Evaluación comparativa de rendimiento de PostgreSQL y Cassandra en operaciones CRUD

Jaramillo Sandoval, Angélica Alexandra y Puchaicela Soliz, Wendy Lisseth

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Trabajo de integración curricular, previo a la obtención del título de Ingenieras en Tecnologías de la Información

Ing. Coronel Guerrero, Christian Alfredo, Mgtr.

10 de agosto del 2022

Reporte Verificación de Contenido

Trabajo_de_Integracion_202250_Jaramillo_Puchaicela.pdf

Scanned on: 3:10 August 11, 2022 UTC

Identical Words	247
Words with Minor Changes	36
Paraphrased Words	208
Omitted Words	0

Firmado electrónicamente por:
CHRISTIAN ALFREDO
CORONEL GUERRERO

Ing. Christian Coronel Guerrero

CI: 1714127139

Director

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Certificación

Certifico que el trabajo de integración curricular: "**Evaluación comparativa de rendimiento de PostgreSQL y Cassandra en operaciones CRUD**" fue realizado por las señoritas **Jaramillo Sandoval, Angélica Alexandra y Puchaicela Soliz, Wendy Lisseth,** el mismo que cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, además fue revisado y analizada en su totalidad por la herramienta de prevención y/o verificación de similitud de contenidos; razón por la cual me permito acreditar y autorizar para que se lo sustente públicamente.

Santo Domingo, 10 de agosto del 2022

Firmado electrónicamente por:
CHRISTIAN ALFREDO
CORONEL GUERRERO

Ing. Coronel Guerreo, Christian Alfredo, Mgtr

C.C: 1714127139

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Autorización de Publicación

Nosotros **Jaramillo Sandoval, Angélica Alexandra y Puchaicela Soliz, Wendy Lisseth,** con cédulas de ciudadanía n° 1720510112 y 2300420425, autorizamos a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de Integración curricular: **Evaluación comparativa de rendimiento de PostgreSQL y Cassandra en operaciones CRUD** en el repositorio institucional, cuyo contenido, ideas y criterios son de nuestra responsabilidad.

Santo Domingo, 10 de agosto de 2022

Jaramillo Sandoval, Angélica Jaramillo

C.C.: 1720510112

Puchaicela Soliz, Wendy Lisseth

C.C.: 2300420425

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Autorización de Publicación

Nosotros **Jaramillo Sandoval, Angélica Alexandra y Puchaicela Soliz, Wendy Lisseth,** con cédulas de ciudadanía n° 1720510112 y 2300420425, autorizamos a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de Integración curricular: **Evaluación comparativa de rendimiento de PostgreSQL y Cassandra en operaciones CRUD** en el repositorio institucional, cuyo contenido, ideas y criterios son de nuestra responsabilidad.

Santo Domingo, 10 de agosto de 2022

Jaramillo Sandoval, Angélica Jaramillo

C.C.: 1720510112

Puchaicela Soliz, Wendy Lisseth

C.C.: 2300420425

Dedicatoria

Esta tesis está dedicada principalmente a mis padres Freddy Jaramillo y Alexandra Sandoval por el gran apoyo incondicional que me han brindado en esta etapa de vida, por inculcar en mí el ejemplo de responsabilidad, perseverancia y respeto. Gracias a ustedes he logrado cumplir un sueño más.

A mis hermanos Byron y Freddy por sus consejos y palabras de aliento que me ayudaron a seguir con valentía en este proceso.

A todas las personas importantes en mi vida, familia, amigas e hijo, que me ayudaron a culminar este logro de vida con éxito, por estar conmigo en momentos difíciles, por su cariño y esfuerzo brindado cada día.

Angélica A. Jaramillo S.

Dedicatoria

Este trabajo de tesis va dedicado a Dios por darme las fuerzas necesarias cada día y guiarme en toda la carrera universitaria, a mis padres Angel Puchaicela y Carmen Soliz, puesto que sin su apoyo no hubiese alcanzado este logro. A mis hermanas y hermano que siempre han estado dispuestos a brindar su ayuda en cualquier dificultad.

Wendy L. Puchaicela S.

Agradecimiento

Mi profundo agradecimiento a mis padres y familiares por confiar en mí y por siempre creer que esta etapa de vida llegaría a su final, por ser el pilar fundamental que me ayudó a cumplir con éxito este trabajo.

A Fernando Savedra por ser una persona muy especial en mi vida, por apoyarme en cada momento que lo necesito, por enseñarme que gracias a la perseverancia se obtienen muchos logros en la vida.

A la universidad de la Fuerzas Armadas ESPE Santo Domingo, por abrirme las puertas y permitirme realizar este trabajo de titulación en su entidad.

A mis docentes, en especial a nuestro director de tesis el Ing. Christian Coronel quien, con su dirección, conocimiento, enseñanza, experiencia y dedicación permitió el desarrollo de este trabajo con gran profesionalismo.

De igual manera, mis agradecimientos a mi compañera de tesis Wendy Puchaicela por su amistad incondicional, gracias a su compromiso, paciencia y perseverancia se pudo lograr este trabajo anhelado.

Angélica A. Jaramillo S.

Agradecimiento

Agradezco a Dios por mantenerme fiel en mis principios, guiarme y por brindarme la sabiduría necesaria durante el transcurso de mi vida universitaria.

A mi familia, padres y hermanos por siempre estar presentes, siendo ese apoyo incondicional que me otorgaba fuerzas en cada dificultad que se presentaba.

A mis compañeros que poco a poco se han convertido en grandes amigos, por hacer esos días divertidos, quedando grandes recuerdos y anécdotas, en especial Angélica Jaramillo mi compañera de tesis, con quien he compartido desde un inicio hasta culminar esta travesía.

Por último, a cada docente por impartir sus conocimientos, en especial al ing. Christian Coronel nuestro tutor de tesis, quien siempre estuvo dispuesto a guiarnos e impartirnos las directrices necesarias en el desarrollo de este proyecto.

Wendy L, Puchaicela S.

Índice de contenidos

Carátula....1

Reporte Verificación de Contenido....2

Certificado del Director....3

Responsabilidad de autoría....4

Autorización de publicación....5

Dedicatoria....6

Agradecimiento....8

Índice de contenidos....10

Índice de Figuras....13

Índice de Tablas....16

Resumen....17

Abstract....18

Introducción....19

Capítulo I: Preliminares....21

- Identificación del problema....21
- Justificación....22
- Objetivos....23
  - Objetivo General....23
  - Objetivos Específicos....23
- Alcance....23

Capítulo II: Marco Teórico 24
Antecedentes Investigativos 24
Bases de Datos Relacionales 25
Tablas: Filas y Columnas 25
Propiedades ACID 27
PostgreSQL 27
Beneficios de utilizar PostgreSQL 28
Tipos de datos de PostgreSQL Añadido 28
Bases de datos No Relacional 30
Teorema CAP 32
Cassandra 34
Características 35
PostgreSQL vs Cassandra 35
Capitulo III: Metodología 37
Desarrollo del modelo YCSB 37
Fase de Carga 38
Fase Transaccional 38
Entorno de trabajo 39
Entorno de Trabajo PostgreSQL 40
Entorno de Trabajo Cassandra 40
Escenarios de trabajo 41

Instalación YCSB....41
Configuración de PostgreSQL....43
Configuración de Cassandra....45
Capitulo IV: Plan de Pruebas y Resultados....48
Plan de Pruebas y Resultados....48
Fase de Carga....48
Fase Transaccional....50
Rendimiento Core i7....61
Rendimiento RIG de Procesamiento....63
Análisis de Rendimiento....64
Resultados....79
Capítulo V: Conclusiones y Recomendaciones....81
Conclusiones....81
Recomendaciones....81
Bibliografía....83

Índice de Figuras

Figura 1. Estructura de Base de Datos Relacional....26

Figura 2. Propiedades de SGBD de NoSQL. 33

Figura 3. Arquitectura del modelo de Datos Apache Cassandra....34

Figura 4. Consola de comandos de Windows....42

Figura 5. Instalación de YCSB....42

Figura 6. Ingresar a la ruta YCSB desde el símbolo del sistema CMD....43

Figura 7. Parámetros de PostgreSQL....44

Figura 8. Estructura de la Base de Datos PostgreSQL 44

Figura 9. Generación de tablas creadas en PostgreSQL....45

Figura 10. Parámetros de Cassandra....46

Figura 11. Estructura de la Base de Datos Cassandra 46

Figura 12. Datastax DevCenter de Cassandra....47

Figura 13. Tiempo de ejecución genera Core i7 49

Figura 14. Fase de Carga RIG de procesamiento....50

Figura 15. Latencia para Escenario A en Core i7....51

Figura 16. Latencia para Escenario B en Core i7....52

Figura 17. Latencia para Escenario C en Core i7 53

Figura 18. Latencia para Escenario D en Core i7 54

Figura 19. Latencia para Escenario E en Core i7....55

Figura 20. Latencia para Escenario F en Core i7....56

Figura 21. Latencia para Escenario A en RIG de procesamiento....57

Figura 22. Latencia para Escenario B en RIG de procesamiento....58

Figura 23. Latencia para Escenario C en RIG de procesamiento....59

Figura 24. Latencia para Escenario D en RIG de procesamiento....59

Figura 25. Latencia para Escenario E en RIG de procesamiento.....60

Figura 26. Latencia para Escenario E en RIG de procesamiento.....61

Figura 27. Rendimiento por cada carga de trabajo en Core i7 62

Figura 28. Rendimiento por cada carga de trabajo en RIG de procesamiento 63

Figura 29. Uso de Memoria Cassandra 64

Figura 30. Uso de Memoria PostgreSQL 65

Figura 31. Consumo de Disco PostgreSQL 65

Figura 32. Consumo de Disco Cassandra 66

Figura 33. Uso de Memoria Cassandra 66

Figura 34. Uso de Memoria PostgreSQL 67

Figura 35. Consumo de Disco PostgreSQL 68

Figura 36. Consumo de Disco Cassandra 68

Figura 37. Uso de Memoria PostgreSQL 69

Figura 38. Uso de Memoria Cassandra 69

Figura 39. Consumo de Disco PostgreSQL 70

Figura 40. Consumo de Disco Cassandra 70

Figura 41. Uso de Memoria Cassandra 71

Figura 42. Uso de Memoria Cassandra 71

Figura 43. Consumo de Disco Cassandra 72

Figura 44. Consumo de Disco PostgreSQL 73

Figura 45. Uso de memoria Cassandra 73

Figura 46. Uso de memoria PostgreSQL 74

Figura 47. Consumo de Disco PostgreSQL 75

Figura 48. Consumo de Disco Cassandra 75

Figura 49. Uso de Memoria Cassandra 76

Figura 50. Uso de Memoria PostgreSQL 77

Figura 51. Consumo de Disco Cassandra 78

Figura 52. Consumo de Disco PostgreSQL 78

Índice de Tablas

Tabla 1. Tipos de datos en PostgreSQL.....28

Tabla 2. Comparativa de funciones de PostgreSQL y Cassandra.....36

Tabla 3. Especificaciones de Computador i7.....39

Tabla 4. Especificaciones de RIG de procesamiento 40

Tabla 5. Análisis comparativo PostgreSQL y Cassandra.....80

Resumen

En la actualidad, las bases de datos relacionales son mucho más utilizadas frente a las no relacionales, debido al gran crecimiento de Internet, las organizaciones necesitan de un gestor de bases de datos que brinde eficiencia tanto en constancia, durabilidad, rendimiento y estabilidad frente a cualquier área empresarial, capaz de almacenar grandes cantidades de datos. Cassandra es altamente comparado con diferentes gestores de base de datos NoSQL, debido a su alta posibilidad de no fallos al almacenar una gran cantidad de datos, por otra parte, tenemos a PostgreSQL, un gestor de base de datos relacional, reconocido por brindar confiabilidad e integridad de datos gracias a la arquitectura que conlleva. El objetivo del siguiente proyecto es mediante un estudio comparativo, analizar el rendimiento de dos bases de datos PostgreSQL y Cassandra; en escritura, lectura y proceso de datos en presencia de una arquitectura en común tanto en hardware como software, realizado bajo las operaciones CRUD, las mismas que estarán basadas en un millón de datos y 16 hilos de trabajo frente al framework YCSB, herramienta que permite el análisis de rendimiento en seis escenarios con distintas características, soportado por los gestores de base de datos lineal (PostgreSQL) y no Lineal (Cassandra). Los resultados demostraron a PostgreSQL con mucho más desempeño que Cassandra ante cada escenario, en vista a la fase de carga de datos y fase transaccional.

Palabras claves: PostgreSQL, Cassandra, Rendimiento, Escalabilidad, SGBD, YCSB.

Abstract

Nowadays, relational databases are much more used than non-relational ones, due to the great growth of the Internet, organizations need a database manager that provides efficiency in terms of consistency, durability, performance and stability to any business area, capable of storing large amounts of data.Cassandra is highly compared to different NoSQL database managers, due to its high possibility of no failures when storing a large amount of data, on the other hand, we have PostgreSQL, a relational database manager, recognized for providing reliability and data integrity thanks to its architecture.The objective of the following project is through a comparative study, to analyze the performance of two databases PostgreSQL and Cassandra; in writing, reading and processing data in the presence of a common architecture in both hardware and software, performed under CRUD operations, which will be based on a million data and 16 threads against the YCSB framework, a tool that allows performance analysis in six scenarios with different characteristics, supported by linear database managers (PostgreSQL) and non-linear (Cassandra). The results showed PostgreSQL with much better performance than Cassandra in each scenario, in view of the data load phase and transactional phase.

Keywords: PostgreSQL, Cassandra, Performance, Scalability, DBMS, YCSB.

Introducción

Actualmente, los datos en la tecnología son el eje principal dentro de una organización y, debido al avance tecnológico todo a nuestro alrededor genera datos e información, provocando el auge de herramientas que permitan procesarlas. A través, de la abstracción de datos se pueden generar varios procesos para forjar una buena gestión de información, por medio de los sistemas gestor de bases de datos (SGBD) se puede obtener un buen manejo de las mismas, es decir permite realizar varias funciones de manera más eficiente.

Los sistemas SQL tuvieron sus inicios en la década de los 70, gracias a los investigadores llamados Ray Boynce y Ted Codd, quienes en busca de un sistema de almacenamiento de datos indefinidamente, dieron a conocer los principios de la estructura de datos persistentes. Por otra parte, los sistemas NoSQL surgieron en la década de los 90 debido a la gran expansión de internet, y gracias a la búsqueda de grandes empresas desarrolladoras de software que usaban algunos sistemas de gestión de bases de datos relacionales (Tatsis, 2022)

Las bases de datos relacionales demuestran ser mucho más complejas que las no relacionales debido a su estructura y lógica de procesamiento, pese a eso demuestran gran coherencia de datos, aunque son propensas a presentar un bajo rendimiento en el sistema. Debido a esta problemática surge la necesidad de beneficiarse de los sistemas de bases de datos NoSQL, que junto con las operaciones CRUD (Crear, Leer, Actualizar, Eliminar), hacen más factible el amplio almacenamiento y rendimiento en el sistema (Yarabarla, Learning Apache Cassandra, 2017).

Existen varios tipos de lenguajes de programación, Frameworks y herramientas que soportan los gestores PostgreSQL y Cassandra. Entre estos se encuentra Java, CQL (Cassandra Query Language) los cuales requieren de drivers, documentos, gráficos, columnas, clave-valor u otras tecnologías para tener un alto rendimiento, antes, durante y después de realizar pruebas de manejo de información. Además, existen estudios comparativos donde se evidencia el tipo de SGBD cuente con mejor rendimiento y facilidad de manejo según las necesidades y acoplamientos que la organización requiera.

Capítulo I: Preliminares

Identificación del problema

Las organizaciones requieren de unas bases de datos que cumplan con los requerimientos funcionales del cliente, que cuenten con un alto rendimiento, que permita la gestión de procesos de manera eficaz. La base de datos PostgreSQL, es un SGBD relacional, Open Source, que hace funcional a todas las construcciones SQL, debido a su estructura es considerada una de las más fuerte en el mercado, otorgando un alto grado de confiabilidad, integridad y precisión en análisis (Manuela et al., 2022).

En cambio, la base de datos Cassandra, es un software Open Source comercializado por DataStax, en base al teorema CAP (Consistencia, Alta disponibilidad, Tolerancia a patrones), este se encuentra situado con una alta disponibilidad y tolerancia a particiones, es decir, no sigue el patrón maestro esclavo, siendo altamente escalable de forma horizontal (Seco, 2017).

Las bases no relacionales pueden ser una alternativa de solución para las organizaciones con grandes volúmenes de datos. En base a ello, se implementó un estudio comparativo entre PostgreSQL (SGBD) y Apache Cassandra (NoSQL). No obstante, se han encontrado estudios comparativos de igual similitud en escenarios relacionados con ambas bases de datos, de las cuales con variada información relevante se obtuvo una clara perspectiva hacia el desarrollo de la investigación.

En consecuencia, es fundamental que las organizaciones cuenten con estudios comparativos, que permitan seleccionar una base de datos, segura. Estos estudios deben contar con un análisis comparativo de pruebas, carga de datos y latencia, siendo estas consideraciones de mayor interés para el análisis de rendimiento de ambos enfoques. Asimismo, la importancia de aclarar cuáles son las bases de datos con mejor rendimiento.

Justificación

Las grandes empresas manejan una gran cantidad de datos donde el rendimiento del sistema es de suma importancia, al no escoger un SGBD adecuado se puede generar una decadencia en el manejo del sistema y grandes pérdidas de datos e incluso ingresos económicos. Esto conlleva a considerar un cambio desde un SGBD SQL a un NoSQL o viceversa según la necesidad de cada organización. Las bases de datos NoSQL han llegado a forjar gran popularidad ante grandes empresas como Twitter, Facebook y Google debido al amplio almacenamiento que ofrece (Salazar, 2014). Por lo tanto, es indispensable realizar un análisis y observar que SGBD es más viable a implementar.

