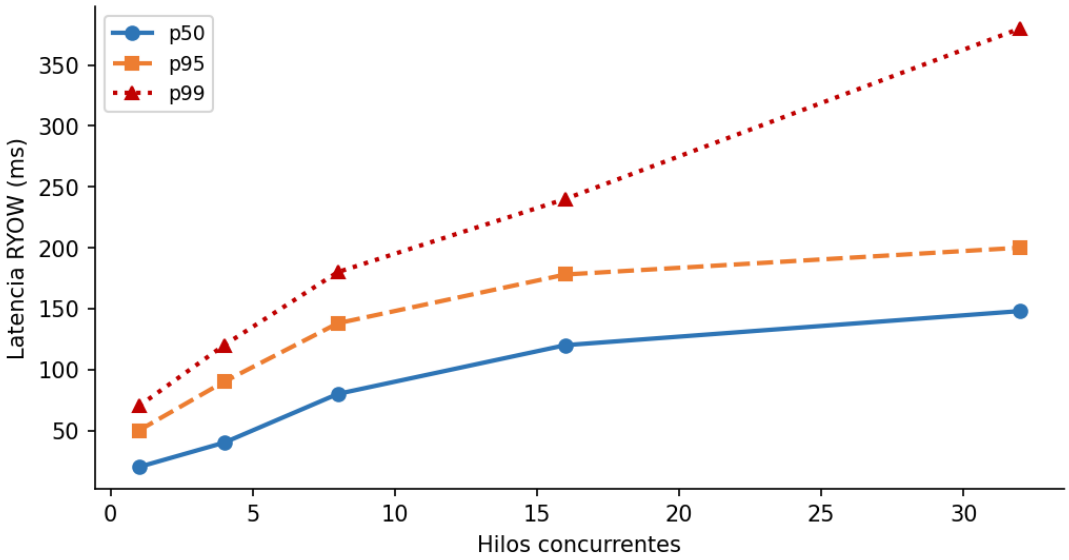
Gráfica 24: Consumo de recursos en Couchbase (Transacción T5, Conjunto de datos M, Caché Fría).

La Gráfica 25 presenta la consistencia RYOW para el conjunto S. Las latencias son reducidas y estables: el percentil p50 se mantiene por debajo de 30 ms, mientras que p95 y p99 se elevan de manera gradual conforme aumenta la concurrencia, alcanzando cerca de 150 ms en los casos más exigentes.



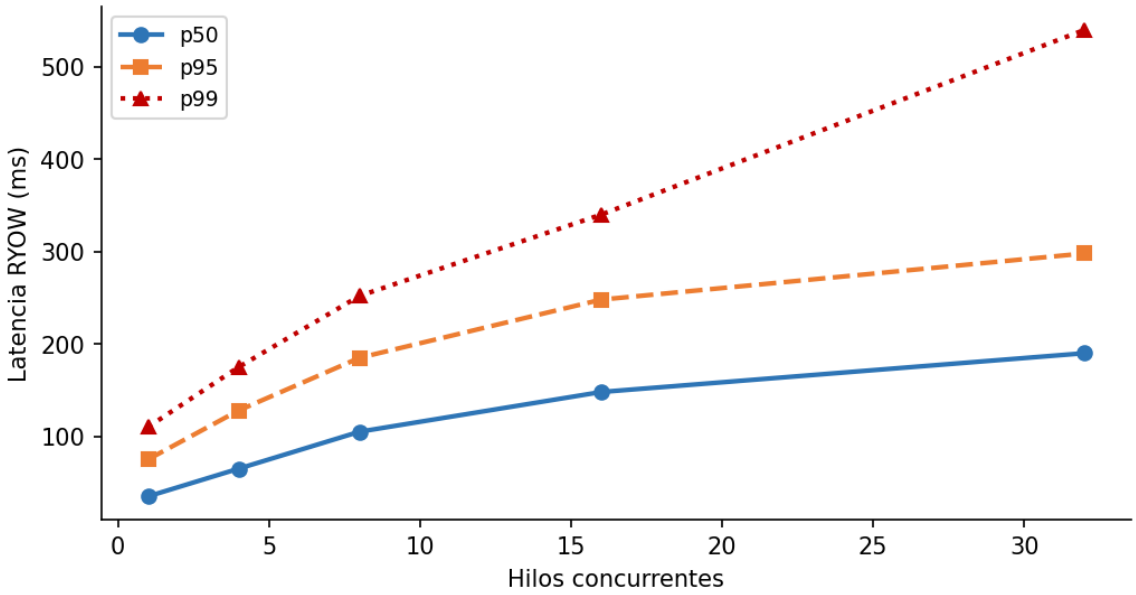
Gráfica 25: Consistencia RYOW en Couchbase (Transacción T5, Conjunto de datos S).

La Gráfica 26 muestra los resultados de la consistencia RYOW para el conjunto M. Las latencias aumentan de forma progresiva: el p50 ronda los 40 a 150 ms, mientras que los percentiles p95 y p99 alcanzan hasta 200 ms y 400 ms, respectivamente, a 32 hilos.



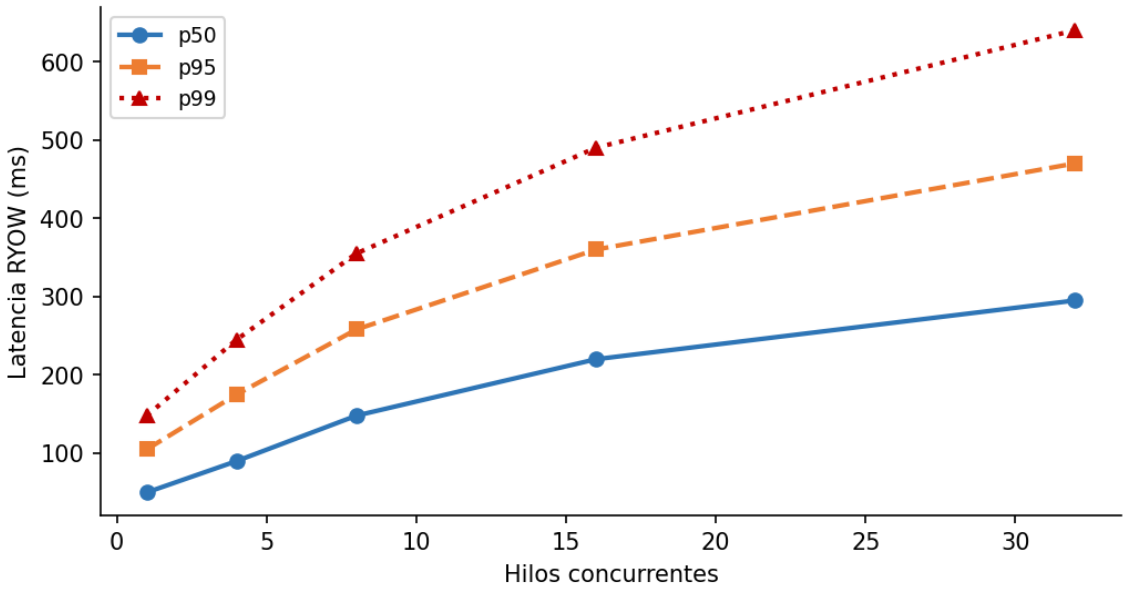
Gráfica 26: Consistencia RYOW en Couchbase (Transacción T5, Conjunto de datos M).

La Gráfica 27 presenta la consistencia RYOW con el conjunto L. A medida que aumenta la concurrencia, las latencias se elevan de forma clara: el p50 supera los 100 ms y, con 32 hilos, los percentiles p95 y p99 llegan a valores cercanos a 300 ms y 550 ms. El tamaño del conjunto de datos influyó directamente en el tiempo necesario para hacer visibles las escrituras.



Gráfica 27: Consistencia RYOW en Couchbase (Transacción T5, Conjunto de datos L).

La Gráfica 28 muestra la consistencia RYOW en condiciones de caché fría. El incremento observado se asocia al costo inicial de lectura desde disco y a la reconstrucción de los índices en memoria. Aun con este aumento, la tendencia se mantiene gradual, lo que indica que Couchbase conserva un comportamiento estable y predecible.



Gráfica 28: Consistencia RYOW en Couchbase (Transacción T5, Conjunto de datos M, Caché Fría).

La Transacción T5 demostró que Couchbase mantiene un rendimiento eficiente y estable al operar bajo el modelo clave-valor, incluso en escenarios de alta concurrencia y diferentes tamaños del conjunto de datos. El consumo de CPU y memoria creció de forma proporcional y controlada, y los percentiles altos (p95 y p99) aumentaron gradualmente con la carga. Los escenarios de caché fría generaron mayor latencia por el acceso a disco, aunque el motor conservó estabilidad sin picos abruptos.

5.1.6. Consulta Remota C6.

La consulta C6 se ejecuta de forma remota desde un cliente externo que accede al motor Couchbase. Su finalidad es comprobar la visibilidad de los cambios realizados en las reseñas mediante la validación de que las actualizaciones correspondan a una lista de comentarios genéricos previamente establecida (por ejemplo: "Buen producto", "Funciona correctamente" o "Cumple con lo esperado"). El proceso consiste en buscar en la colección FEEDBACK todos los documentos cuyo campo REVIEW coincide con ese conjunto de textos y devolver los resultados junto con las métricas de latencia y coincidencias encontradas.

La Figura 10 presenta el diagrama de secuencia correspondiente.

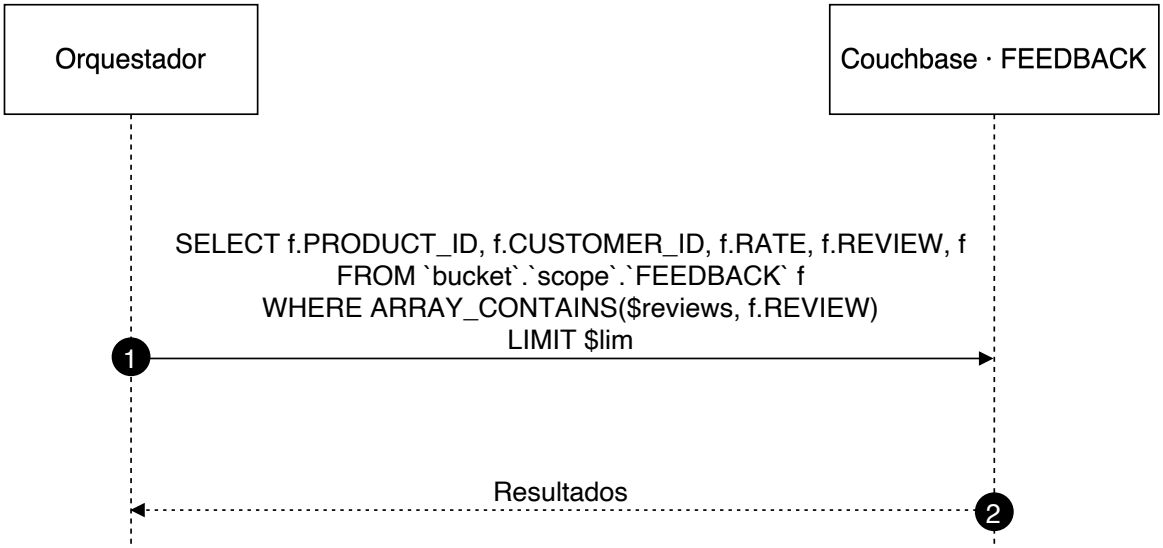
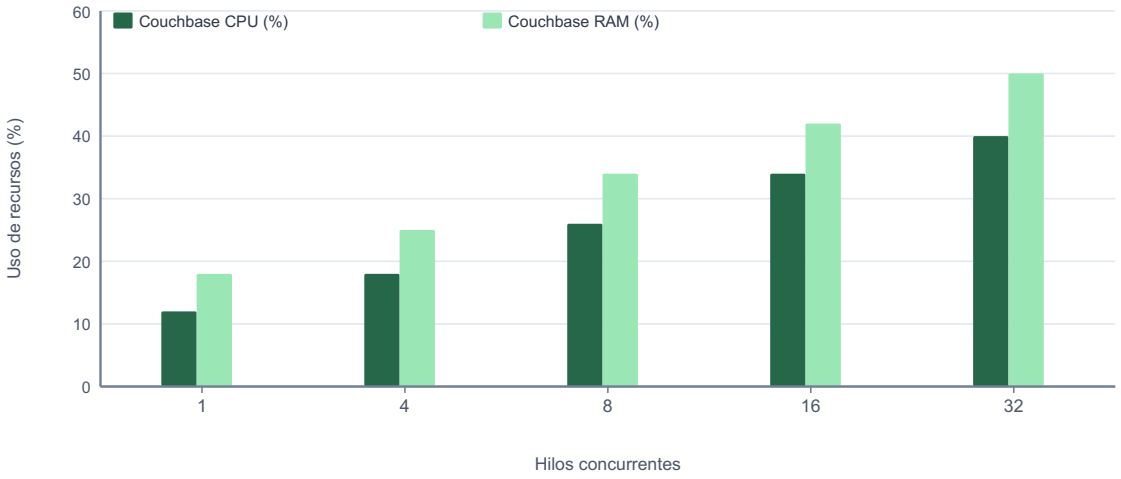