El rendimiento que otorgue el SGBD ahorrará tiempo tanto en limpieza de datos como en el manejo de las operaciones CRUD, un sistema puede ser mínimamente lento, pero ya ocasiona un disgusto hacia el cliente, por lo tanto, es importante tener en claro los objetivos que como empresa ayudará a elegir de manera correcta del SGBD, así como el Framework o herramienta que le acompañan a su desarrollo (Kozma & Morschheuser, 2019).

En la historia tecnológica, se considera que las bases de datos relaciones son más utilizadas, gracias a la variedad de características que hacen que su manejo sea más fiable y fácil. Pero no son muy eficientes para formar un análisis en grandes volúmenes de datos y realización de operaciones conjuntas (Veronica & Bernardino, 2013).

El proyecto presente permitirá a profesionales del área de TI (Tecnologías de la Información), principalmente a los administradores de bases de datos, preferir que SGBD se adapte mejor a sus necesidades entre PostgreSQL y Cassandra.

Objetivos

Objetivo General

Evaluar una comparativa de rendimiento de PostgreSQL y Cassandra en operaciones CRUD

Objetivos Específicos

- Realizar una evaluación comparativa de PostgreSQL y Cassandra en operaciones CRUD utilizando YCSB Framework.
- Comparar el enfoque relacional y no-relacional mediante una exhaustiva revisión de la literatura.
- Proponer escenarios de pruebas basadas en operaciones CRUD utilizando YCSB Framework.
- Obtener mediciones de rendimiento con respecto al tiempo general de ejecución, latencia y tasa de transferencia efectiva.

Alcance

Con el desarrollo de la presente investigación, se espera obtener un análisis comparativo entre PostgreSQL y Cassandra, además de realizar pruebas de rendimiento en base a las operaciones CRUD, las cuales serán analizadas mediante escenarios de evaluación, en la cual se experimentan a través de 2 fases; fase de carga y fase transaccional. Luego de esto, se estima comparar las Bases de Datos Relacionales y NoSQL sobre los escenarios propuestos en el Framework YCSB, el rendimiento de las bases de datos en las diferentes fases y la latencia que arroja cada base de datos en la fase transaccional.

Capítulo II: Marco Teórico

Antecedentes Investigativos

Las bases de datos son la pieza fundamental en las tecnologías de la información, además de ser un sistema que almacena grandes cantidades de datos e información, tanto para las organizaciones como para los desarrolladores. A continuación, se presentan estudios relacionados con el problema descrito en el Capítulo I, en el cual se obtendrá información relacionada con el presente proyecto.

(Lima et al., 2018) en su artículo “Estudio Comparativo entre PostgreSQL (SGBD) e apache Cassandra (No SQL)” pretende explicar las características más relevantes de las bases de datos NoSQL y las bases de datos relacionales, para obtener una comparación de archivos de texto con 3 y 6 mil datos, utilizando Java como lenguaje de programación.

En la tesis de (Kozma & Morschheuser, 2019) titulada “Cloud Service Enviorenment PostgreSQL vs. Cassandra” se busca comparar dos bases de datos diferentes, PostgreSQL y Apache Cassandra, con un enfoque en el rendimiento de latencia, sobre 4 operaciones estándar crear, leer, actualizar y escanear. Utilizando la herramienta de benchmarking “yahoo! Punto de referencia del servicio en la nube”.

(Alyasiri, Sahi, & AL-Khafaji, 2022) en su árticulo “NoSQL: Will it be alternative to relational database? Mysql vs MongoDB comparison”, presenta una comparativa entre una base de datos Mysql y MongoDB mediante la instalación de servidores, abordando los temas de comparación en características generales, calidad de consultas, comparación de popularidad y según el teorema CAP.

En el artículo de (Christian et al.) “Insertion speed of indexed spatial data: comparing MySQL, PostgreSQL and MongoDB”, pretende demostrar que SGDB brinda mayor rendimiento para la inserción de datos espaciales en dispositivos IoT, las pruebas fueron realizadas mediante el método cuantitativo, utilizando contenedores Docker.

Teniendo en cuenta los artículos e investigaciones citadas anteriormente, se destacan términos como: NoSQL, SQL, rendimiento y SGBD. Por demás, se extrajo información de documentos actualizados en su gran mayoría del presente año (2022), destacando a los exploradores IEEE Explorer, Google academics, Scielo, ITU-T donde se extrajo información de mayor índole.

Bases de Datos Relacionales

Según Amazon Web Service (AWS, n.d.), define a una base de datos relacional como una colección de elementos de datos estructurados con relaciones predefinidas, organizados mediante tablas, con filas y columnas. Donde, las columnas guardan un determinado tipo de datos y las filas equivalen a una recolección de las relaciones de un objeto. Además, las bases de datos relacionales se manejan en lenguaje SQL así se logra acceder y manejar los datos brindando grandes beneficios, en resumen, recuperación e informes de datos (Nishtha et al., 2019).

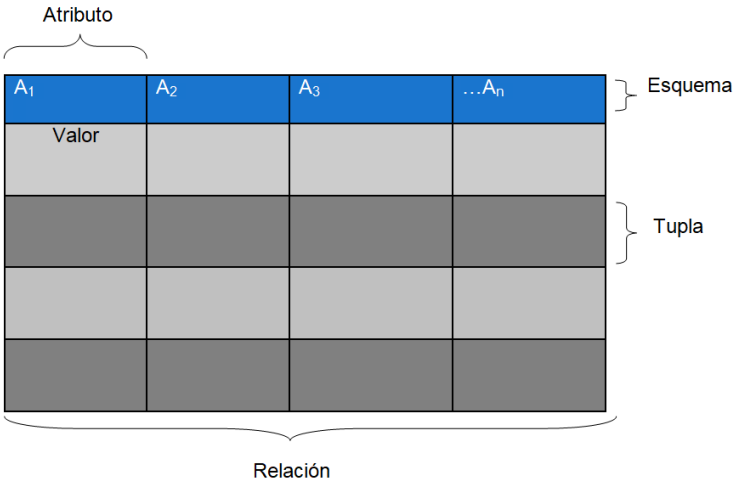
Tablas: Filas y Columnas

EF Codd inventor de las bases de datos, planteo la idea de en lugar de almacenar datos con estructuras jerarquicas; organizar los datos mediante tablas con filas y columnas. Dentro de las bases de datos relaciones cada tabla o relación se complementa con más de una categoría de datos en columnas o tambien conocido como atributos. En cambio, las fila, registro o tupla, esta definida de una instancia única. Ademas, cada tabla contiene una clave primaria con el objetivo de identificar la informacion de la tabla (Lutkevich & Biscobing, 2021)

Las relaciones trabajan mediante el uso de claves externas que llegan a vincularse por medio de un campo externo gracias a la clave principal.

Figura 1.

Estructura de Base de Datos Relacional



Nota. Se muestra la estructura de una base de datos relacional, donde las columnas se representan por atributos y las filas por registros o tupla. Tomado de (Ayudaley, 2020)

El objetivo de las bases de datos relacionales es la posibilidad de formar información relevante al unir tablas, estas uniones facilitan comprender las relaciones que existen entre los datos y las conexiones con las tablas. Permiten concretar operaciones de manera más fácil y rápida para los analistas de datos como; ordenar resultados por nombre, fecha, establecer rangos de búsqueda, entre un sinnúmero de funciones. Dichas características hacen que las bases de datos relacional sea la más popular en las organizaciones a nivel de forjar consultas. (IBM, 2019)

Propiedades ACID

Dentro de las bases de datos relacionales existen propiedades concretas al realizae transacciones como la atomicidad, consistencia, aislamiento, y durabilidad, conocido como ACID .

- **Atomicidad:** Especifica los componentes que existen dentro de una transaccion en una base de datos completa.
- **Coherencia:** Especifica reglas a mantener los datos en un estado correco al terminar una transacción.
- **Aislamiento:** Conserva un efecto de transacción invisible ante los demas hasta implicarse.
- **Durabilidad:** Permite que los datos se conviertan permanentes una vez confrimada la transacción.

PostgreSQL

Según (Nayantara et al., 2022), PostgreSQL también conocido como Postgres es un SGBD relacional Open Source de apoyo para la construcción de sentencias SQL. Famosamente conocido por su alto grado para confiabilidad, integridad y precisión de datos, ante el uso de varios sistemas operativos. Además, maneja una gran parte de sentencias SQL y ofrece muchas características más, como Consultas SQL complejas, Forein Keys, Triggers, View, SQL Sub-selects, Nulti-version concurrency, Streaming Replication.

PostgreSQL es una base de datos con alto nivel de aprobación ante los SGBD existentes. Incluso, demuestra cómo cada día se encuentra en busca de mejoras para cubrir las necesidades de sus clientes. Si cada organización plantea los requerimientos necesarios, abordando cada necesidad PostgreSQL, garantiza un rendimiento exitoso así existan cambios en el diseño del sistema al pasar los años (Panchenko, 2021).

PostgreSQL, se destaca debido a distintas características que ofrece entre ellas tenemos:

- Tipos de datos
- Funciones
- Operadores
- Funciones Agregadas
- Métodos de índice

Beneficios de utilizar PostgreSQL

Reduce costos, permitiendo la suscripción de forma gratuita a organizaciones ofreciendo un manejo robusto y personalizable. Cada organización puede manipular las funciones que ofrece a su conveniencia debido a que es un software de código abierto.

Cuenta con un registro de escritura anticipada otorgando un alto nivel de tolerancia ante fallas, sin dar afectaciones y logrando ser ejecutadas en sitios web dinámicos y aplicaciones web. Permite la incorporación de objetos geográficos, por ello es posible usarlo para servicios orientado en ubicación, almacén de datos geoespaciales e información geográfica (Peterson, 2020).

Tipos de datos de PostgreSQL Añadido

Tabla 1.

Tipos de datos en PostgreSQL

Tipo de dato	Descripción
Bigint	Entero con ocho byts con signo
Bigserial	Entero de ocho byts de incremento automático
Bit [(n)]	Cadena de bits de longitud fija

Bit varying [(n)]	Cadena de bits de longitud variable
Boolean	Booleano lógico
Box	Caja rectangular en un plano
Bytea	Datos binarios
Character [(n)]	Cadena de caracteres de longitud fija
Character varying [(n)]	Cadena de caracteres de longitud variable
Cidr	Dirección de red IPv4 o IPv6
Circle	Circulo en plano
Date	Calendario (año/mes/dia)
Doublé precisión	Numero de punto flotante
Inet	Dirección host IPv4 o IPv6
Integer	Entero
Interval [fields][(p)]	Tiempo span
Json	Textual JSON DATA
Jsonb	Binario JSON data
Line	Infinito línea en un plano
Lseg	Línea en un plano
Macaddr	Mac address
Money	Correncia de moneda
Numeric [(p, s)]	Número exacto de precisión
Path	Geométrica path en plano
Pg_lsn	Numero de secuencia de registro de PostgreSQL
Point	Punto geométrico en un plano
Polygon	Trayectoria geométrica cerrada en un plano

Real	Numero de punto flotante de presición simple (4 bytes)
Smallint	Entero de dos bytes con signo
Smallserial	Entero de dos bytes de incremento automático
Serial	Entero de cuatro bytes con incremento automático
Text	Cadena de caracteres de longitud variable
Time[(p)][without Time zone]	Hora del dia (sin zona horaria)
Time[(p)] with Time zone	Hora incluida zona horaria
Timestamp	Fecha y hora, sin zona horaria
Timestamp	Fecha y hora, incluida sin zona horaria
Tsquery	Consulta de búsqueda de texto
Tsvector	Documento de búsqueda de texto
Txid_snapshot	Instantánea de ID de transacción a nivel de usuario
Uuid	Identificador unico universal
xml	Datos xml

Nota: Se presenta todos los datos que ofrece PostgreSQL para el manejo de su entorno

Bases de datos No Relacional

En la actualidad muchas de las aplicaciones más utilizadas hoy en día utilizan bases de datos NoSQL integradas desde su arquitectura. No obstante, a medida que los datos incrementan las bases de datos deben aumentar su capacidad de almacenamiento. Por ello, algunos de los investigadores o desarrolladores dieron inicio a soluciones con respecto a bases de datos No Relacional.

NoSQL (Not Only SQL), tuvo sus inicios en el siglo XXI cuando el mundo empezó con la creación de bases de datos relacionales, es decir, atender a millones de usuarios conectados y ahora con las soluciones propuestas de los desarrolladores las bases de datos NoSQL son capaces de atender a miles de millones de usuarios conectados entre dispositivos móviles, dispositivos inteligentes y dispositivos de automóviles, entre otros (Vaish, 2013).

Para no perder la capacidad de almacenar los datos, surge la idea de escalar los datos horizontalmente. El escalado horizontal se puede definir como un mecanismo potente en base al manejo de sentencias, con la capacidad de mejorar el rendimiento en grandes cantidades de almacenamiento (Veronica & Bernardino, 2013).

En la siguiente sección se describirán las 4 categorías principales de bases de datos NoSQL (Aniceto et al., 2015):

- **Key-value:** Bases de datos muy parecidas a un diccionario, es decir, los datos están aislados e independiente de otros.
- **Column family**: Bases de datos que almacenan sus datos en columnas, en lugar de filas como las bases de datos tradicionales.
- **Graph databases:** Bases de datos que se encuentran formados por nodos y aristas, es decir, las relaciones entre nodos las conforman las aristas. La mayor potencia que ofrecen estas bases de datos es el almacenamiento tanto en nodos como en aristas.
- **Document:** Bases de datos que almacenan los datos en forma de documentos. Los documentos son colecciones de atributos, donde los documentos pueden ser almacenados en formato JSON con la flexibilidad de ingresar muchos JSON sin afectar la base de datos.

Teorema CAP

A través del teorema CAP, Cassandra permite elegir qué punto de reparación entre la consistencia y tolerancia a fallos se puede obtener. Sin embargo, algunos sistemas distribuidos se encuentran inmersos en esta lógica; es decir, solo podrá entregar dos de las tres características que ofrece.

El teorema CAP está basada en tres características de un sistema distribuido, de las que hace mención a su nombre:

Consistencia (C): En base al teorema CAP, se refiere a los efectos de una operación de escritura; es decir, los datos deben estar siempre disponibles a los usuarios o clientes que realicen las operaciones de lectura.

Disponibilidad (A): En base al teorema CAP, se refiere a que, el sistema debe estar siempre abierto para el acceso a los clientes en operaciones de lectura y escritura.

Tolerancia a Particiones (P): En base al teorema CAP, se refiere a un continuo proceso del sistema, así un nodo falle o no se conecten entre sí.

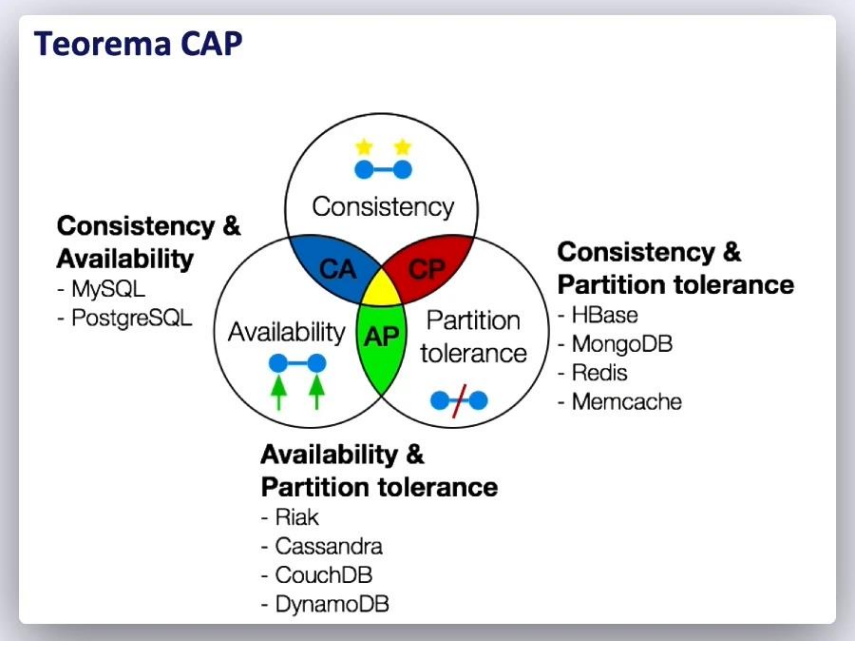
El teorema CAP establece y asegura que solo dos de sus tres contribuyentes pueden entregarse en cualquier momento, como son las categorías:

- **Sistema CP** (Consistencia y Tolerancia a la partición)
- **Sistema AP** (Disponibilidad y Tolerancia a la partición)
- **Sistema CA** (Consistencia y Disponibilidad)

Cassandra al contar con una arquitectura no maestro y múltiples puntos de fallos, se estipula que se encuentra en la categoría AP, proporcionando disponibilidad y tolerancia a fallo sacrificando un poco la consistencia. Sin embargo, Cassandra al no tener nodos principales los demás deben estar altamente disponibles para las operaciones, utilizando lo mínimo de su consistencia para permitir a los usuarios escribir en los nodos en cualquier momento.

Figura 2.

Propiedades de SGBD de NoSQL.



Nota**.** Se muestra como un sistema distribuido no puede conseguir las tres siglas y las bases de datos que se encuentran en cada una de las categorías. Tomado de (Mesa, 2019).