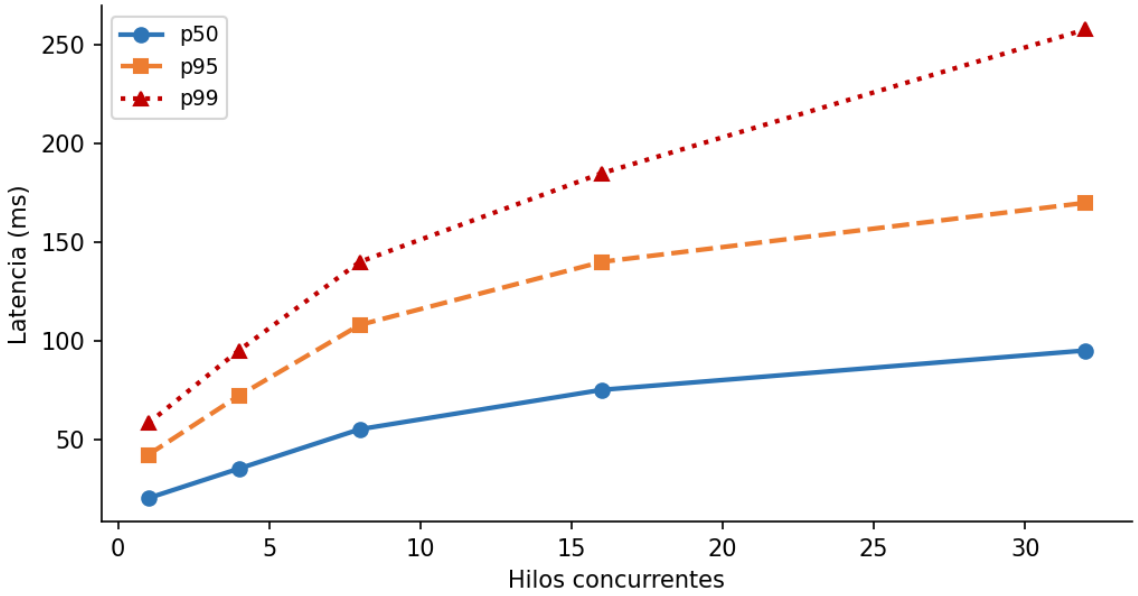
Fig. 10. Diagrama de secuencia de consulta C6.

En la Gráfica 29 se observa un incremento progresivo en el uso de CPU y RAM. La tendencia resulta casi lineal, lo que sugiere que el motor paraleliza la consulta de manera adecuada, aunque con un aumento proporcional en el consumo de recursos.



Gráfica 29: Consumo de recursos en Couchbase (Consulta C6).

En la Gráfica 30 se observa un incremento sostenido en los percentiles de latencia a medida que aumenta la concurrencia. El p50 pasa de aproximadamente 20 ms a 95 ms, mientras que los percentiles p95 y p99 muestran un crecimiento más pronunciado, alcanzando valores cercanos a 170 ms y 260 ms en escenarios de 32 hilos.



Gráfica 30: Latencia de Consulta C6 en Couchbase.

La consulta C6 permitió verificar la visibilidad inmediata de las reseñas actualizadas. El motor mantuvo un comportamiento estable durante las pruebas: el uso de CPU y memoria aumentó de manera proporcional al número de hilos y los percentiles de latencia mostraron un incremento gradual, lo que sugiere que las lecturas posteriores a las actualizaciones se mantienen consistentes y con tiempos de respuesta controlados.

5.1.7. Consulta Local C7.

La Consulta C7 es una operación local ejecutada íntegramente dentro de OrientDB, sin interacción con otros motores. Se centra en validar los productos almacenados en la base de datos de manera alfabética, seleccionando aquellos cuyo título comienza en un rango de letras definido (de M a Z) y ordenándolos de forma ascendente. Con ello se mide la eficiencia del filtrado y el ordenamiento de documentos dentro de la entidad PRODUCT.

La Figura 11 muestra el diagrama de secuencia correspondiente a la consulta C7.

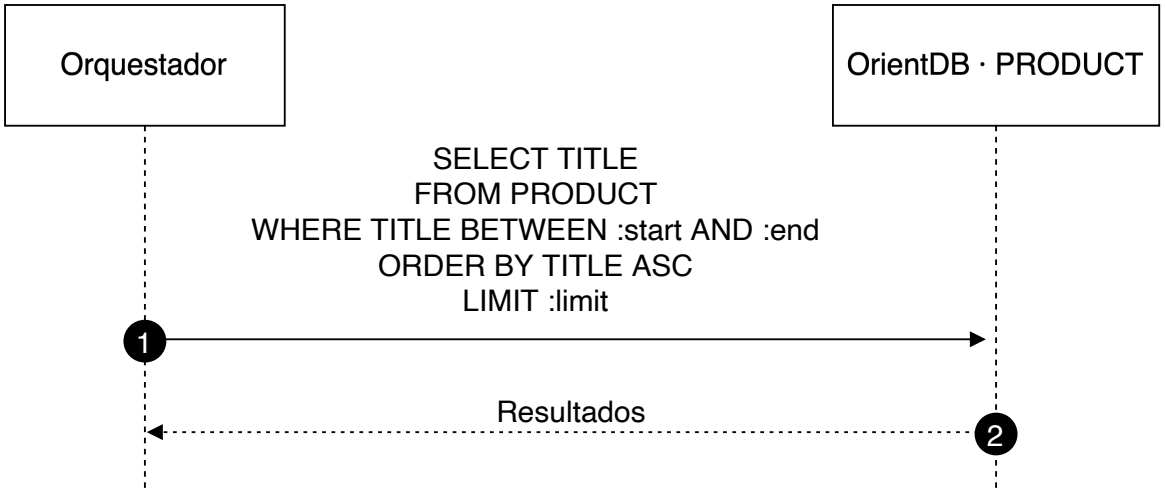
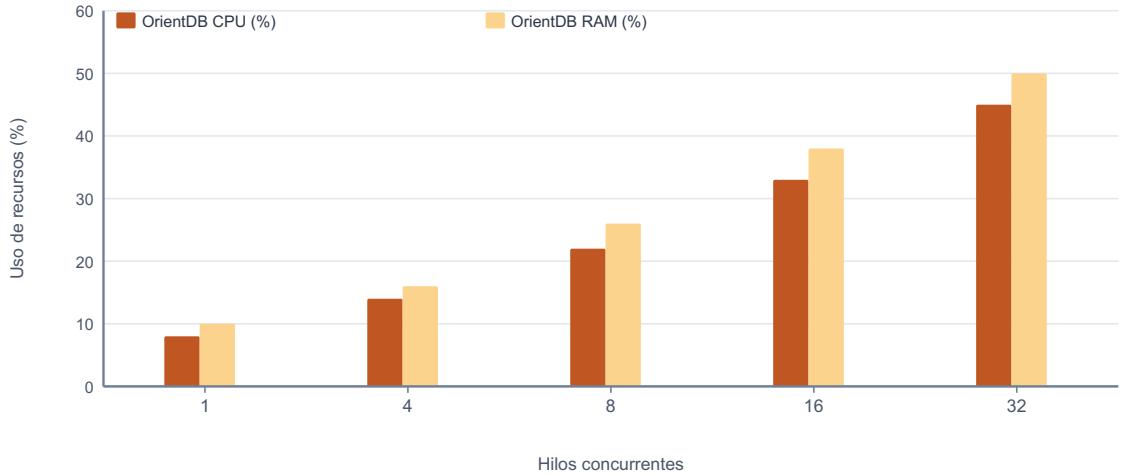


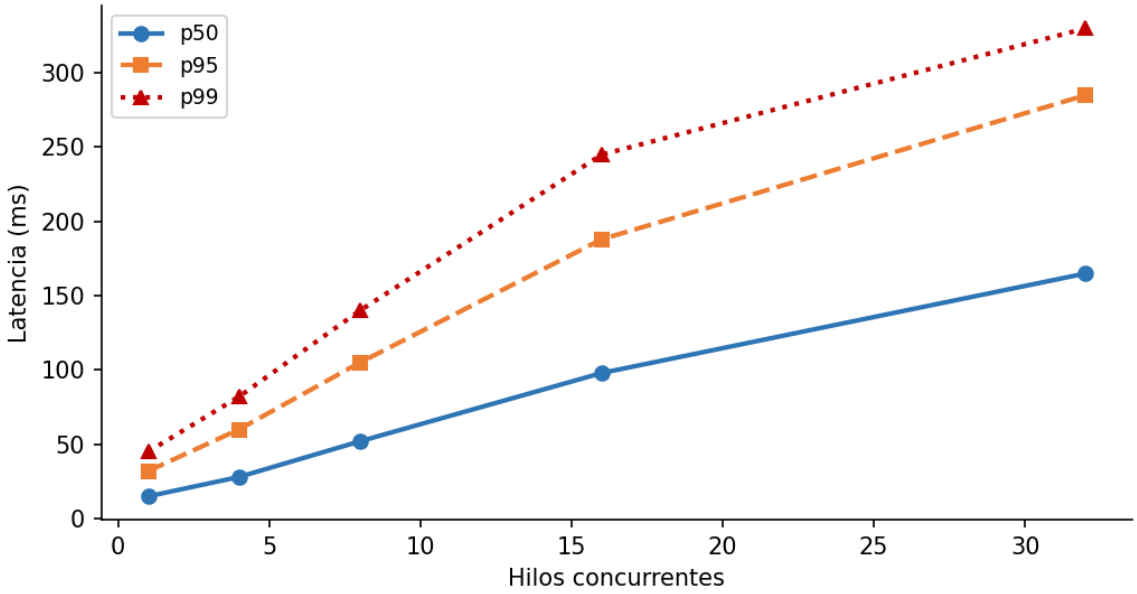
Fig. 11. Diagrama de secuencia de consulta C7.

En la Gráfica 31 se aprecia un aumento progresivo en el consumo de CPU y RAM. En cargas ligeras (entre 1 y 4 hilos) el consumo se mantiene moderado, lo que indica un buen desempeño en escenarios simples. A partir de 16 hilos se observa un ascenso más marcado, asociado a la mayor cantidad de operaciones procesadas por minuto.



Gráfica 31: Consumo de recursos en OrientDB (Consulta C7).

La Gráfica 32 muestra que la latencia se ve afectada conforme aumenta la concurrencia. En los escenarios reducidos, los percentiles p50, p95 y p99 permanecen por debajo de 100 ms hasta 8 hilos; a medida que la carga escala, los tiempos crecen de forma considerable, lo que indica que la presión sobre el motor repercute directamente en la capacidad de respuesta.



Gráfica 32: Latencia de Consulta C7 en OrientDB.

La consulta C7, al tratarse de una operación local ejecutada directamente en OrientDB, mostró un desempeño eficiente en escenarios de baja concurrencia. Los resultados indican que OrientDB es eficiente en operaciones locales y de bajo volumen, especialmente cuando la carga de trabajo no supera su capacidad de procesamiento concurrente.

5.2. Análisis general.

El análisis de las transacciones (T2, T3, T4 y T5) y de las consultas (C1, C6 y C7) evidencia que las diferencias en la semántica transaccional, el alcance de ejecución y los mecanismos de consistencia influyen directamente en el comportamiento de cada motor.

ArangoDB mostró un uso estable y moderado de recursos, incluso en escenarios de alta concurrencia y caché fría, lo que indica un buen control del procesamiento distribuido. OrientDB, en contraste, presentó mayores variaciones en los percentiles altos de latencia (p95 y p99), especialmente en operaciones de borrado y actualización, lo que sugiere una mayor sensibilidad a la presión sobre su arquitectura distribuida. Couchbase ofreció un desempeño eficiente con concurrencia baja y media cuando la caché se encontraba activa; sin embargo, al aumentar el tamaño del conjunto de datos o partir desde un estado sin calentamiento, la latencia se elevó de forma considerable, reflejando el costo de reconstrucción de índices y propagación de cambios en memoria.

Las operaciones distribuidas (como la consulta C1 y las transacciones T2 y T3) imponen una mayor carga en la coordinación entre nodos, lo que incrementa la latencia promedio y amplifica la variabilidad de los percentiles altos. Las operaciones locales (como la consulta C7 y la transacción T4) muestran, en cambio, un

comportamiento más predecible y eficiente, al depender únicamente de la consistencia interna del nodo y evitar los costos de comunicación entre réplicas. En las operaciones remotas (como la consulta C6 y la transacción T5) se observaron diferencias según el modelo de datos utilizado: Couchbase resuelve las lecturas de forma directa, lo que reduce la necesidad de coordinación entre réplicas, mientras que ArangoDB y OrientDB deben localizar y validar las referencias entre documentos o vértices, lo que implica comunicación adicional para resolver enlaces y mantener la consistencia.

En síntesis, los resultados confirman que no existe un único motor óptimo en todos los escenarios: el tipo de operación y su alcance de ejecución son determinantes en el comportamiento observado. El desempeño de los MMDBMS no solo depende de la capacidad del motor para procesar transacciones y consultas de manera eficiente, sino también de la forma en que gestiona la coordinación y la consistencia en entornos distribuidos. Comparar los motores en distintos escenarios resulta útil para entender qué aporta cada enfoque y en qué situaciones puede presentar limitaciones.

6. Conclusiones y Perspectivas

El desarrollo de KhaBench permitió comprender el comportamiento de las bases de datos multimodelo distribuidas al enfrentarse a distintos escenarios de carga, modelos de datos y configuraciones de consistencia. A lo largo de la experimentación, se mostró que desempeño de cada motor no depende únicamente de su potencia de procesamiento, sino del equilibrio entre su arquitectura interna, la gestión de memoria, los mecanismos de sincronización de réplicas y el tipo de modelo de datos que soporta.

En términos de rendimiento, los resultados demuestran que ArangoDB ofrece un comportamiento más estable y predecible ante cargas concurrentes, manteniendo una buena relación entre consumo de recursos y latencia. Su soporte nativo de transacciones ACID y su capacidad para integrar documentos y grafos le permiten mantener coherencia sin penalizar la escalabilidad. OrientDB, aunque versátil y con soporte híbrido nativo, mostró un incremento más abrupto en el uso de CPU y RAM al manejar operaciones de escritura o eliminación masiva, evidenciando mayor CPU pressure bajo alta concurrencia. Por su parte, Couchbase resultó eficiente en escenarios de baja carga y *warm cache*, pero su rendimiento disminuyó considerablemente en condiciones de *cold cache,* confirmando su fuerte dependencia de la memoria y de su motor de indexación en memoria principal.