Cassandra

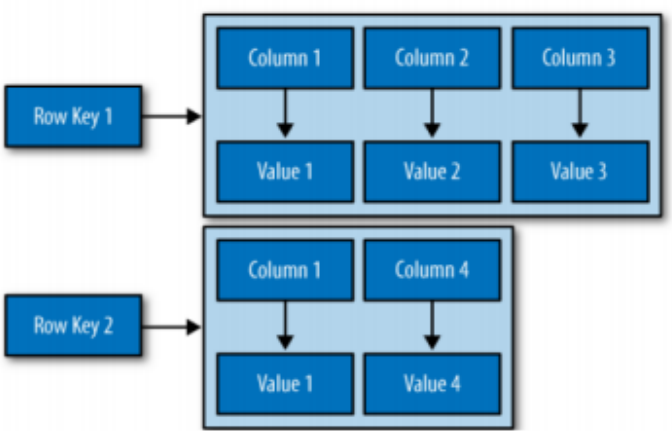
Definición

De acuerdo con el libro (Yarabarla, Learning Apache Cassandra, 2015) Cassandra es una base de datos OpenSource orientada a sistemas y completamente distribuida, con una capacidad de ofrecer a los usuarios una escalabilidad con tolerancia a fallos en comparación con otras bases de datos relacionales, es decir, está diseñada para almacenar grandes volúmenes de datos e información. Por ello, algunas empresas como Netflix, Apple e Instagram utilizan esta base de datos para almacenar su información.

Cassandra es considerada como una base de datos no relacional, formando parte de la familia de las columnas, es decir, almacena sus datos en columnas con la principal diferencia de otras tradicionales, es que, los datos pueden ser estructurados o no estructurados. Además, de no contar con un clúster no maestro, esto beneficia a Cassandra a manera de que, si un clúster o nodo falla, no afecta al sistema debido a que los demás nodos toman una copia del clúster maestro con la misma capacidad de lectura y escritura.

Figura 3.

Arquitectura del modelo de Datos Apache Cassandra



Nota. Se muestra como se encuentran estructurados los datos en la arquitectura de la base de datos Apache Cassandra. Tomado de (Madisyn, 2012)

Características

En el transcurso de investigación se ha ido mencionando algunas características que hacen de Cassandra un sistema de gestor de base de datos más potente para su clasificación. Es momento de entrar a detalle con cada una de ellas.

Recordando, Cassandra al ser de la familia de columnas proporciona el modelo P2P (peer-to-peer) para distribuir los datos. Es por ello, que todos los nodos creados serán iguales, contando con la seguridad de que Cassandra es la que se encarga de replicar los nodos a través del modelo P2P para que no exista una tolerancia a fallos. Es decir, agregar varios nodos al clúster replicados con tokens no ocasionará que el sistema falle, debido a que, si un nodo falla los demás están replicados con la misma información.

Escalabilidad horizontal

La escalabilidad horizontal de Cassandra se refiere a la capacidad de rendimiento de una base de datos, agregando más servidores un clúster determinado.

- Cuenta con su **sistema distribuido**: Se refiere a la ejecución de un grupo de nodos en varios servidores de la base de datos.
- **Alta disponibilidad**: Si un nodo falla el sistema no se verá afectado, debido a que las escrituras se transmiten a los demás del clúster.

PostgreSQL vs Cassandra

Obtenida información relevante sobre las funciones que ofrecen las bases de datos PostgreSQL (SGBD) y Cassandra (NoSQL), es momento de averiguar cuál de las dos mantiene características que sean fáciles de encajar. La siguiente tabla indica las funciones de cada una de ellas:

Tabla 2.

Comparativa de funciones de PostgreSQL y Cassandra

Funciones	PostgreSQL	Cassandra
Lenguaje de consulta	SQL	CQL
Modelo de datos	Relacional Base de datos	No relacional Base de datos Columnas
Ordenación rápida de resultados	Si	Si
Consistencia inmediata	Si	Configurable a nivel de consulta
Escrituras de alto rendimiento	No	Si
Fragmentación transparente	No	Si
Uniones relacionales	Si	No
Índices secundarios	Si	Si
Registros Estructurados	Si	Si
Colecciones de escritura discreta	Si	Si
Ningún Punto único de falla	No	Si

Se observa como Cassandra tiene algunas similitudes con PostgreSQL en cuanto a funciones. Pero, en comparación Cassandra cuenta con la disponibilidad y escalabilidad que puede ser de gran beneficio para las organizaciones que requieran utilizar la base de datos Cassandra.

Capitulo III: Metodología

Desarrollo del modelo YCSB

Yahoo Cloud Serving Benchmark es un conjunto de herramientas reconocido para medición del rendimiento en distintos SGBD abarcando diferentes modelos de estructura y otros lenguajes de consulta (Cooper et al., 2010).

YCSB ofrece resultados concisos en sus comparaciones, ya que permite realizar pruebas con elementos comunes dentro de los SGBD. Además, brinda sus resultados en formatos de texto perfectamente estructurados para el análisis de cada carga de trabajo, resumiendo el rendimiento de las operaciones, latencia y tiempo de ejecución.

YCSB primero inicia con la fase de carga mediante un conjunto de datos insertándolos dentro de una base de datos, luego pasa por la fase transaccional. Cada escenario a evaluar se denomina Workloads y cada dato cargado está formado por un conjunto de caracteres que son registrados aleatoriamente, estos registros ocupan una cantidad de 1KB conformada por la clave primaria y mediante los campos creados.

Las cargas de trabajo se basan en las operaciones CRUD, en cada escenario es posible establecer el número de operaciones y la cantidad de hilos con que se vaya a trabajar. YCSB establece seis escenarios de estimación que son:

- **Workload A** (Actualizar gran carga de trabajo): Este escenario realiza una combinación de 50/50, 50% en lectura y 50% escritura.
- **Workload B** (Lectura de carga de trabajo): Este escenario realiza una combinación de 95% lectura y 5% actualización.
- **Workload C** (Solo lectura): Este escenario realiza 100% de lectura.
- **Workload D** (Leer última carga de trabajo): Este escenario realiza una combinación de 95% de lectura de registros nuevos y 5% de inserción.

- **Workload E** (Distancias Cortas): Este escenario consulta rangos cortos de registros, en lugar de individuales. Realiza una combinación de 95% scanner y 5% insertar.
- **Workload F** (Lectura-modificación-escritura): Este escenario el cliente leerá un registro, lo modificará y escribirá los cambios. Realiza una combinación de 50% de lectura y 50% de lectura-modificación-escritura.

Fase de Carga

En esta fase YCSB se encarga de insertar los datos dentro de la base de datos creada, en esta sección se establece los parámetros de ejecución con los que se trabaja.

- **Recordcount**: Representa la cantidad de registros que serán insertados en la base datos.
- **Threads**: La cantidad de hilos que vaya a manipular los datos con los métodos CRUD.
- **S**: Presenta el estado de tiempo en que se encuentre la carga u transacción.
- **Comando de carga**: .\bin\ycsb load [GESTOR A UTILIZAR] -P .\workloads\workloada -P .\[DIRECION DE ARCHIVO DE PROPIEDADES] > [ARCHIVO DONDE GUARDA LOS DATOS].txt

Fase Transaccional

Esta fase se encarga de realizar las operaciones CRUD utilizando los datos insertados en la base de datos y según las funciones de cada escenario.

- **Operationcount**: Representa la cantidad de operaciones de carga de trabajo que va a realizar YCSB entre las funciones CRUD.

- **Threads**: La cantidad de hilos que vaya a manipular los datos con los métodos CRUD
- **S:** Presenta el estado de tiempo en que se encuentre la carga u transacción.
- **Comando para Transacción**:.\bin\ycsb run [GESTOR A UTILIZAR] -P .\workloads\workloada -P .\[DIRECION DE ARCHIVO DE PROPIEDADES] > [ARCHIVO DONDE GUARDA LOS DATOS].txt

Entorno de trabajo

Para el análisis de rendimiento en YCSB, se optó por el uso de un computador Core i7 y a su vez se realizó la estabilidad hacia un RIG de procesamiento. Dentro del desarrollo se utilizó un disco de estado sólido con 228 GB, e instalando en un sistema operativo Windows 10 pro. Estas opciones se escogieron de manera subjetiva tomando en consideración la forma de manejo, instalación e información en cuanto a las herramientas utilizadas.

Tabla 3.

Especificaciones de Computador i7

Especificaciones del procesador core i7
Procesador Intel Core I7-6700 CPU (2.80 GHz 2.81 GHz)
RAM 8,00 GB DDR4 (7,88 GB usable)
SO Windows pro de 64 bits, procesador basado en x64

Nota. Se muestra las especificaciones del computador Core i7 con el disco SSD que fue empleado para la instalación de YCSB, en PostgreSQL y Cassandra:

Tabla 4.

Especificaciones de RIG de procesamiento

Especificaciones del dispositivo	Características
Procesador	AMD Ryzen Threadripper 2920X 12-Core Processor (3.50 GHz)
RAM	16,00 GB DDR4
SO	Windows pro de 64 bits, procesador basado en x64

Nota. La tabla presenta las especificaciones del RIG de procesamientos con el disco SDD.

Entorno de Trabajo PostgreSQL

Se realizó la instalación de PostgreSQL quien otorga el servicio PgAdmin el entorno gráfico de PostgreSQL, quien permite la visualización de los procesos en carga y transacción de los escenarios. Los programas necesarios instalados y cada una de sus versiones utilizadas son:

- PostgreSQL versión 14.2.2
- Java JDK Versión 8u251
- PostgreSQL JDBC 4.2 Driver

Para el uso de Java JDK es importante instalar y establecer la respectiva variable de entorno para el correcto uso de trabajo

Entorno de Trabajo Cassandra

Para la instalación de Cassandra se hizo el uso de DataStax para el manejo un entorno gráfico y quien presenta el análisis de procesos en carga y transacción. Los programas instalados son los siguientes:

- RAR de Cassandra versión 3.11.10
- JDK Versión 8u251
- Python verion 2.7.18
- DevCenter DataStax Cassandra version 1.6

Escenarios de trabajo

YCSB cuenta distintas especificaciones para iniciar su herramienta, como tal hemos establecido los siguientes pasos para lograr su manejo en pruebas de rendimiento entre PostgreSQL y Cassandra:

1. Establecer las Bases de datos para Analizar rendimiento (PostgreSQL – Cassandra).
2. Utilizar cada escenario que otorga YCSB para medir rendimiento.
3. Establecer los parámetros necesarios a utilizar.
4. Iniciar fase de carga.
5. Iniciar fase Transaccional.

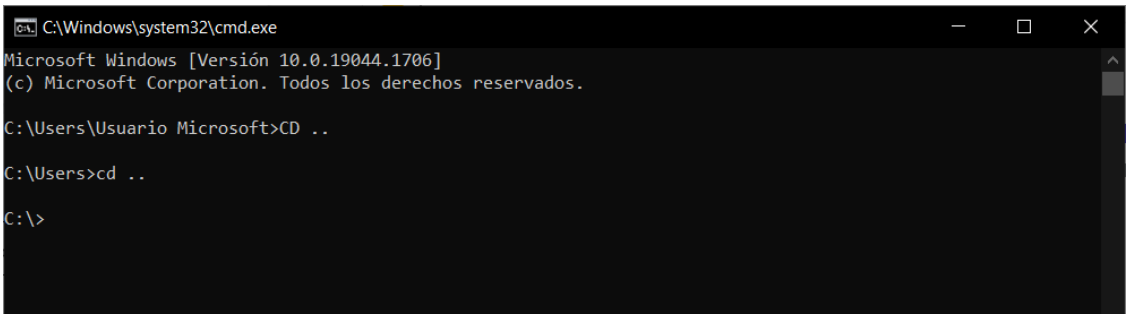
A continuación, se realiza paso a paso la instalación de YCSB, junto con la configuración necesaria de las bases de datos PostgreSQL y Cassandra.

Instalación YCSB

Abrir el CMD del equipo, colocar *cd..* y dar clic para salir del usuario, realizar este paso hasta que la dirección se encuentre únicamente en *C:*\>, como muestra la figura 3.

Figura 4.

Consola de comandos de Windows

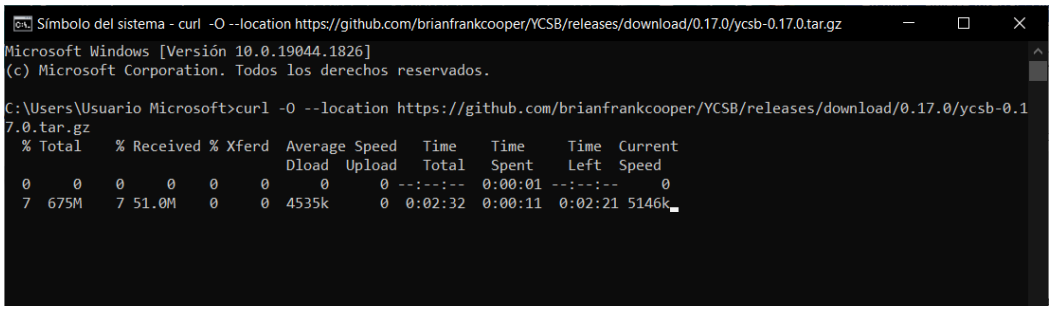


Nota. Se muestra cómo debe estar la consola de comando para ingresar al directorio de YCSB.

Se inicia la descarga con el comando: curl -O –location https://github.com/brianfrankcooper/YCSB/releases/download/0.17.0/ycsb-0.17.0.tar.gz. Posteriormente se descomprime el archivo descargado.

Figura 5.

Instalación de YCSB

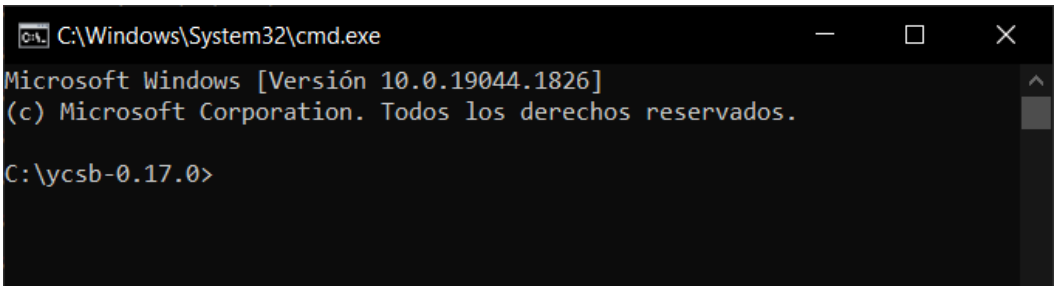


Nota. La figura muestra cómo se instala YCSB a través del comando.

Es importante colocar la carpeta en el disco C, dentro de ella ingresar a la ruta YCSB desde el símbolo del sistema CMD.

Figura 6.

Ingresar a la ruta YCSB desde el símbolo del sistema CMD



Nota. Creación propia.

Configuración de PostgreSQL

Dentro del directorio de YCSB, en la carpeta JDBC (C:\ycsb-0.17.0\jdbc-binding), creamos un archivo de propiedades donde se configura los parámetros presentados en la figura 6.

- **db.driver:** es el driver de PostgreSQL descrito en la sección Entorno de Trabajo PostgreSQL
- **db.url:** Define la conexión al servidor PostgreSQL, en este apartado reemplazar localhost por la dirección del servidor, se incluye el puerto y se establece el nombre de la base de datos creada, en este caso testa.
- **db.user:** Establece el nombre de usuario del cliente para conectarse al servidor.
- **db.passwd:** Define la contraseña del cliente para conectarse con el servidor.

Figura 7.

Parámetros de PostgreSQL

```
db.driver=org.postgresql.Driver
db.url=jdbc:postgresql://127.0.0.1:5432/testa
db.user=postgres
db.passwd=
```

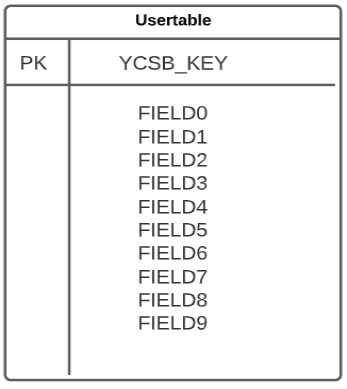
Nota. Creación propia.

Estructura de la base de datos

La base de datos que se utilizó consta de una tabla nombrada "*usertable*", que contiene la PK YCSB_KEY y nueve campos de tipo *varchar* como se muestra en la figura 7.

Figura 8.

Estructura de la Base de Datos PostgreSQL

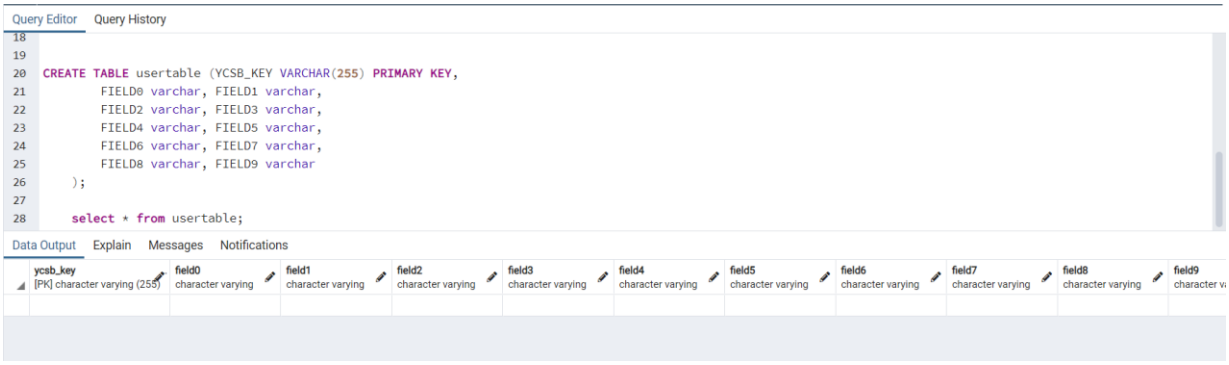


Nota. Creación propia.

Al generar la sentencia en PostgreSQL obtenemos los siguientes resultados presentes en la figura 8.

Figura 9.

Generación de tablas creadas en PostgreSQL



Nota. Creación propia.

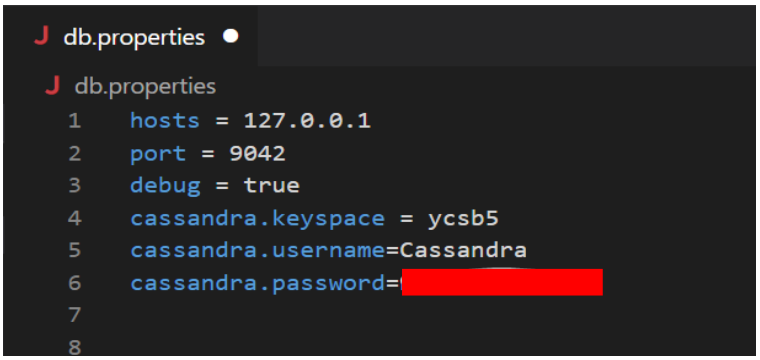
Configuración de Cassandra

Dentro del directorio C:\ycsb-0.17.0\cassandra-binding creamos un archivo de propiedades donde se configura los siguientes parámetros para generar las fases de carga y transaccional, como presenta la figura 9.

- **hosts:** Define la dirección con la que se conecta la base de datos.
- **port:** Define el puerto de conexión.
- **Debug**: permite encontrar errores
- **cassandra.keyspace**: Define el nombre de la base de datos o keyspace creada, para este caso ycsb5.
- **Cassandra.username:** Establece el nombre de usuario del cliente para conectarse al servidor.
- **Cassandra.password:** Define la contraseña del cliente para conectarse con el servidor.

Figura 10.

Parámetros de Cassandra



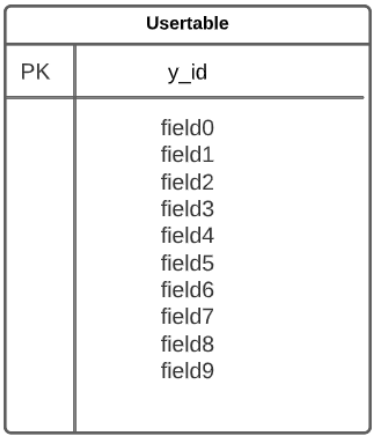
Nota. Creación propia

Estructura de Base de datos

En el entorno de Cassandra se debe mantener corriendo el demonio para el uso de mismo, para crear la base de datos utilizamos el programa DevCenter DataStax, mencionada en la sección Entorno de trabajo Cassandra. La estructura utilizada se presenta en la figura 10.

Figura 11.

Estructura de la Base de Datos Cassandra

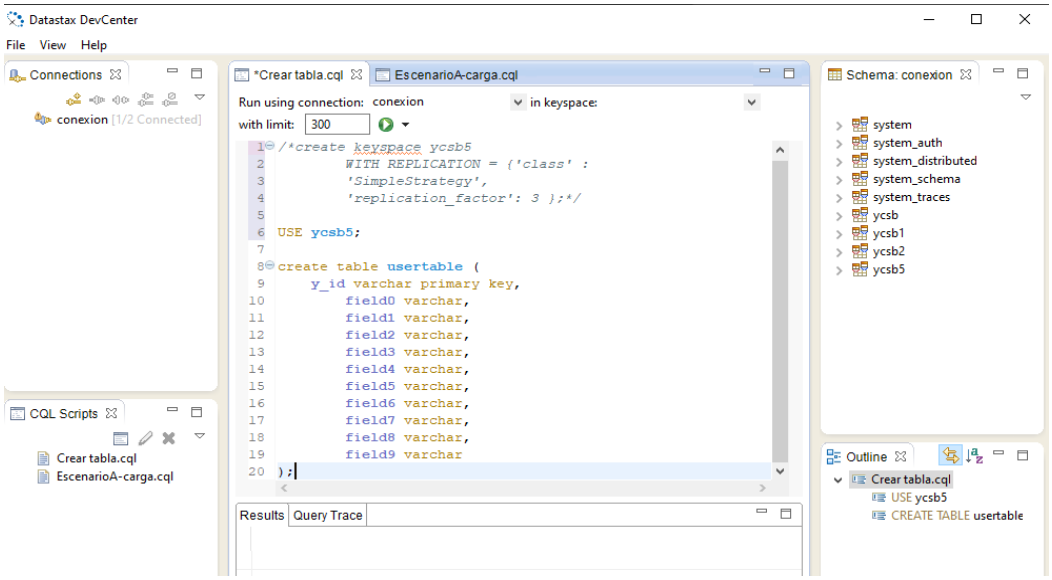


Nota. Creación propia

Dentro del Datastax se puede observar los keyspace creados, como muestra la figura 12.

Figura 12.

Datastax DevCenter de Cassandra



Nota. Creación propia

Capitulo IV: Plan de Pruebas y Resultados

Plan de Pruebas y Resultados

Una vez instalado los escenarios de trabajo y configurado las bases de datos tanto PostgreSQL como Casandra, se realizó un análisis comparativo con respecto a pruebas de dos fases consecutivas: carga y transacción. Los análisis de estas fases se basan en rendimiento y latencia.

1) Fase de carga, los resultados se basan en el tiempo de ejecución general.
2) Fase transaccional, los resultados se basan en la latencia y el rendimiento.

Fase de Carga

Se utilizan 2 bases de datos: PostgreSQL y Cassandra. Las pruebas de cargas están generadas por un 1 millón de registros de 1KB, 16 hilos de ejecución para obtener los tiempos generados. Los tiempos de ejecución total obtenidos en cada escenario fueron los siguientes.

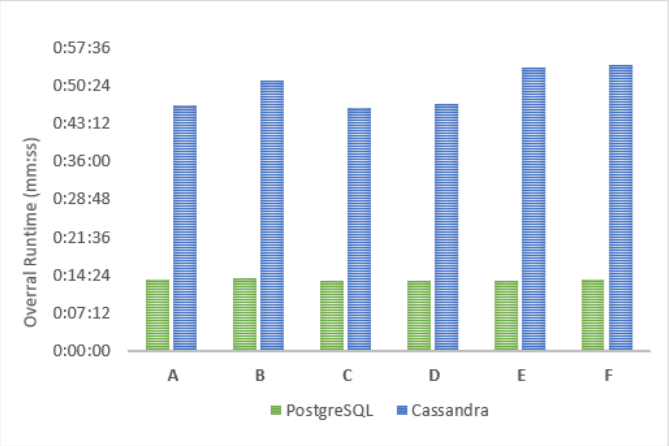
Fase de Carga Core i7

- El mejor tiempo de ejecución general para PostgreSQL tiene 13:10 para el escenario D; es decir, fue más rápido cuando solo había un 5 % de operaciones de lectura. Sin embargo, el mayor tiempo de ejecución general tiene 13:44 para el escenario B; es decir, más lento cuando solo había un 5 % de operaciones de actualización.
- El mejor tiempo de ejecución general para Cassandra tiene 46:04 para el escenario C; es decir, cuando solo había un 100 % de operaciones de lectura. Sin embargo, el mayor tiempo de ejecución general para Cassandra tiene 54:16 para el escenario D; es decir, más lento cuando había un 50 % de operaciones para lectura, escritura y modificación (RMW).

- En general, PostgreSQL se toma los mejores tiempos en comparación a Cassandra en todos los escenarios de trabajo, considerando que el tiempo más rápido en todas las cargas fue del escenario D con 13:10; es decir 3 veces más rápido que Cassandra. La figura 13, muestra el tiempo de ejecución general obtenido por cada carga de trabajo.

Figura 13.

Tiempo de ejecución genera Core i7



	Workloads					
PostgreSQL	0:13:26	0:13:44	0:13:16	0:13:10	0:13:13	0:13:29
Cassandra	0:46:40	0:51:18	0:46:04	0:46:54	0:53:36	0:54:16

Nota. Creación propia

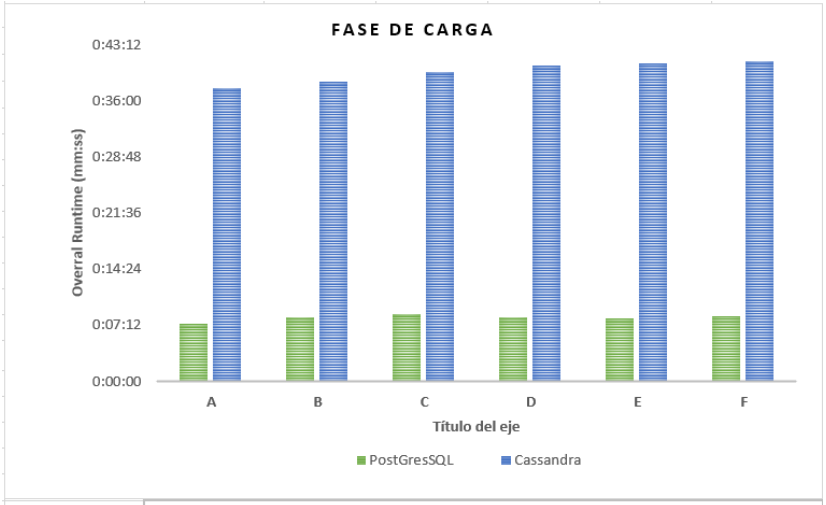
Fase de Carga - RIG de Procesamiento

- El mejor tiempo de ejecución general para PostgreSQL es de 7:20 (mm:ss) para el escenario A. Sin embargo, el mayor tiempo de ejecución general es 8:38 (mm:ss) para el escenario C.
- El mejor tiempo de ejecución general para Cassandra tiene 37:34 (mm:ss) para el escenario A. Sin embargo, el mayor tiempo de ejecución general para Cassandra tiene 41:05 (mm:ss) para el escenario F.

- En general, PostgreSQL se toma los mejores tiempos en comparación a Cassandra en todos los escenarios de trabajo.

Figura 14.

Fase de Carga RIG de procesamiento



	Workloads					
PostGresSQL	0:07:20	0:08:15	0:08:38	0:08:13	0:08:03	0:08:18
Cassandra	0:37:34	0:38:31	0:39:39	0:40:30	0:40:48	0:41:05

Nota. Creación Propia

Fase Transaccional

En el análisis de esta fase se utilizaron 2 métricas: la latencia definida en milisegundos (ms) y el rendimiento definido en operaciones por segundo (ops / seg). Para esta fase se considera un 1 millón de registros con 16 subprocesos (hilos) para cada base de datos.

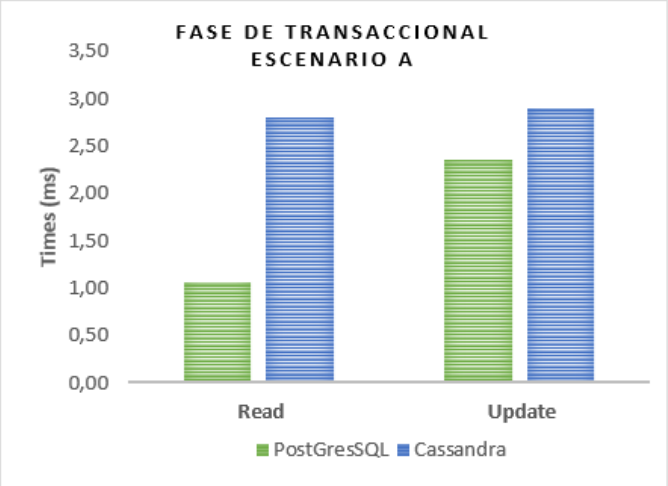
Latencia: se considera como el tiempo total que tarda en realizar cientos de operaciones en la base de datos hasta obtener una respuesta ante la solicitud. Las latencias obtenidas por cada escenario se muestran a continuación:

Fase Transaccional Core i7

Escenario A, para este escenario se utilizaron dos tipos de operaciones: 50 % de lectura y 50 % de actualización. La figura 15, muestra una comparación en tiempos de latencia para ambas bases de datos. PostgreSQL obtuvo 500.222 operaciones de lectura en 1.06 milisegundos y 499.778 operaciones de actualización en 2.35 milisegundos. Además, las latencias para PostgreSQL en ambas operaciones tuvieron una diferencia de 1.29 milisegundos; es decir, el comportamiento de rendimiento en ambas operaciones fue más rápido para las operaciones de lectura. En cambio, Cassandra tuvo tiempos de latencia similares en ambas operaciones, con 500.298 operaciones de lectura en 5.24 milisegundos y 499.702 operaciones de actualización en 5.07 milisegundos; es decir, el comportamiento de Cassandra en ambas operaciones de latencia fue de mayor tiempo para las operaciones de lectura.

Figura 15.

Latencia para Escenario A en Core i7



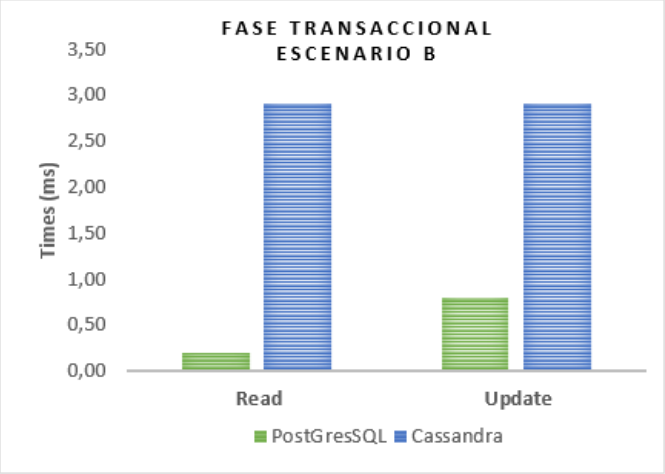
PostGresSQL	1,06	2,35
Cassandra	2,80	2,90

Nota. Creación propia.

Escenario B, para este escenario se utilizaron dos tipos de operaciones: 95 % de lectura y 5 % de actualización. La figura 16, muestra una comparación en tiempos de latencia para ambas bases de datos. PostgreSQL obtuvo 950.212 operaciones de lectura en 0.19 milisegundos y 49.787 operaciones de actualización en 0.79 milisegundos, las latencias en ambas operaciones tuvieron una diferencia de 0.60 milisegundos; es decir, el comportamiento de rendimiento en ambas operaciones fue más rápido para las operaciones de lectura. En cambio, Cassandra tuvo tiempos de latencia similares en ambas operaciones, con 949.599 operaciones de lectura en 5.8 milisegundos y 50.401 operaciones de actualización en 5.08 milisegundos; es decir, el comportamiento de Cassandra en ambas operaciones de latencia fue de mayor tiempo para las operaciones de lectura.

Figura 16.

Latencia para Escenario B en Core i7



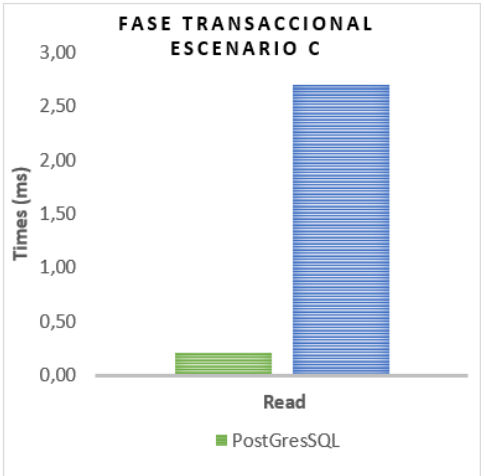
PostGresSQL	0,22	0,66
Cassandra	2,90	2,90

Nota. Creación propia.

Escenario C, para este escenario se utilizó un tipo de operación: 100 % de lectura. La figura 17, muestra una comparación en tiempos de latencia para ambas bases de datos. Al ser solo de lectura se realizan la misma cantidad de operaciones en ambas bases de datos. PostgreSQL tuvo 0.21 milisegundos, a comparación con Cassandra que tuvo una latencia de lectura de 4.06 milisegundos; es decir, PostgreSQL es más rápido en las operaciones de lectura con una diferencia de 3.85 milisegundos para Cassandra.

Figura 17.

Latencia para Escenario C en Core i7



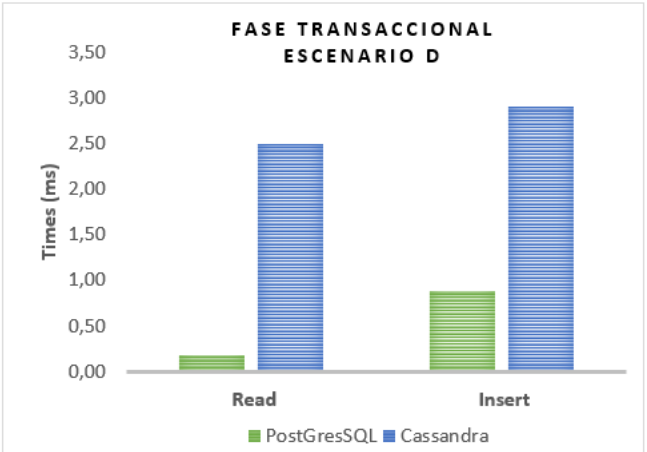
PostGresSQL	0,22
Cassandra	2,70

Nota. Creación propia.

Escenario D, para este escenario se utilizó dos tipos de operaciones: 95 % de lectura y 5 % de inserción. La figura 18, muestra una comparación en tiempos de latencia para ambas bases de datos. PostgreSQL obtuvo 950.291 operaciones de lectura en 0.18 milisegundos y 49.709 operaciones de inserción en 0.89 milisegundo, las latencias en ambas operaciones tuvieron una diferencia de 0.71 milisegundos; es decir, el comportamiento de rendimiento en ambas operaciones fue más rápido para las operaciones de lectura. En cambio, Cassandra tuvo tiempos de latencia similares en ambas operaciones, con 950.037 operaciones de lectura en 4.30 milisegundos y 49.963 operaciones de inserción en 4.26 milisegundos; es decir, el comportamiento de Cassandra en ambas operaciones de latencia fue de mayor tiempo para las operaciones de lectura.

Figura 18.