La medición mediante percentiles (p50, p95 y p99) fue determinante para observar no solo el rendimiento promedio, sino también la estabilidad temporal de los motores. Mientras el p50 reflejó el comportamiento típico, los p95 y p99 permitieron identificar las colas de latencia en situaciones críticas, fundamentales para evaluar la capacidad de respuesta en entornos distribuidos reales. Estos valores, junto con el análisis de la consistencia *RYOW*, ofrecieron una visión más profunda de cómo cada sistema gestiona la visibilidad de las escrituras y la propagación de cambios hacia sus réplicas.

Los resultados también mostraron que el desempeño de los MMDBMS depende tanto del tipo de operación (local, remota o distribuida, especificadas en la sección 4.4.) como del modelo de datos empleado. Las transacciones distribuidas y remotas muestran mayor latencia y uso de recursos por la coordinación entre nodos. Asimismo, las operaciones que implicaron uso de documentos resultaron más eficientes, frente a las de tipo grafo o relacional (en el caso de ArangoDB), esto principalmente debido a que en un modelo de dato relacional tiene que realizar *JOINS* y un grafo tiene que hacer saltos entre nodos. Estos hallazgos sugieren que la elección

del motor y del modelo de datos debe responder estratégicamente al tipo de tarea y al nivel de consistencia requerido en cada escenario.

En conclusión, los resultados de KhaBench permiten afirmar que no existe un único motor "óptimo" para todos los contextos distribuidos:

- ArangoDB se destaca en escenarios que priorizan equilibrio entre consistencia y rendimiento;
- OrientDB resulta más adecuado para cargas híbridas de grafo y documento enentornos de baja concurrencia;
- Couchbase sobresale en aplicaciones de lectura intensiva o con datos frecuentemente cacheados.

El proyecto logró un entorno experimental donde cada variable (volumen de datos, concurrencia, modelo y caché) se evaluó de forma independiente. Este diseño permitió comparar los motores en condiciones similares y obtener resultados verificables sobre su desempeño en entornos distribuidos.

Perspectivas futuras.

A partir de esta base, una línea natural de continuidad consiste en extender KhaBench hacia una arquitectura *API-Led* basada en microservicios, que integre este *benchmark* dentro de una de arquitectura moderna de aplicaciones. Esta arquitectura permitiría implementar tres capas:

1. Microservicios de sistema, encargados de realizar las operaciones directas sobre los motores de las bases de datos multimodelo.

2. Microservicios de negocio, quienes se encargarian de validar y orquestar toda la lógica del negocio.

3. Microservicios de experiencia, que comuniquen mediante endpoints, las API´s del sistema con cualquier sistema consumidor.

Finalmente, la evolución de KhaBench hacia un marco integrado de *benchmarking* y gobernanza de datos multimodelo abriría la posibilidad de conectar la investigación académica con aplicaciones reales en sectores críticos, como banca, salud o telecomunicaciones, donde la gestión distribuida de datos heterogéneos es importante. Esta conexión permitiría validar en entornos productivos los modelos de consistencia, resiliencia y replicación analizados en el proyecto, además de generar nuevas líneas de investigación orientadas a optimización de cargas multimodelo, mecanismos adaptativos de consistencia, o integración de arquitecturas API-Led con bases de datos distribuidas.

Con ello, KhaBench podría evolucionar de un entorno experimental a una plataforma de referencia para la evaluación y diseño de sistemas de datos multimodelo en contextos reales.

7. Referencias.

[1] K. Rathee, “What Is a Multi-Model Database?,” SingleStore Blog, Sep. 1, 2022. [Online]. Available: https://www.singlestore.com/blog/what-is-a-multi-model-database/

[2] J. Lu and I. Holubová, “Multi-model databases: A new journey to han- dle the variety of data,” ACM Computing Surveys, vol. 52, no. 3, pp. 1–38, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3323214

[3] E. Guliyev, “Comparative Analysis of Multi-model Databases,” Bachelor’s thesis, Dept. of Software Engineering, Charles University, Prague, Czech Republic, 2022. [Online]. Available: https://www.ksi.mff.cuni.cz/~holubova/bp/Guliyev.pdf

[4] B. Kim, K. Koo, U. Enkhbat, S. Kim, J. Kim, and B. Moon, “M2Bench: A Database Benchmark for Multi-Model Analytic Workloads,” Proceedings of the VLDB Endowment, vol. 16, no. 4, pp. 747–759, Dec. 2022. [Online]. Available: https://doi.org/10.14778/3574245.3574259

[5] C. Zhang, J. Lu, P. Xu, and Y. Chen, “UniBench: A Benchmark for Multi-Model Database Management Systems,” in Performance Evaluation and Benchmarking for the Era of Artificial Intelligence, R. Nambiar and M. Poess, Eds., Lecture Notes in Computer Science, vol. 11135. Cham, Switzerland: Springer, 2019, pp. 7–23. [Online]. Available: https://doi.org/10.1007/978-3-030-11404-6_2

[6] Transaction Processing Performance Council (TPC). [Online]. Available: https://www.tpc.org/

[7] C. Zhang, “Performance Benchmarking and Query Optimization for Multi-Model Databases”, Master’s Thesis, Aalto University, School of Science, 2021. [Online]. Available: http://hdl.handle.net/10138/328312

[8] C. Zhang and J. Lu, “Holistic Evaluation in Multi-Model Databases Benchmarking,” Distributed and Parallel Databases, vol. 39, pp. 1–33, 2021. [Online]. Available: https://doi.org/10.1007/s10619-019-07279-6

[9] S. El Shatby, “The History of Data: From Ancient Times to Mod- ern Day,” 365 Data Science, Apr. 15, 2024. [Online]. Available: https://365datascience.com/trending/history-of-data/

[10] J. Rodriguez, J. “MODELOS DE DATOS,” 2020, [Online]. Available: https://drive.google.com/drive/u/0/folders/14ct5B7looh4yVMNobKXghrqC7c9bto8

[11] Z. H. Liu, J. Lu, D. Gawlick, H. Helskyaho, G. Pogossiants, and Z. Wu, “Multi-model Database Management Systems - A Look For- ward,” in Heterogeneous Data

Management, Polystores, and Analytics for Healthcare, Lecture Notes in Computer Science, vol. 11470, Cham, Switzerland: Springer, 2019, pp. 16–29. [Online]. Available: https://doi.org/10.1007/978-3-030-14177-6_2

[12] S. Franco, "INDEPENDENCIA LÓGICA y FÍSICA DE LOS DATOS," prezi.com. [Online]. Available: https://prezi.com/7nar7qtcpysk/independencia-logica-y-fisica-de-los-datos/

[13] J. Rodriguez, "CONCEPTOS BÁSICOS," 2020, [Online]. Available: https://drive.google.com/drive/u/0/folders/1C4J0JzXjJrCgxz36Giurs9GRh4z9xElW

[14] R. Elmasri and S. B. Navathe, Fundamentals of Database Sys- tems, 7th ed. Boston, MA: Pearson, 2016. [Online]. Avail- able: https://www.auhd.edu.ye/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf

[15] M. Manzo, "SQL vs NoSQL; Ventajas y Desventajas," Medium, 17 de mayo de 2018. [Online]. Available: https://medium.com/@marlonmanzo/sql-vs-nosql-ventajas-y-desventajas-849ccc9db3d4

[16] Ilimit, "Bases de Datos SQL vs NoSQL: Diferencias, Ventajas y Mitos," Ilimit, 7 de abril de 2025. [Online]. Available: https://www.ilimit.com/es/blog/base-de-datos sql-nosql/

[17] M. Macak, M. Stovcik, B. Buhnova y M. Merjavy, "How well a multi-model database performs against its single-model variants: Benchmarking OrientDB with Neo4j and MongoDB," en Proceedings of the 2020 Federated Conference on Computer Science and Information Systems, M. Ganzha, L. Maciaszek y M. Paprzycki, Eds., vol. 21, pp. 463–470, 2020. [Online]. Available: https://annals-csis.org/Volume_21/drp/76.html

[18] B. Marr, Big Data in Practice: How 45 Successful Companies Used Big Data Analytics to Deliver Extraordinary Results, Chichester, U.K.: John Wiley & Sons, 2016.