Latencia para Escenario D en Core i7



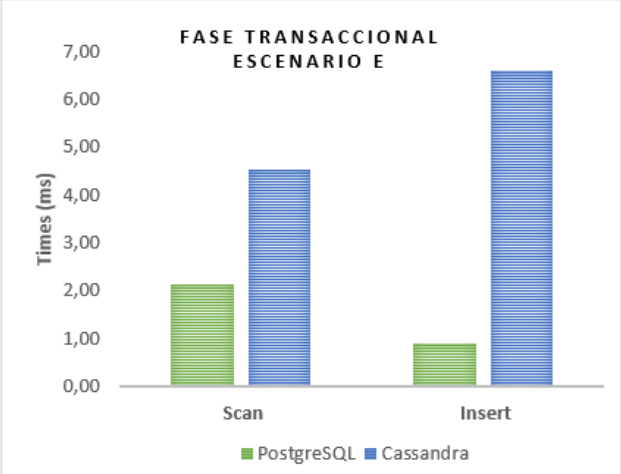
PostGresSQL	0,18	0,89
Cassandra	2,50	2,90

Nota. Creación propia

Escenario E, para este escenario se utilizó dos tipos de operaciones: 95 % de escaneo y 5 % de inserción. La figura 19, muestra una comparación en tiempos de latencia para ambas bases de datos. PostgreSQL obtuvo 949.898 operaciones de escaneo en 2.12 milisegundos y 50102 operaciones de inserción en 0.88 milisegundos, las latencias en ambas operaciones tuvieron una diferencia de 1.24 milisegundos; es decir, el comportamiento de rendimiento en ambas operaciones fue más rápido para las operaciones de inserción. En cambio, Cassandra obtuvo 949.955 operaciones de escaneo en 4.51 milisegundos y 50.045 operaciones de inserción en 6.57 milisegundos; es decir, Cassandra obtuvo un comportamiento de rendimiento diferente en ambas operaciones.

Figura 19.

Latencia para Escenario E en Core i7



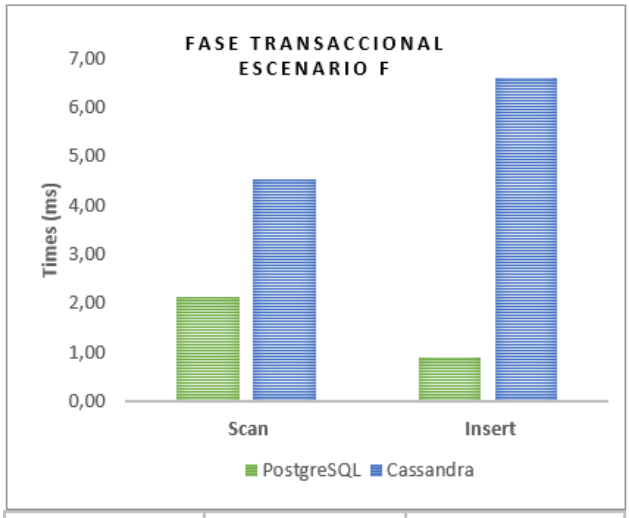
PostgreSQL	2,12	0,88
Cassandra	4,51	6,57

Nota. Creación propia

Escenario F, para este escenario se utilizó tres tipos de operaciones: lectura, modificación y escritura. La figura 20, muestra una comparación en tiempos de latencia para ambas bases de datos. PostgreSQL tuvo un tiempo de latencia de 2.77 milisegundos para lectura, modificación y escritura. En cambio, Cassandra tuvo un tiempo de latencia más elevado de 8.25 milisegundos.

Figura 20.

Latencia para Escenario F en Core i7



PostgreSQL	0,28	2,77
Cassandra	4,37	8,25

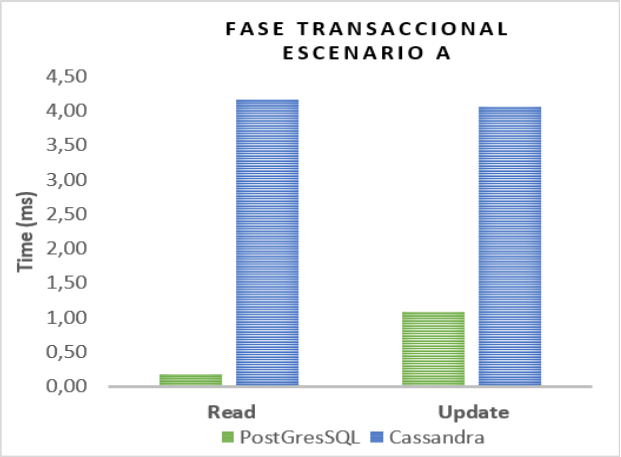
Nota. Creación propia

Fase Transaccional RIG de Procesamiento

Escenario A, PostgreSQL obtuvo PostgreSQL obtuvo 500.183 operaciones de lectura en 0.18 milisegundos y 499.817 operaciones de actualización en 1.08 milisegundos. Además, las latencias para PostgreSQL en ambas operaciones tuvieron una diferencia de 1.29 milisegundos. En cambio, Cassandra tuvo tiempos de latencia similares en ambas operaciones, con 500.443 operaciones de lectura en 1,08 milisegundos y 499.557 operaciones de actualización en 4,06 milisegundos.

Figura 21.

Latencia para Escenario A en RIG de procesamiento



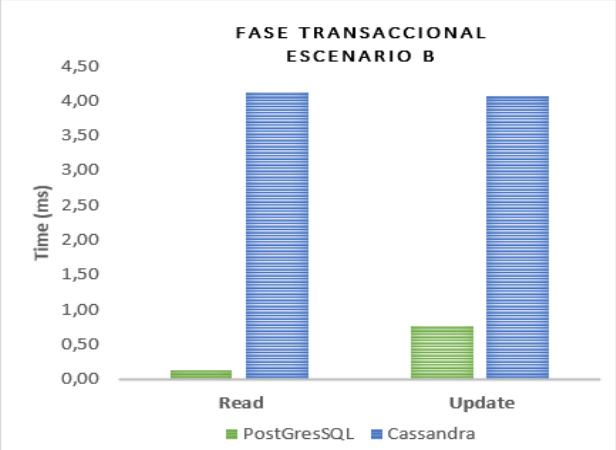
PostGresSQL	0,175	1,084
Cassandra	4,17	4,06

Nota. Creación propia

Escenario B, PostgreSQL obtuvo 950.262 operaciones de lectura en 0.13 milisegundos y 49.738 operaciones de actualización en 0.76 milisegundos, las latencias en ambas operaciones tuvieron una diferencia de 0.60 milisegundos. En cambio, Cassandra tuvo tiempos de latencia similares en ambas operaciones, con 950.061 operaciones de lectura en 4,12 milisegundos y 49.939 operaciones de actualización en 4.06 milisegundos.

Figura 22.

Latencia para Escenario B en RIG de procesamiento



PostGresSQL	0,125	0,761
Cassandra	4,12	4,06

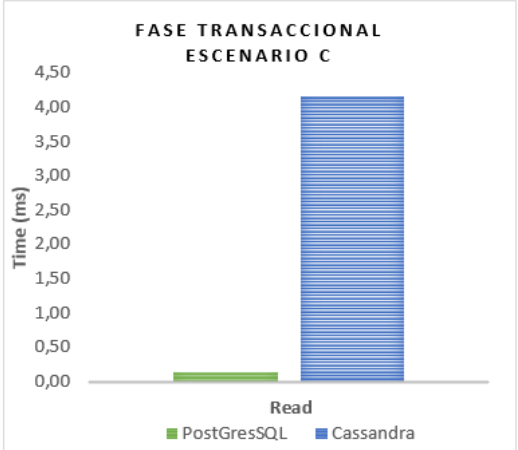
Nota. Creación propia

Escenario C, PostgreSQL tuvo 0.13 milisegundos, a comparación con Cassandra que tuvo una latencia de lectura de 4.16 milisegundos con un millón de datos, como muestra la figura 23.

Escenario D, PostgreSQL obtuvo 950.334 operaciones de lectura en 0.12 milisegundos y 49.666 operaciones de inserción en 1.50 milisegundo, las latencias en ambas operaciones tuvieron una diferencia de 0.71 milisegundos. En cambio, Cassandra tuvo tiempos de latencia similares en ambas operaciones, con 949.748 operaciones de lectura en 4.14 milisegundos y 49.963 operaciones de inserción en 4.15 milisegundos, como muestra la figura 24.

Figura 23.

Latencia para Escenario C en RIG de procesamiento

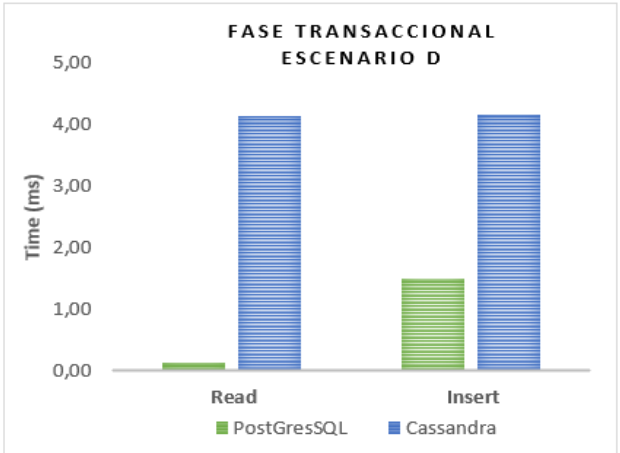


PostGresSQL	0,128
Cassandra	4,159

Nota. Creación propia

Figura 24.

Latencia para Escenario D en RIG de procesamiento



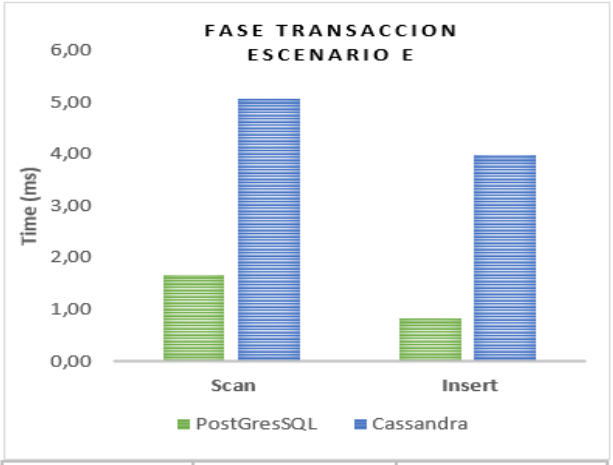
PostGresSQL	0,119	1,496
Cassandra	4,143	4,147

Nota. Creación propia

Escenario E, PostgreSQL obtuvo 952.165 operaciones de escaneo en 1.67 milisegundos y 950.165 operaciones de inserción en 0.82 milisegundos, las latencias en ambas operaciones tuvieron una diferencia de 1.24 milisegundos. En cambio, Cassandra obtuvo 950.097 operaciones de escaneo en 5.06 milisegundos y 49.903 operaciones de inserción en 3.99 milisegundos.

Figura 25.

Latencia para Escenario E en RIG de procesamiento



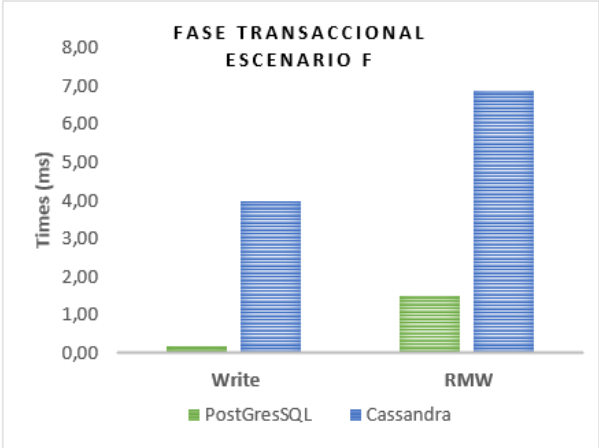
PostGresSQL	1,67	0,823
Cassandra	5,063	3,989

Nota. Creación propia

Escenario F, PostgreSQL tuvo un tiempo de latencia de 1,51 milisegundos para lectura, modificación y escritura. En cambio, Cassandra tuvo un tiempo de latencia más elevado con un valor de 6.86 milisegundos.

Figura 26.

Latencia para Escenario E en RIG de procesamiento



PostGresSQL	0,2	1,509
Cassandra	3,999	6,859

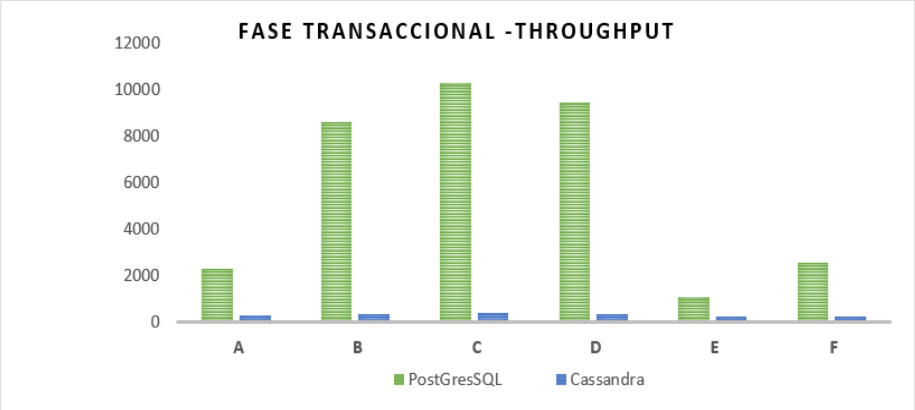
Nota. Creación propia

Rendimiento Core i7

Se considera como el número de operaciones que la base de datos realiza en cada carga de trabajo, estas operaciones están definidas en segundos (ops/seg). La figura 27, muestra el número de operaciones realizadas por cada carga de trabajo.

Figura 27.

Rendimiento por cada carga de trabajo en Core i7



	Workloads					
PostGresSQL	2308	8598	10292	9465	1110	2549
Cassandra	334	352	413	381	263	247

Nota. Creación propia

El rendimiento obtenido por cada carga de trabajo fue el siguiente:

- El mejor rendimiento para PostgreSQL con el mayor número de operaciones por segundo para el escenario C con 10292 ops/seg, no obstante, el menor número de operaciones realizadas fue el escenario D con 1110 ops/seg; es decir, PostgreSQL está ejecutando más operaciones.
- El mejor rendimiento para Cassandra fue la carga de trabajo del escenario D con 363 ops/seg, no obstante, el menor número de operaciones realizadas fue el escenario F con 247 ops/seg; es decir, Cassandra realiza menos operaciones en comparación con PostgreSQL.

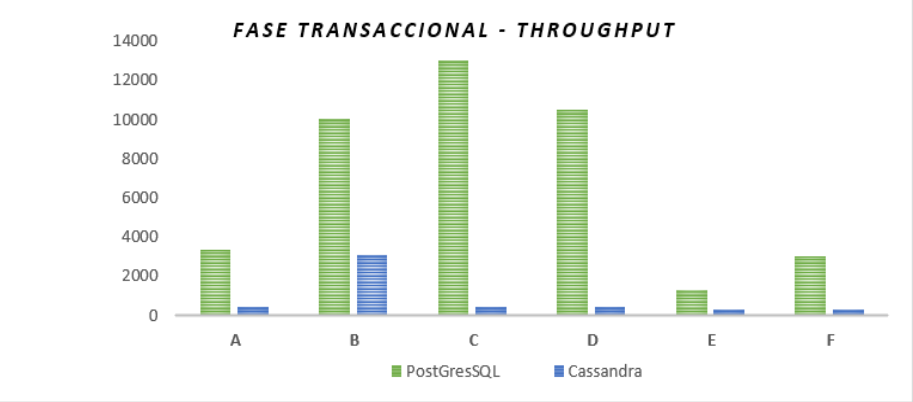
Rendimiento RIG de Procesamiento

La figura 28, muestra el número de operaciones realizadas por cada carga de trabajo, el rendimiento obtenido fue el siguiente:

- El mejor rendimiento para PostgreSQL con el mayor número de operaciones por segundo para el escenario C con 13027 ops/seg, no obstante, el menor número de operaciones realizadas fue el escenario E con 1282 ops/seg; es decir, PostgreSQL está ejecutando más operaciones.
- El mejor rendimiento para Cassandra fue la carga de trabajo del escenario B con 3073 ops/seg, no obstante, el menor número de operaciones realizadas fue el escenario F con 314 ops/seg; es decir, Cassandra realiza menos operaciones en comparación con PostgreSQL.

Figura 28.

Rendimiento por cada carga de trabajo en RIG de procesamiento



	Workloads					
PostGresSQL	3328	10038	13027	10496	1282	3001
Cassandra	442	3073	439	437	334	314

Nota. Creación propia

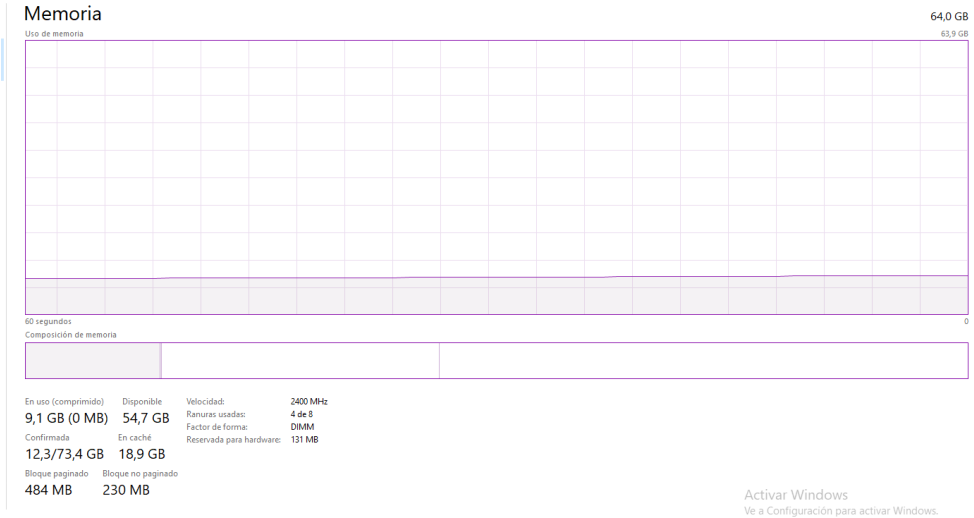
Análisis de Rendimiento

A continuación, se presenta los resultados obtenidos del administrador de tareas tanto del disco solido i7 como del RIG de procesamiento. Cada análisis de rendimiento fue tomado de la fase transaccional de cada escenario en un tiempo intermedio de ejecución.

- **Memoria - Escenario A:** Cassandra hace uso de memoria en un 9,1 GB, mientras PostgreSQL utiliza 5,6 GB de memoria al realizar un trabajo de 50 % lectura y 50 % actualización.

Figura 29.

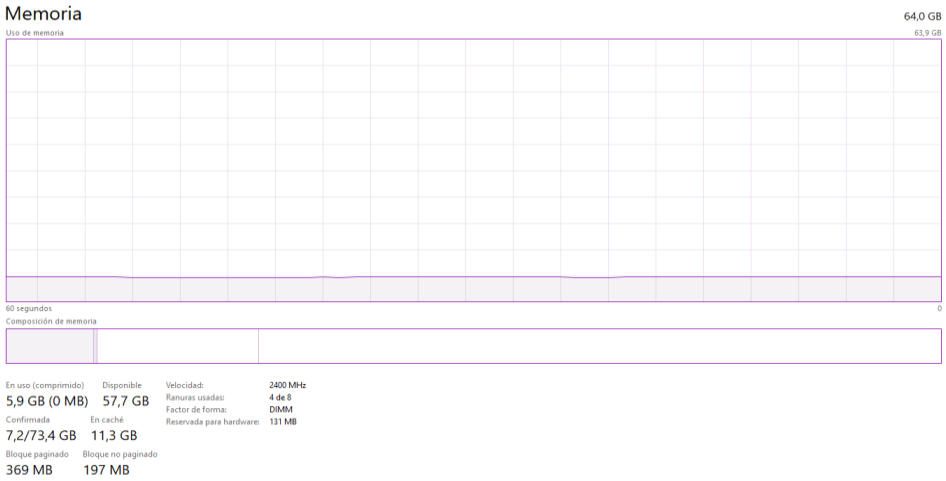
Uso de Memoria Cassandra



Nota: Creación propia

Figura 30.

Uso de Memoria PostgreSQL

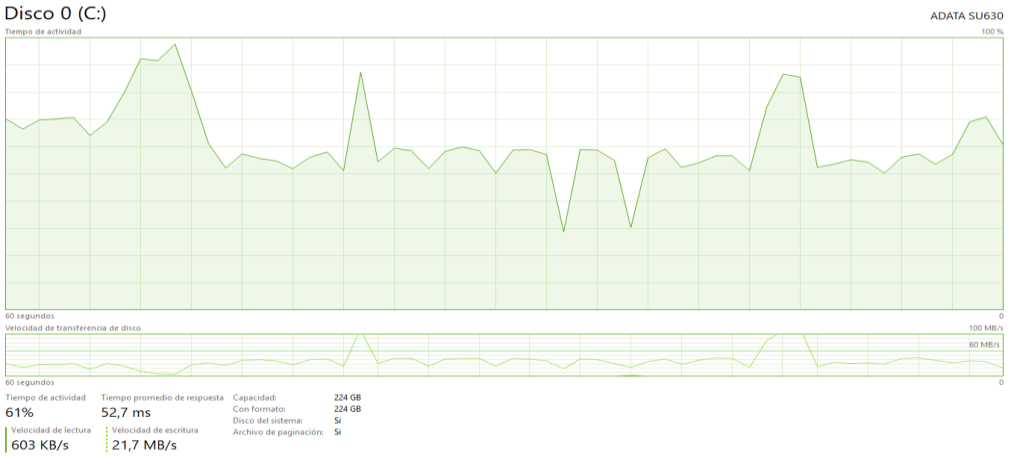


Nota: Creación propia

- **Disco – Escenario A**: PostgreSQL hace el consumo de un 100 MB/s en disco y 60 MB/s en velocidad de transferencia, mientras Cassandra consume 500 kB/s en disco y 450 kB/s en velocidad de transferencia.

Figura 31.

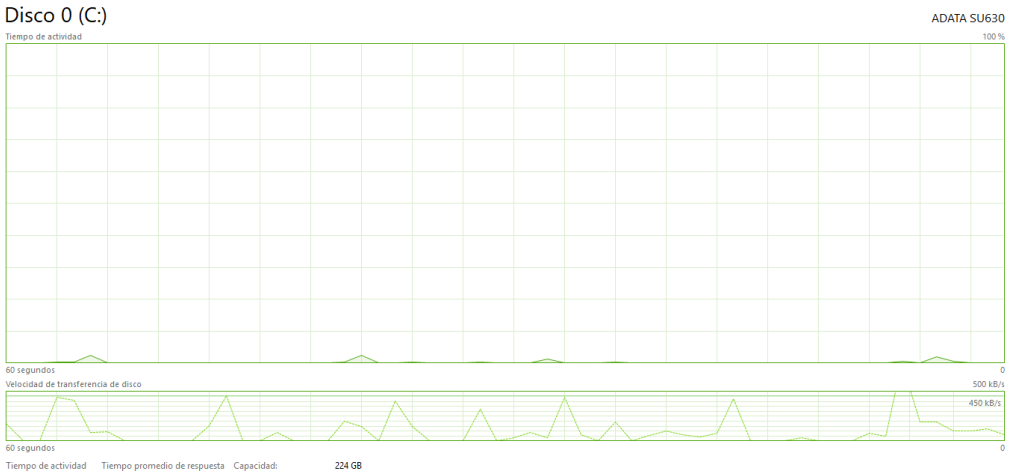
Consumo de Disco PostgreSQL



Nota: Creación Propia

Figura 32.

Consumo de Disco Cassandra

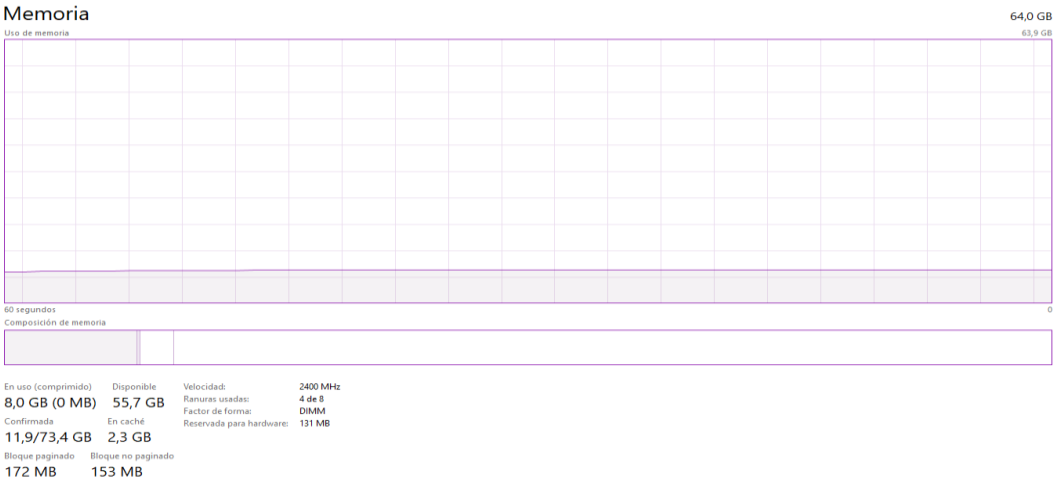


Nota: Creación Propia

- **Memoria - Escenario B:** Cassandra hace uso de memoria de 8,0 GB, mientras PostgreSQL hace uso de 6,0 GB al realizar un trabajo de 95 % de lectura y 5 % de actualización.

Figura 33.

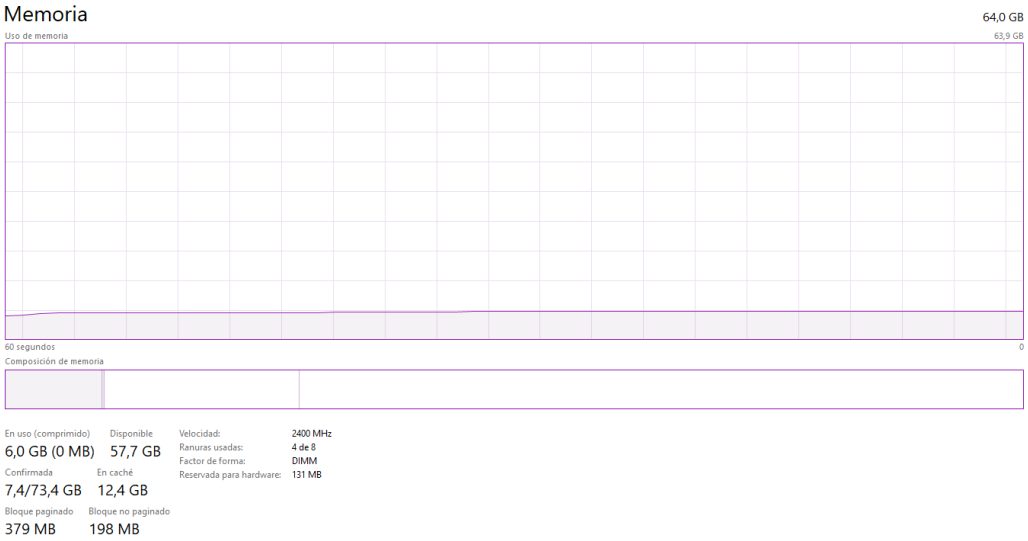
Uso de Memoria Cassandra



Nota: Creación Propia

Figura 34.

Uso de Memoria PostgreSQL

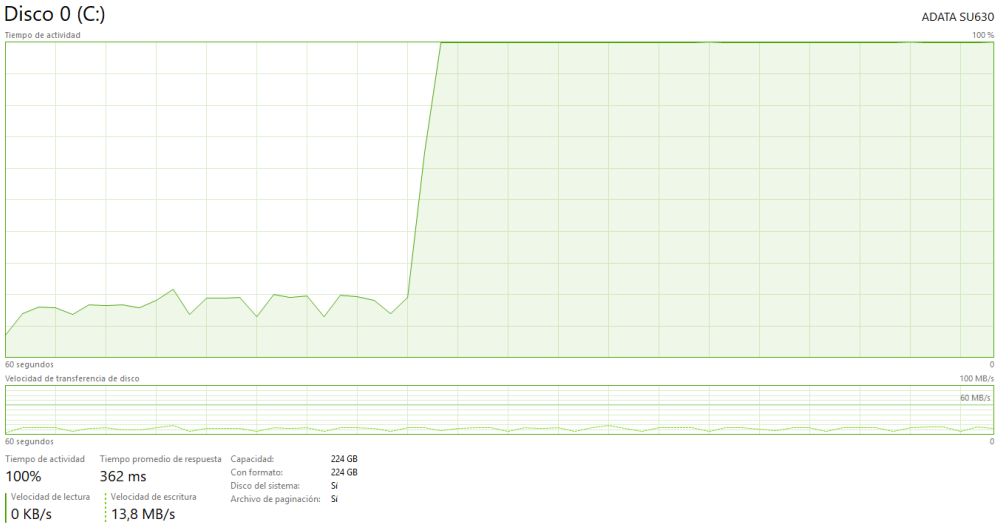


Nota: Creación propia

- **Disco - Escenario B:** PostgreSQL hace uso del 100% en disco y un 60 MB/s en velocidad de transferencia de disco como presenta en la figura 35, se eleva el consumo al iniciar la fase. Cassandra usa 10 MB/s de disco y un 7 MB/s en Velocidad de transferencia de disco.

Figura 35.

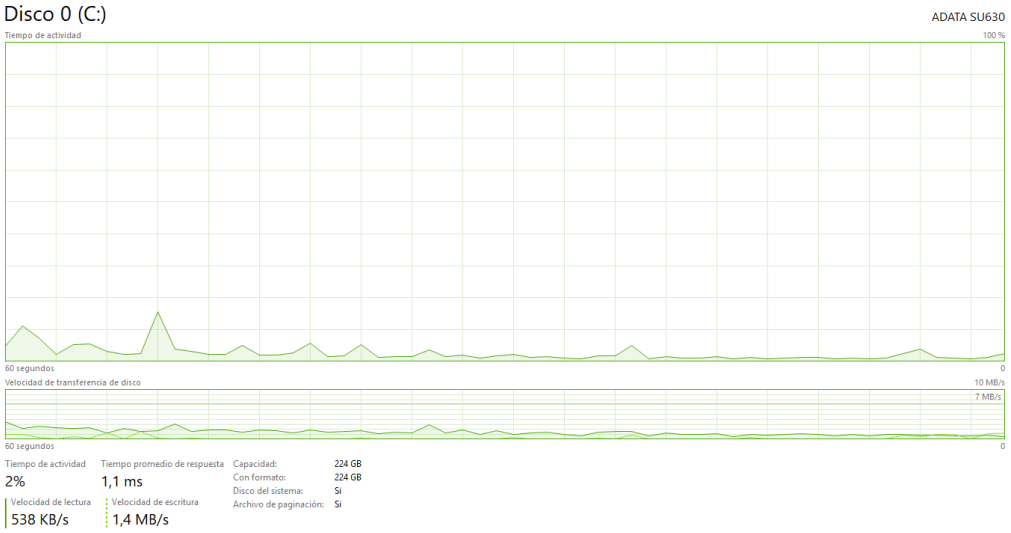
Consumo de Disco PostgreSQL



Nota: Creación propia

Figura 36.

Consumo de Disco Cassandra

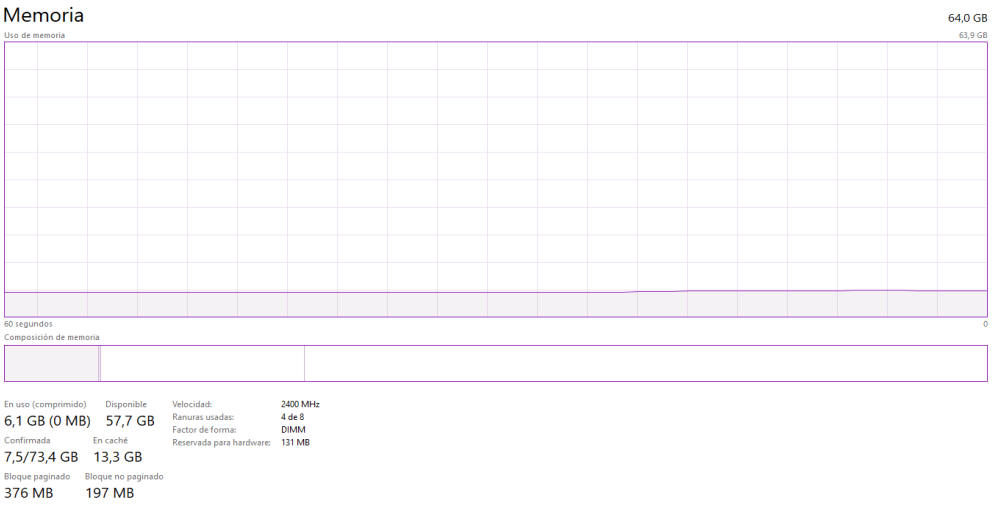


Nota: Creación propia

- **Memoria - Escenario C:** PostgreSQL utiliza 6,1 GB en memoria que cassandra hace uso de un 11,0 GB cuando se realiza un trabajo de 100% lectura.

Figura 37.

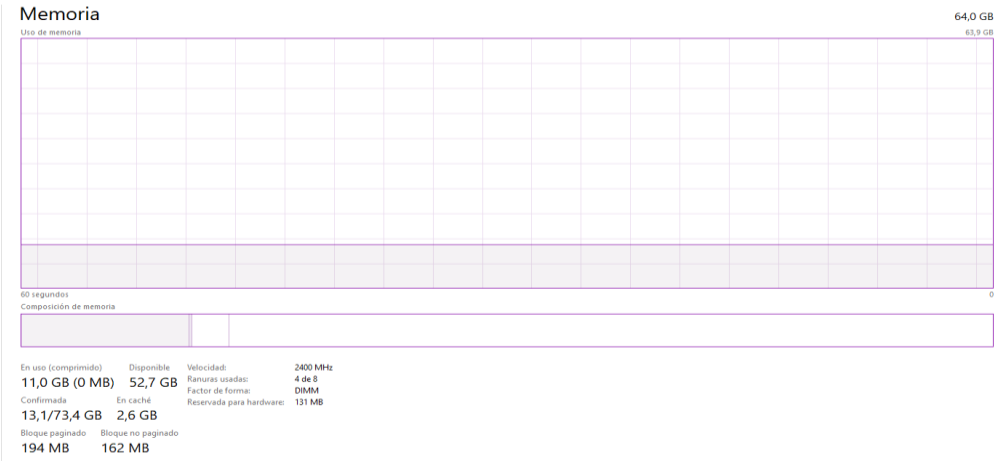
Uso de Memoria PostgreSQL



Nota: Creación propia

Figura 38.

Uso de Memoria Cassandra

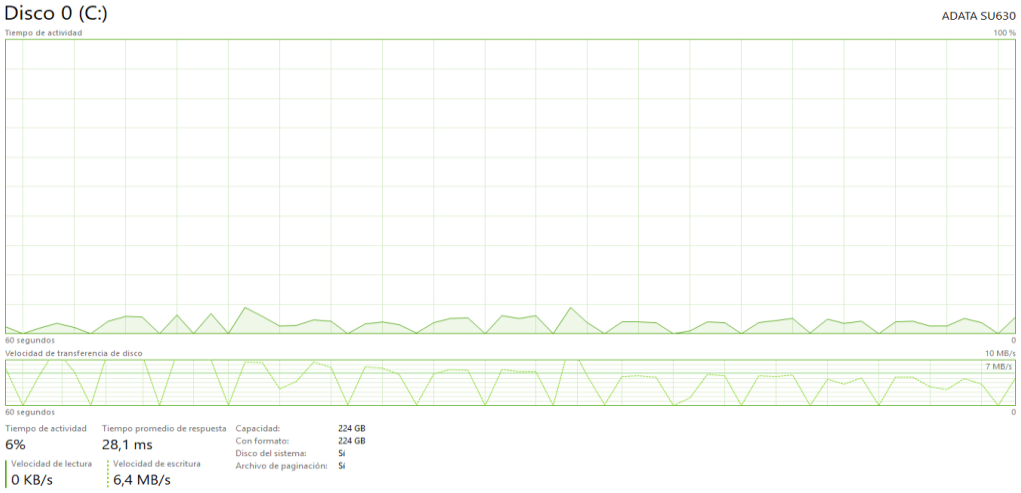


Nota: Creación propia

- **Disco – Escenario C:** PostgreSQL en este escenario usa 10 MB/s de disco con una velocidad de transferencia de 7MB/s por otra parte, cassandra baja el uso de disco a un 500 kB/s y en velocidad a un 450 kB/s.