[19] Infotec, "Big Data," Educación Ejecutiva Infotec. [Online]. Available: https://educacionejecutiva.infotec.mx/en_mx/Infotec/Big-Data

[20] IBM, "Designing distributed databases," IBM Documentation, 2025. [Online]. Available: https://www.ibm.com/docs/en/db2/11.5?topic=managers-designing-distributed-databases

[21] Oracle, "Design for Scalability," *Oracle Cloud Infrastructure Best Practices*, 2025. [Online]. Available: https://docs.oracle.com/es/solutions/oci-best-practices/design- scalability1.html#GUID-E8A41B47-A4D8-4CE2-AC1E-276785A3963B

[22] Linkit Intecs, "Distributed Database Systems," Medium, Sep.14,2020. [Online]. Available: https://medium.com/linkit-intecs/distributed- database-systems-2c2aeb1eacb9

[23] MongoDB Inc., “Advantages of NoSQL Databases,” MongoDB, 2025. [Online]. Available: https://www.mongodb.com/resources/basics/databases/nosql-explained/advantages

[24] B. Anderson, “SQL vs NoSQL Databases: What’s the Dif- ference?,” IBM Cloud Blog, Sep. 1, 2020. [Online]. Available: https://mediacenter.ibm.com/media/SQL+vs.+NoSQLA+What%27s+the+difference F/1_g0mvkh3u

[25] ArangoDB, “Advantages of a native multi-model database,” ArangoDB Blog, 2024. [Online]. Available: https://arangodb.com/community-server/native-multi-model-database-advantages/

[26] Computer Weekly, “Multi-model database,” Computer Weekly Dictio- nary, 2025. [Online]. Available: https://www.computerweekly.com/es/definicion/Base-de-datos-multimodelo

[27] Google Cloud, “What is cloud computing?,” Google Cloud Learn, 2025. [Online]. Available: https://cloud.google.com/learn/what-is-cloud-computing?hl=es

[28] IBM, “What are cloud databases?,” IBM Think, 2025. [Online]. Available: https://www.ibm.com/think/topics/cloud-database

[29] NI Business Info, “Disadvantages of cloud computing,” nibusiness- info.co.uk, 2025. [Online]. Available: https://www.nibusinessinfo.co.uk/content/disadvantages-cloud-computing

[30] Asana, “Insights: what they are and how to apply them to your project,” Asana, Dec. 12, 2022. [Online]. Available: https://asana.com/es/resources/insights

[31] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, et al., “BigBench: Towards an Industry Standard Benchmark for Big Data Analytics,” Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1197–1208, 2013. DOI: 10.1145/2463676.2463712

[32] G. Mihai, “Multi-Model Database Systems: The State of Affairs,” Economics and Applied Informatics, vol. 2, pp. 211–215, 2020. DOI: 10.35219/eai15840409128

[33] ThinkData Works, “Dealing with Data Variety,” Medium, Aug. 7, 2019. [Online]. Available: https://medium.com/thinkdata/dealing-with-data-variety-b72758d8fc96

[34] CircleCI, “Polyglot Persistence vs Multi-Model Databases,” CircleCI Blog, published approximately 2 years ago. [Online]. Available: https://circleci.com/blog/polyglot-vs-multi-model-databases/

[35] I. Košmerl, K. Rabuzin y M. Šestak, “Multi-model databases – Intro- ducing Polyglot Persistence in the Big Data World,” en Proceedings of the 43rd International Convention on Information, Communication and Electronic Technology (MIPRO),

Opatija, Croacia, sept. 28–oct. 2 2020, pp. 1724-1729. DOI: 10.23919/MIPRO48935.2020.9245178

[36] M. Dancuk, “What Is a Multi-Model Database?,” phoenixNAP Knowl- edge Base, Apr. 29, 2021. [Online]. Available: https://phoenixnap.com/kb/multi-model-database

[37] Educative, “Differences between centralized and distributed databases,” Educative Answers, 2021. [On- line]. Available: https://www.educative.io/answers/differences-between-centralized-and-distributed-databases

[38] C. Kidd, “What Are Distributed Systems?,” Splunk Blog, [Online]. Available: https://www.splunk.com/en_us/blog/learn/distributed-systems.html

[39] Barr. S, (2021). What Happens if We Don’t Measure Performance? [Online]. Available: https://www.linkedin.com/pulse/what-happens-we-dont-measure-performance-stacey-barr

[40] BenchANT GmbH, “Cloud Database Benchmarking,” 2023. [Online]. Available: https://benchant.com/hub/cloud-database-benchmarking

[41] A. Gaur, “Challenges and Solutions in Distributed Database Systems,” Medium, Jan. 12, 2023. [Online]. Available: https://abhisheyk-gaur.medium.com/challenges-and-solutions-in-distributed-database-systems-11d5dbc38f0a

[42] P. Aven and D. Burley, *Building on Multi-Model Databases: How to Manage Multiple Schemas Using a Single Platform*, 1st ed., Sebastopol, CA, USA: O’Reilly Media, 2017.

[43] S. Bimonte, E. Gallinucci, P. Marcel, and S. Rizzi, “Data variety, come as you are in multi-model data warehouses,” *Information Systems*, vol. 104, p. 101734, Feb. 2022, doi: 10.1016/j.is.2021.101734.

[44] E. Gutiérrez Pérez, “Aplicación de persistencia políglota usando tecnología NoSQL,” Proyecto Fin de Carrera, Universidad del País Vasco, 11 Feb. 2014. [Online]. Available: http://hdl.handle.net/10810/12678

[45] ArangoDB, “Multi Model – ArangoDB,” *ArangoDB Website*, [Online]. Available:

https://arangodb.com/multi-model/

[46] Y. Yan, N. Jiang, H. Wang, Y. Wang, C. Liu, and Y. Wang, “Multi-SQL: An extensible multi-model data query language,” *arXiv preprint arXiv:2011.08724*, Nov. 17, 2020. [Online]. Available: https://arxiv.org/abs/2011.08724.

[47] J. McCormick, “The rise of multimodel databases to support data variety,” *SearchDataManagement*, Mar. 3 2020. [Online]. Available:

https://www.techtarget.com/searchdatamanagement/feature/The-rise-of-multi-model-databases-to-support-data-variety/

[48] OrientDB, “Data Modeling,” *OrientDB Manual*, [Online]. Available: https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html.