Figura 39.

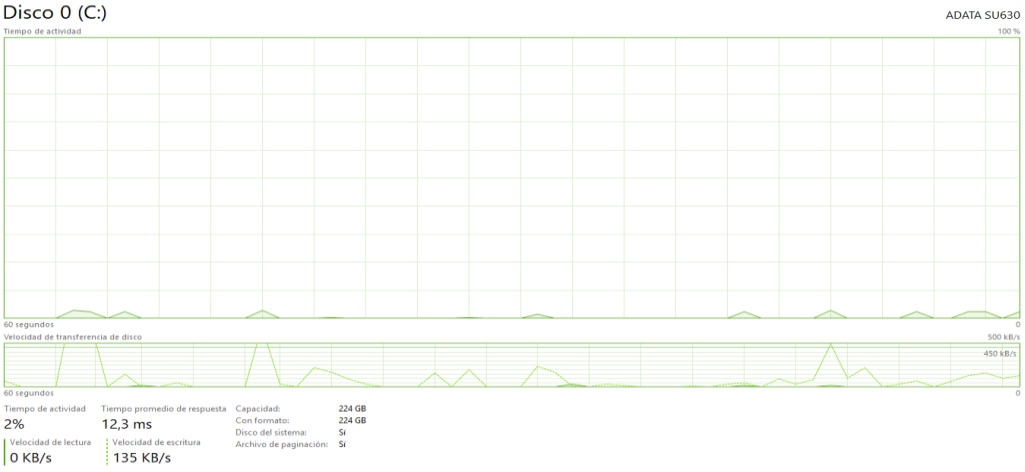
Consumo de Disco PostgreSQL



Nota: Creación Propia

Figura 40.

Consumo de Disco Cassandra

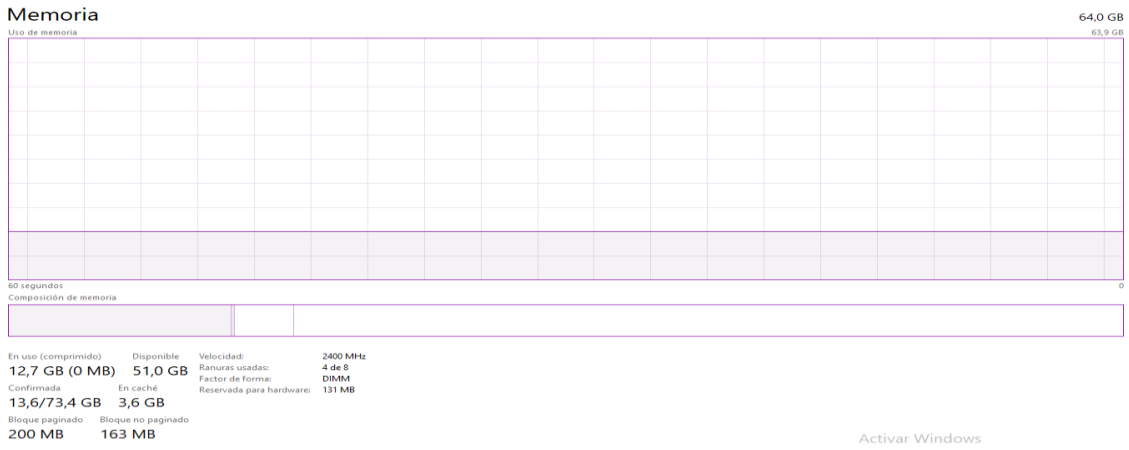


Nota: Creación propia

- **Memoria – Escenario D:** Cassandra hace uso de memoria de un 12,7 GB más a comparación de PostgreSQL que usa un 8,5 GB de memoria para una corrida de trabajo de 100% lectura.

Figura 41.

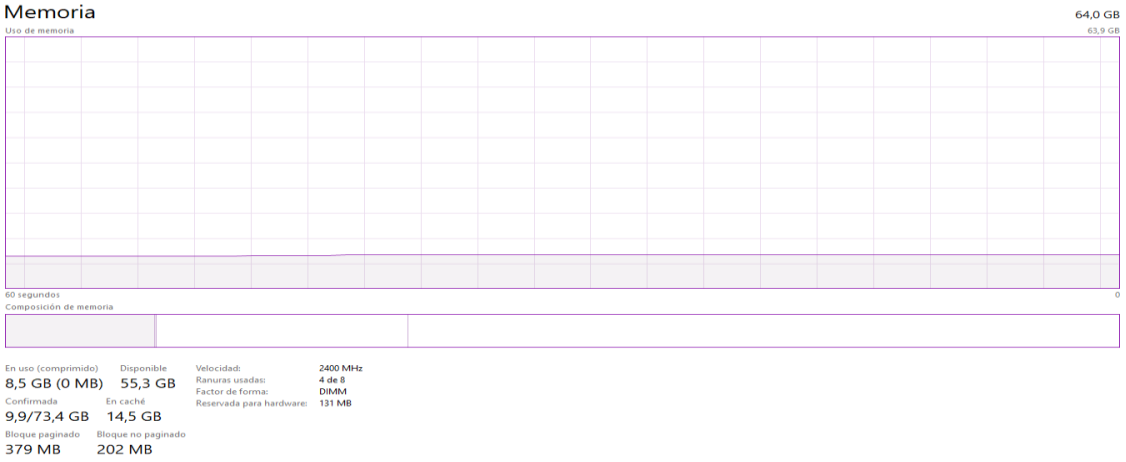
Uso de Memoria Cassandra



Nota: Creación propia

Figura 42.

Uso de Memoria Cassandra

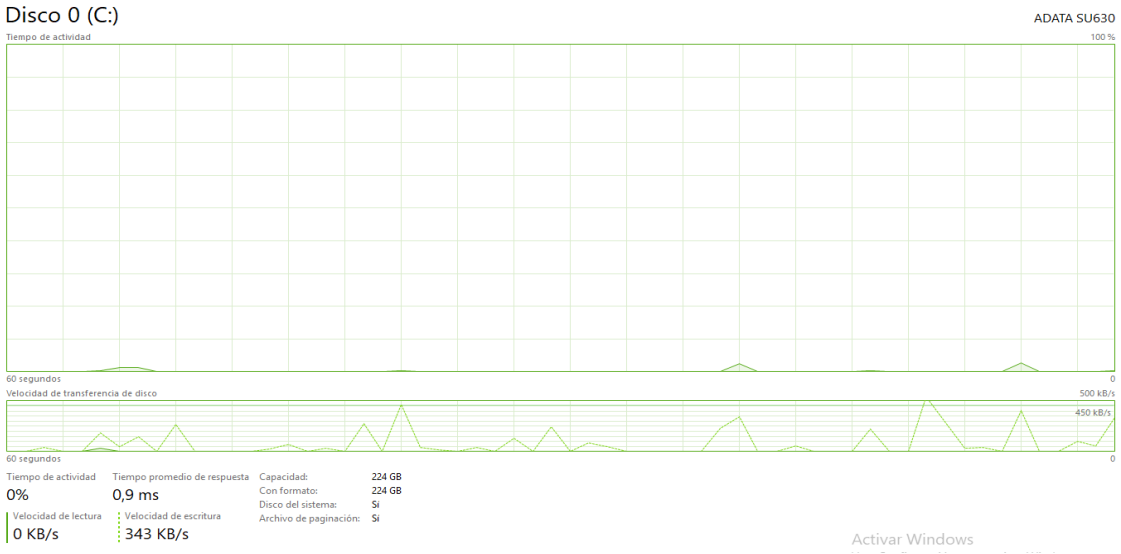


Nota: Creación propia

- **Disco – Escenario D:** En esta fase transaccional cassandra no hace consumo de del disco, se mantiene en 500 kB/s en consumo y 450 kB/s en velocidad de transferencia de disco, mientras en PostgreSQL se eleva en ciertos momentos, pero como tal hace uso de 10 MB/s en disco y 7 MB/s en velocidad de transferencia.

Figura 43.

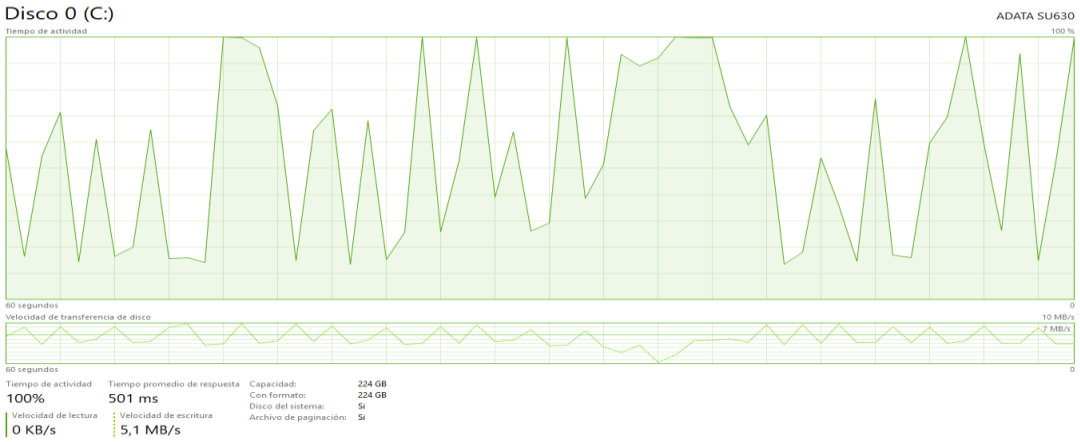
Consumo de Disco Cassandra



Nota: Creación propia

Figura 44.

Consumo de Disco PostgreSQL

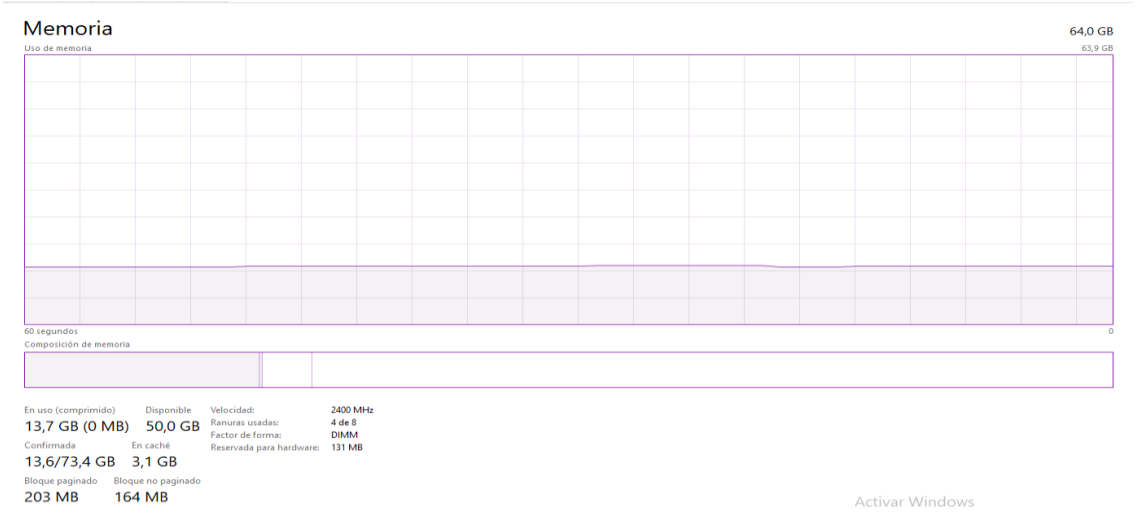


Nota: Creación propia

- **Memoria – Escenario E:** Cassandra hace uso de memoria en un 13,7 MB, es decir más que PostgreSQL quien utiliza 8,6 MB de memoria cuando realiza un trabajo de 95 % de lectura y 5 % de inserción.

Figura 45.

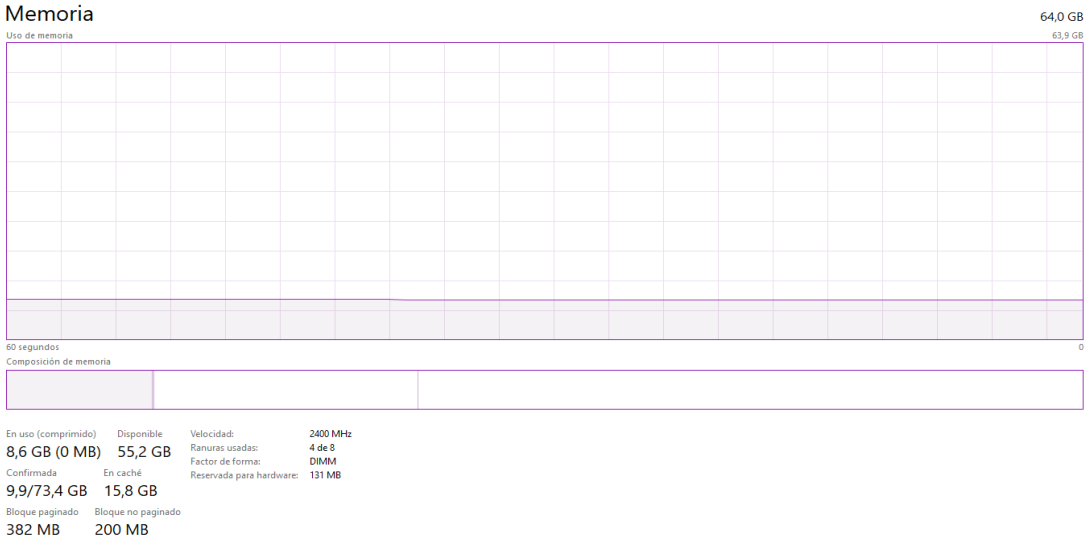
Uso de memoria Cassandra



Nota: Creación propia

Figura 46.

Uso de memoria PostgreSQL

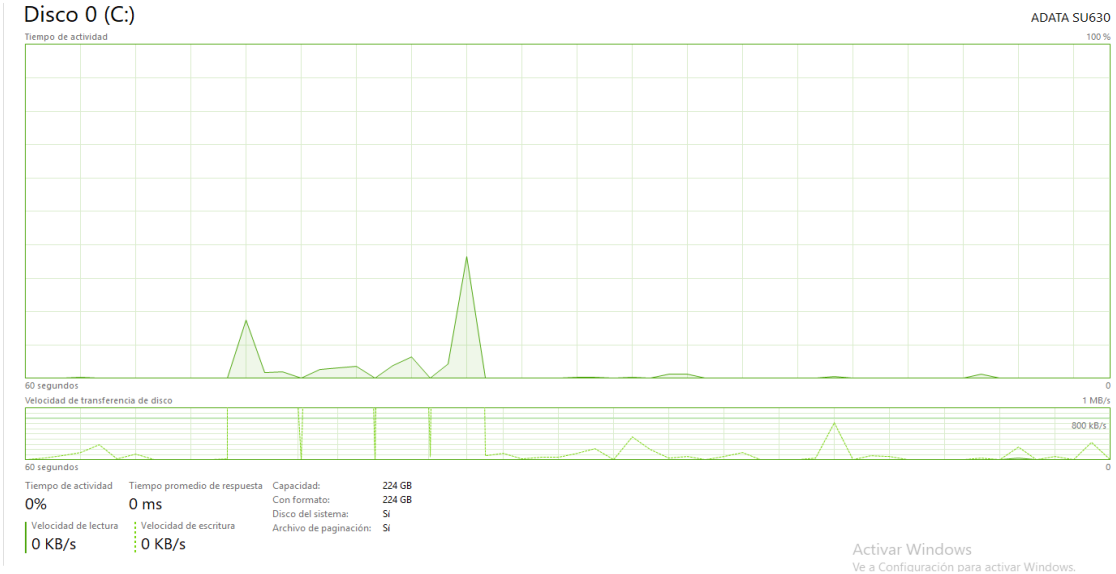


Nota: Creación propia

- **Disco – Escenario E:** PostgreSQL hace un consumo del disco de 10 MB/s y 7 MB/s en velocidad de transferencia de disco mientras cassandra utiliza 1 MB/s y 800 kB/s en velocidad de transferencia de disco.

Figura 47.

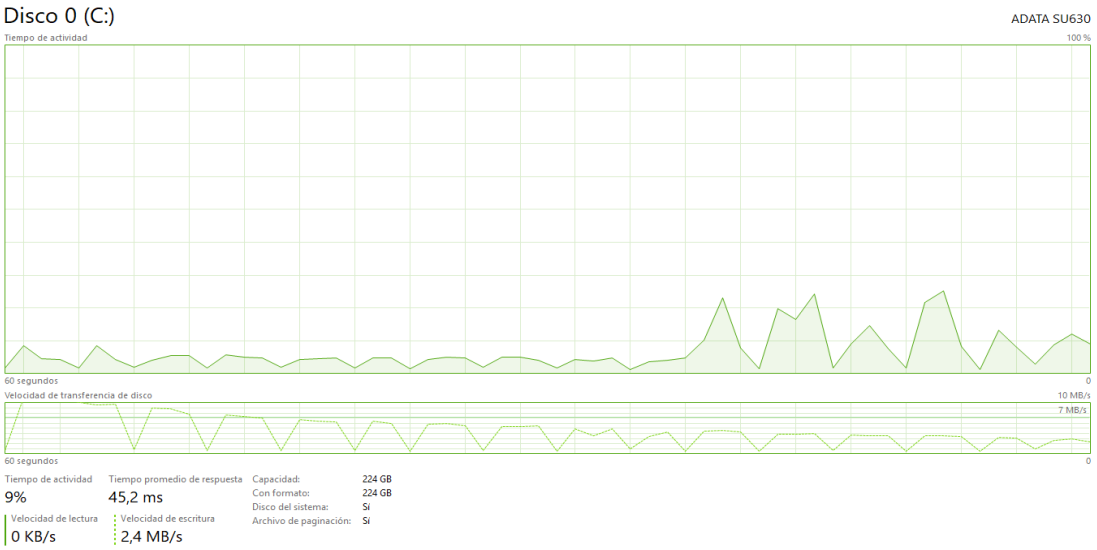
Consumo de Disco PostgreSQL



Nota: Creación propia

Figura 48.