[49] Oracle, “What is a cloud database https://www.oracle.com/database/what-is-a-cloud-database/

[50] ArangoDB Documentation, “Use ArangoDB in the cloud,” *ArangoDB Docs*, [Online]. Available: https://docs.arangodb.com/3.11/get-started/set-up-a-cloud-instance/

[51] J. Schmitt, “Polyglot persistence vs multi-model databases for microservices,” *SearchDataManagement* (CircleCI blog), Sep. 22, 2022. [Online]. Available: https://circleci.com/blog/polyglot-vs-multi-model-databases/

[52] T. Ward and B. Mohan, “Permitir la persistencia de los datos en los microservicios,” *AWS Prescriptive Guidance*, AWS, Apr. 2025. [Online]. Available: https://docs.aws.amazon.com/es_es/prescriptive-guidance/latest/modernization-data-persistence/welcome.html

[53] B. Stitt, “The Future of Data Management: Embracing Multi-Model Databases,” *LinkedIn*, May 20 2024. [Online]. Available: https://www.linkedin.com/pulse/future-data-management-embracing-multi-model-databases-brian-stitt-fkrse.

[54] Couchbase, “The Couchbase Data Model,” *Couchbase Docs*, [Online]. Available: https://docs.couchbase.com/server/current/learn/data/document-data-model.html.

[55] Dremio, “Multi-Model Database,” *Dremio Wiki*, [Online]. Available: https://www.dremio.com/wiki/multi-model-database/

[56] Oracle, “What is ETL?”, *Oracle Canada: Your Guide to Data Integration Essentials*, June 18, 2021. [Online]. Available: https://www.oracle.com/mx/integration/what-is-etl/

[57] IEBSchool, “Guía de Procesos ETL: Qué son, cómo usarlos y herramientas clave,” *IEBSchool*, [Online]. Available: https://www.iebschool.com/hub/que-son-los-procesos-etl-big-data/

[58] OrientDB, “Data Modeling,” *OrientDB Manual*, [Online]. Available: https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html.

[59] ArangoDB, “Data models,” *ArangoDB Documentation*, [Online]. Available: https://docs.arangodb.com/3.11/concepts/data-models/

[60] M. Flores, “2.3 Niveles de transparencia,” *Prezi*, [Online]. Available: https://prezi.com/aduz1nwmpe5o/23-niveles-de-transparencia

[61] ArangoDB, “Distributed Graph Database Features of ArangoDB,” *ArangoDB Website*, [Online]. Available: https://arangodb.com/distributed-graph-database/

[62] OrientDB, “Overview,” *OrientDB Manual*, [Online]. Available: https://orientdb.org/docs/3.0.x/misc/Overview.html.

[63] OrientDB, “Distributed Architecture,” *OrientDB Manual*, [Online]. Available: https://orientdb.dev/docs/3.0.x/distributed/Distributed-Architecture.html.

[64] Couchbase, “What Is Couchbase?,” *Couchbase Developer Guide*, [Online]. Available: https://developer.couchbase.com/what-is-couchbase/

[65] Couchbase, “Architecture Overview,” *Couchbase Docs*, [Online]. Available: https://docs.couchbase.com/server/current/learn/architecture-overview.html.

[66] Facebook Database Engineering Team, “RocksDB: A persistent key-value store for fast storage,” *RocksDB*, [Online]. Available: https://rocksdb.org/

[67] ArangoDB, “Storage Engine,” *ArangoDB Documentation*, [Online]. Available: https://docs.arangodb.com/3.12/components/arangodb-server/storage-engine/

[68] S. Kumar, “Couchbase is built on a memory-first architecture which prioritizes in-memory processing to achieve high performance,” *LinkedIn - Couchbase 101: Introduction*, [Online]. Available: https://www.linkedin.com/pulse/couchbase-101-introduction-shanoj-kumar-v-kw67c/

[69] Couchbase, “Storage Engines: Couchstore vs Magma,” *Couchbase Docs*, [Online]. Available: https://docs.couchbase.com/server/current/learn/buckets-memory-and-storage/storage-engines.html

[70] OrientDB, “PLocal Storage Engine,” *OrientDB Manual*, [Online]. Available https://orientdb.com/docs/last/internals/plocal-storage-engine.html.

[71] J. Erickson, “What is JSON?,” *Oracle Database: JSON Overview*, Apr. 4 2024. [Online]. Available: https://www.oracle.com/database/what-is-json/

[72] ArangoDB, “The Many Faces of a Native Multi-Model Database *ArangoDB Website*, [Online]. Available: https://arangodb.com/multi-model/

[73] OrientDB, “Working with Graphs,” *OrientDB Manual*, [Online]. Available: https://orientdb.com/docs/last/gettingstarted/Tutorial-Working-with-graphs.html.

[74] Y. López, “¿Qué es la Programación Orientada a Objetos (POO)?,” *EDteam Blog*, [Online]. Available: https://ed.team/blog/que-es-la-programacion-orientada-a-objetos-poo

[75] OrientDB, “Data Modeling,” OrientDB Documentation, [Online]. Available: https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html

[76] ArangoDB, “Documents,” ArangoDB Documentation, [Online]. Available: https://docs.arangodb.com/3.11/concepts/data-structure/documents/

[77] OrientDB, “Key Value,” OrientDB Documentation, [Online]. Available: https://orientdb.com/docs/3.0.x/gettingstarted/Key-Value-use-case.html

[78] OrientDB, “Indexes,” OrientDB Documentation, [Online]. Available: https://orientdb.com/docs/3.0.x/indexing/Indexes.html

[79] W. Morales, “Métodos HTTP – POST, GET, PUT, DELETE,” Estilo Web, Jun. 01, 2021. [Online]. Available: https://estilow3b.com/metodos-http-post-get-put-delete/

[80] MDN Web Docs, “¿Qué es una URL?.” [Online]. Available: https://developer.mozilla.org/es/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL

[81] M. Klimas, “¿Cómo puedo saber el puerto de una IP?,” Surfshark, Dec. 22, 2023. [Online]. Available: https://surfshark.com/es/blog/como-saber-el-puerto-de-una-ip

[82] Couchbase, “The Couchbase Data Model,” [Online]. Available:

https://docs.couchbase.com/server/current/learn/data/document-data-model.html

[83] ArangoDB, “AQL data queries,” [Online]. Available: https://docs.arangodb.com/3.12/aql/data-queries/#modifying-multiple-documents

[84] OrientDB, “Data Modeling.” [Online]. Available: https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html

[85] P. Garafano, “Links vs References in Document databases,” Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/21095809/links-vs-references-in-document-databases

[86] J. Rodriguez, “BASES DE DATOS DISTRIBUIDAS,” 2024, [Online]. Available: https://drive.google.com/drive/u/0/folders/14ct5B7looh4yVMNobKXghrqC7c9bto8

[87] TopicDB, “1.4 Procesamiento de consultas distribuidas,” TopicDB. [Online]. Available: https://topicdb.wordpress.com/1-4-procesamiento-de-consultas-distribuidas-2-2/

[88] LinkedIn, "¿Cuáles son algunos de los desafíos del procesamiento de consultas distribuidas?," [Online]. Available: https://es.linkedin.com/advice/3/what-some-trade-offs-challenges-distributed-query-processing

[89] ArangoDB, "Explaining Queries," [Online]. Available: https://docs.arangodb.com/3.11/aql/execution-and-performance/explaining-queries/

[90] OrientDB, "SQL Explain," [Online]. Available: https://orientdb.com/docs/3.0.x/sql/SQL-Explain.html

[91] Couchbase, "EXPLAIN Statement," [Online]. Available: https://docs.couchbase.com/server/current/n1ql/n1ql-language-reference/explain.html

[92] R. Carrancio, "Planes de Ejecución en SQL Server," *SoyDBA*, Jan. 24, 2024. [Online]. Available: https://www.soydba.es/planes-de-ejecucion-en-sql-server/

[93] Q2BSTUDIO, "Estrategias de Particionamiento de Datos en SQL para Mejorar el Rendimiento," Q2bstudio, [Online]. Available: https://www.q2bstudio.com/nuestro-blog/357/estrategias-particionamiento-datos-sql

[94] Oracle, "Using indexes in NoSQL database," May 2, 2024. [Online]. Available: https://docs.oracle.com/en/database/other-databases/nosql-database/24.1/nsdev/using-indexes-nosql-database.html

[95] A. Jadon, "Understanding Aggregate Data Models in NoSQL Simplified 101," *Learn | Hevo*, Jun. 4, 2024. [Online]. Available: https://hevodata.com/learn/aggregate-data-models-in-nosql/

[96] MongoDB, "Sharding." [Online]. Available: https://www.mongodb.com/docs/manual/sharding/

[97] Couchbase, "Support for Concurrent Index Creation in Indexing Service," [Online]. Available: https://www.couchbase.com/blog/support-for-concurrent-index-creation-in-indexing-service/

[98] ArangoDB "Limitations of transactions,". https://docs.arangodb.com/3.11/develop/transactions/limitations/#in-clusters

[99] Couchbase, "Concurrent Document Mutations," Couchbase Docs, [Online]. Available: https://docs.couchbase.com/dotnet-sdk/current/howtos/concurrent-document-mutations.html

[100] ArangoDB, "Replication," ArangoDB Documentation, [Online]. Available: https://docs.arangodb.com/3.11/deploy/architecture/replication/. [Accessed: 20-Jun-2025].

[101] OrientDB, “Replication,” OrientDB Documentation, [Online]. Available: https://orientdb.com/docs/3.0.x/distributed/Replication.html

[102] Couchbase, “Cross Data Center Replication (XDCR),” *Couchbase Docs*, [Online]. Available: https://docs.couchbase.com/server/current/learn/clusters-and-availability/xdcr-overview.html

[103] Clavijero, “Tema 3.3: Procesamiento de consultas distribuidas,” *Curso de Bases de Datos*, [Online]. Available: https://cursos.clavijero.edu.mx/cursos/080_bdd/modulo3/contenidos/tema3.3.html

[104] IBM, “Replicación,” 2021. [Online]. Available: https://www.ibm.com/topics/data-replication

[105] T. Naeem, “Data Replication - An Overview and Benefits,” Astera, Apr. 16, 2024. [Online]. Available: https://www.astera.com/es/type/blog/data-replication/

[106] B. Priya, “¿Qué es la fragmentación de bases de datos?,” Geekflare Spain, May 14, 2024. [Online]. Available: https://geekflare.com/es/database-sharding/

[107] Erwin, “Denormalizing NoSQL database models.” [Online]. Available: https://bookshelf.erwin.com/bookshelf/public_html/12.5/Content/User%20Guides/erwin%20Help/Denormalizing_NoSQL_Database_Models.html

[108] A. Gaitonde, “Distributed transactions & two-phase commit - geek culture - Medium,” *Medium*, Mar. 24, 2021. [Online]. Available: https://medium.com/geekculture/distributed-transactions-two-phase-commit-c82752d69324

[109] W. Effelsberg and M. V. Mannino, “Attribute equivalence in global schema design for heterogeneous distributed databases,” *Information Systems*, vol. 9, no. 3–4, pp. 237–240, Jan. 1984, doi: 10.1016/0306-4379(84)90006-1.

[110] ArangoDB, “Sharding,” [Online]. Available: https://docs.arangodb.com/3.11/deploy/architecture/data-sharding/

[111] OrientDB, “Sharding · OrientDB Manual.” [Online]. Available: https://orientdb.com/docs/3.0.x/distributed/Distributed-Sharding.html

[112] Couchbase Docs, “Transactions,” https://docs.couchbase.com/server/current/learn/data/transactions.html

[113] E. Brewer, “Towards robust distributed systems (abstract),” Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 7–10, ACM, 2000. https://doi.org/10.1145/343477.343502

[114] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, “Ernest: Efficient performance prediction for large-scale advanced analytics,” USENIX Symposium on

Networked Systems Design and Implementation (NSDI), pp. 363–378, 2016. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/venkataraman

[115] W. Vogels, "Eventually consistent," Communications of the ACM, vol. 52, no. pp. 40–44, 2009. https://doi.org/10.1145/1435417.1435432

[116] D. Bermbach and S. Tai, "Eventual Consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior," Proceedings of the 6th Workshop on Middleware for Service

[117] Vogels, W. (2009). Eventually consistent. Communications of the ACM, 52(1), 40-44. https://dl.acm.org/doi/fullHtml/10.1145/1435417.1435432

8. Bibliografía.

Las siguientes fuentes fueron consultadas como material de referencia general durante el desarrollo de este trabajo, pero no se citan directamente en el texto principal.

[B1] M. T. Özsu y P. Valduriez, *Principles of Distributed Database Systems*, 4th ed. Cham, Switzerland: Springer, 2020.

[B2] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, CA, USA: O'Reilly Media, 2017.

[B3] P. A. Bernstein, V. Hadzilacos y N. Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, MA, USA: Addison-Wesley, 1987.

[B4] P. J. Sadalage y M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Upper Saddle River, NJ, USA: Addison-Wesley, 2013.

[B5] J. Gray y A. Reuter, *Transaction Processing: Concepts and Techniques*. San Mateo, CA, USA: Morgan Kaufmann, 1992.

[B6] H. Boral y D. J. DeWitt, "A methodology for database system performance evaluation," *ACM SIGMOD Record*, vol. 14, no. 2, pp. 176–185, jun. 1984. DOI: 10.1145/971697.602283

[B7] E. Brewer, "CAP twelve years later: How the rules have changed," *IEEE Computer*, vol. 45, no. 2, pp. 23–29, feb. 2012. DOI: 10.1109/MC.2012.37

[B8] Raspberry Pi Foundation, "Raspberry Pi 5 Product Brief," 2023. [En línea]. Disponible en: https://www.raspberrypi.com/products/raspberry-pi-5/

[B9] Facebook Database Engineering Team, "RocksDB: A High Performance Embedded Database," *RocksDB GitHub*, 2024. [En línea]. Disponible en: https://github.com/facebook/rocksdb