Consumo de Disco Cassandra

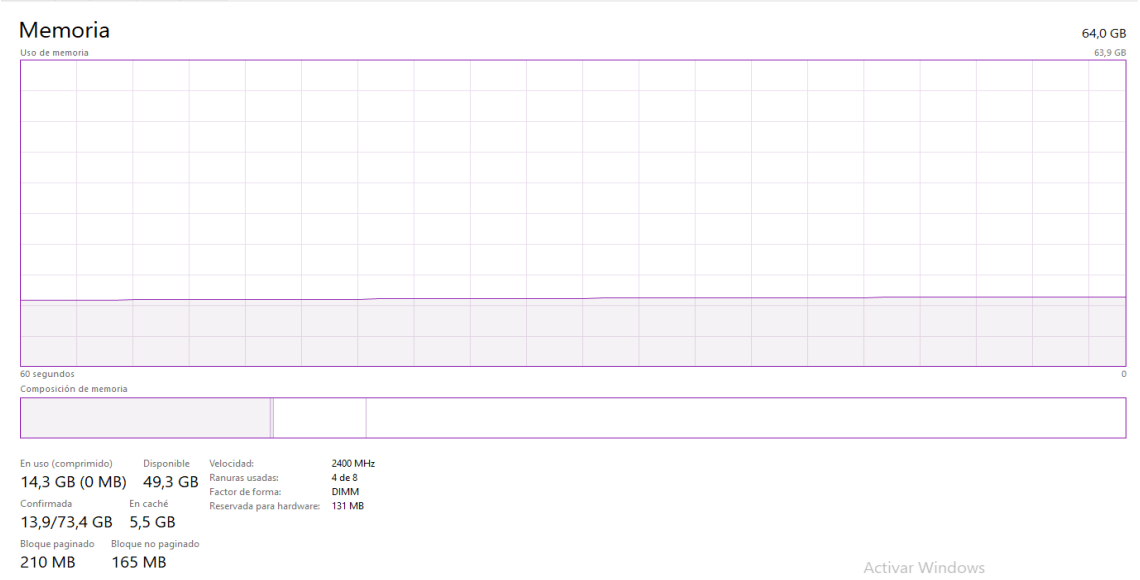


Nota: Creación propia

- **Memoria – Escenario F:** Cassandra utiliza 14,3 GB de memoria más a comparación de PostgreSQL que usa 6,2 GB en memoria cuando hace un trabajo de lectura, modificación y escritura.

Figura 49.

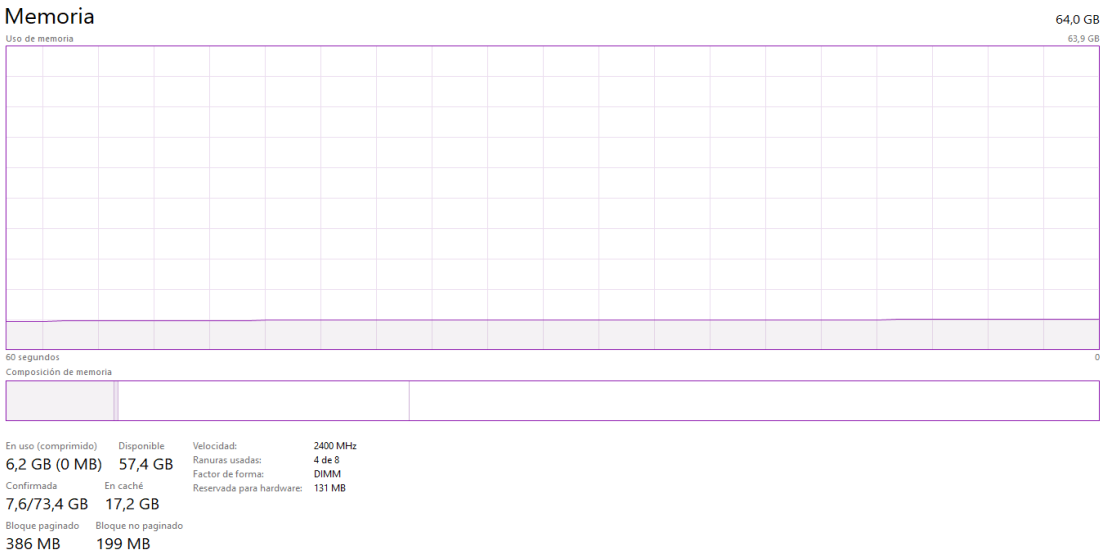
Uso de Memoria Cassandra



Nota: Creación propia

Figura 50.

Uso de Memoria PostgreSQL

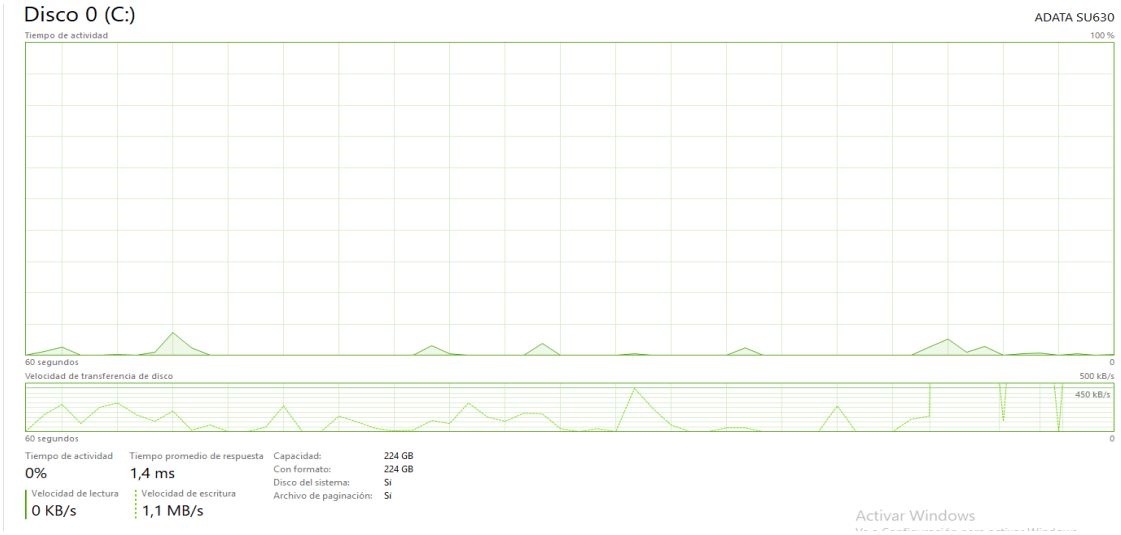


Nota: Creación propia

- **Disco – Escenario F:** PostgreSQL consume 100 MB/s en disco y 60 MB/s en velocidad de transferencia de disco, por otro lado, cassandra no hace casi uso del disco 500 kB/s en consumo y 450 kB/s en velocidad de transferencia.

Figura 51.

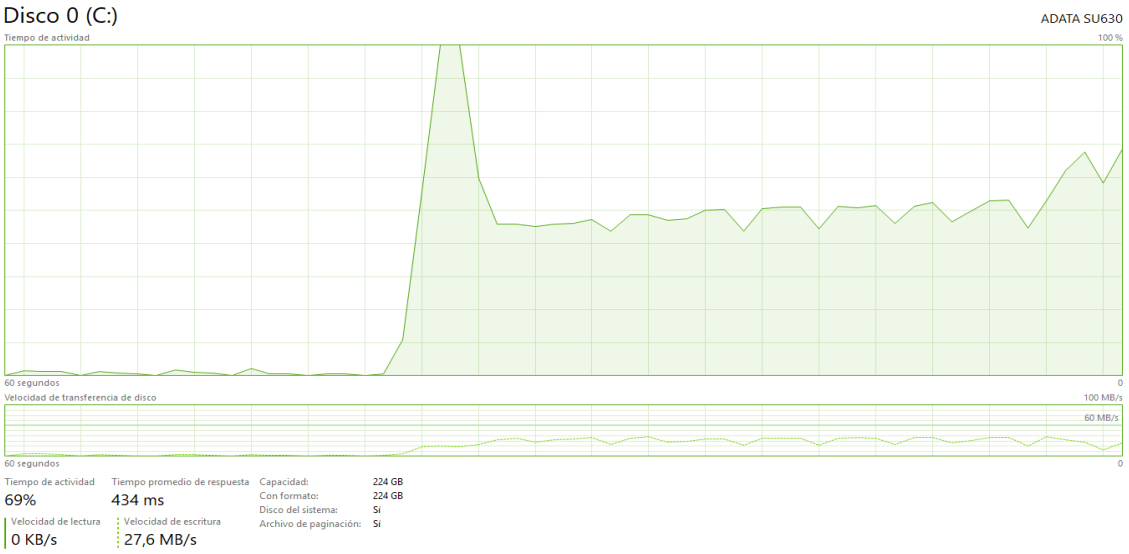
Consumo de Disco Cassandra



Nota: Creación propia

Figura 52.

Consumo de Disco PostgreSQL



Nota: Creación propia

Resultados

Tras realizadas las pruebas entre ambas bases de datos PostgreSQL y Cassandra se obtienen los siguientes resultados:

Ante los escenarios trabajados con el procesador core I7, se utilizó un millón de registros con 16 subprocesos (hilos), para el RIG de procesamiento se trabajó con 32 subprocesos los cuales fueron basados en una arquitectura de hardware y software bajo las mismas condiciones detalladas en el capítulo III, sección entorno de trabajo. Adicionalmente, se logra analizar el rendimiento de ambas bases de datos, demostrando que PostgreSQL fue el más rápido pese a ser una base de datos relacional. Cassandra demostró un rendimiento más lento en cuanto a las pruebas tanto en la fase de carga de datos, como en la fase transaccional.

Frente a la escalabilidad hacia el RIG de procesamiento se obtuvo que PostgreSQL tuvo tiempos de ejecución general menores a los obtenidos con el procesador Core I7, estimando una diferencia entre 6 – 7 minutos. Así mismo, Cassandra obtuvo tiempos menores, estimando una diferencia de 10 – 15 minutos.

Para esta investigación Cassandra al ser una base de datos de la familia de las columnas tiende a mostrar un tardío en la inserción de operaciones de lectura y escritura. Por otra parte, PostgreSQL destaca por realizar la inserción de datos en forma horizontal (filas).

Cassandra demostró ser más lento en comparación a PostgreSQL, debido a que, en esta investigación se trabajó de forma local; es decir, se utilizó un solo nodo para todo el proceso de trabajo. No obstante, si se toma a Cassandra en un entorno de trabajo virtualizado (Docker) con el uso de varios nodos (clúster), Cassandra demostrará el triple de rapidez a comparación con PostgreSQL.

Tabla 5.

Análisis comparativo PostgreSQL y Cassandra

		Procesador Core I7		
		ß	16 Subprocesos	à
Carga de trabajo	**Descripción**	**Cassandra**	**PostgreSQL**	**Desempeño Relativo**
A	50 % Lectura / 50 % Actualización	344.087	2308.583	6.70x
B	95 % Lectura / 5% Actualización	352.367	8598.156	24.40x
C	100 % Lectura	413.690	10292.936	24.88x
D	95 % Lectura / 5% Inserción	381.227	9465.125	24.82x
E	95 % Escaneo / 5% Inserción	263.291	1110.528	4.21x
F	50 % Lectura / 50% RMW	247.150	2549.271	10.31x
		RIG de Procesamiento		
		ß	32 Subprocesos	à
A	50 % Lectura / 50 % Actualización	442.181	3328.518	7.52x
B	95 % Lectura / 5% Actualización	3.073	10038.548	3.26x
C	100 % Lectura	439.075	13027.279	29.66x
D	95 % Lectura / 5% Inserción	437.187	10496.924	24.01x
E	95 % Escaneo / 5% Inserción	334.718	1282.848	3.83x
F	50 % Lectura / 0% RMW	314.676	3001.209	9.53x

Nota. La tabla representa la escalabilidad entre ambas bases de datos con diferentes procesadores.

Capítulo V: Conclusiones y Recomendaciones

Conclusiones

En base a la evaluación comparativa y pruebas de desarrollo se obtuvo que PostgreSQL obtuvo mayor rendimiento en el manejo de las operaciones CRUD frente a la herramienta YCSB, destacándose en mostrar mejores tiempos de latencia y alto consumo en servicios del sistema como en disco duro, CPU y memoria.

Mediante la revisión de la literatura e investigaciones similares sobre la comparativa entre bases de datos relaciones y no relaciones, se determinó que las bases de datos no relaciones son capaces de procesar grandes volúmenes de datos con relación a los métodos CRUD.

Con los seis escenarios de pruebas realizadas con el Framework YCSB se facilitó el manejo de las operaciones CRUD, generando datos precisos con el manejo de un millón de datos con 16 subprocesos.

Respecto a las mediciones de rendimiento al tiempo general ambas bases de datos se destacaron en mayor rapidez, estableciendo que PostgreSQL trabajo con una latencia de 13.10 (mm/ss) para el escenario D. Por otra parte, Cassandra trabajo con una latencia de 46.04 (mm/ss) para el escenario C.

Recomendaciones

Cassandra al ser una base de datos distribuida es capaz de procesar grandes cantidades de datos, se recomienda trabajar con un clúster mínimo de 5 nodos para obtener mejores tiempos al momento de realizar operaciones o transacciones.

Se recomienda realizar el mismo trabajo de investigación con distintas bases de datos relacionales y no relacionales para comparar los resultados futuros ante los resultados obtenidos en este proyecto.

Para obtener una mayor velocidad en cuanto a las operaciones de ambas bases de datos, se recomienda agregar índices, que permitirá un rápido acceso a los registros de las tablas.

Bibliografía

Alyasiri, B., Sahi, B., & AL-Khafaji, N. (2022). NoSQL: Will it be an alternative to a relational database? MySQL vs MongoDB comparison. *European Union Digital Library*, 15. https://doi.org/http://dx.doi.org/10.4108/eai.7-9-2021.2314925

Aniceto, R., Xavier, R., Guimarães, V., & Hondo, F. (2015). Evaluating the Cassandra NoSQL Database Approach for Genomic Data Persistency. *Hindawi Publishing Corporation*, 7. https://doi.org/http://dx.doi.org/10.1155/2015/502795

AWS. (n.d.). https://aws.amazon.com/es/relational-database/#:~:text=Una%20base%20de%20datos%20relacional%20es%20una%20recopilaci%C3%B3n%20de%20elementos,en%20la%20base%20de%20datos.

Ayudaley. (2020, Septiembre 07). https://ayudaleyprotecciondatos.es/bases-de-datos/relacional/#:~:text=Una%20base%20de%20datos%20relacional%20es%2C%20en%20esencia%2C%20un%20conjunto,los%20atributos%20de%20los%20datos.

Christian, A., Eirik, S., Ingvild Løver, T., Lars-Olav, V., & Lukas, T. (n.d.). Insertion speed of indexed spatial data: comparing MySQL, PostgreSQL and MongoDB.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking Cloud Serving Systems with YCSB. *Association for Computing Machinery*, 143. https://doi.org/https://doi.org/10.1145/1807128.1807152

IBM. (2019, Agosto 06). https://www.ibm.com/cloud/learn/relational-databases

Kozma, F., & Morschheuser, T. (2019). Cloud Service Environment PostgreSQL vs. Cassandra. *DiVA Digitala Vetenskapliga Arkivet*, 39. https://doi.org/diva2:1328885

Lima, A. T., Diógenes, V. F., & Pereira, J. Q. (2018). ESTUDO COMPARATIVO ENTRE POSTGRESQL (SGBD) E APACHE CASSANDRA (NOSQL). *ECOP*(5).

Lutkevich, B., & Biscobing, J. (2021, Junio 24). https://www.techtarget.com/searchdatamanagement/definition/relational-database

Madisyn. (2012). *Apache Cassandra*. https://cassandra.apache.org/_/cassandra-basics.html

Manuela, N. J., Senuri, S., Chathurika, S., Yasanthy, K., & Indraka, U. (2022). Cognitive Visual-learning Environment for PostgreSQL. *Cornell University*, 10. https://doi.org/https://doi.org/arXiv:2205.04834

Mesa, A. R. (2019, Junio 17). https://openwebinars.net/blog/que-es-apache-cassandra/

Nayantara, J. M., Sucharitharathna, S., Senarath, C., Kanagaraj, Y., & Udayakumara, I. (2022). Cognitive Visual-learning Environment for PostgreSQL. *Cornell University*. https://doi.org/https://doi.org/10.48550/arXiv:2205.04834

Nishtha, J., Sahil, P., Mehak, A., Ishita, K., & Dishant, G. (2019). A Survey and Comparison of Relational and Non-Relational Database. *International Journal of Engineering Research & Technology, 01*, 05. https://doi.org/10.1.1.678.9352

Panchenko, I. (2021, Mayo 27). https://www.infoworld.com/article/3619531/PostgreSQL-benefits-and-challenges-a-snapshot.html

Peterson, R. (2020, Enero 06). *Guru99*. https://www.guru99.com/introduction-PostgreSQL.html

Salazar, J. E. (2014). *Análisis comparativo de dos bases de datos SQL y dos bases de datos no SQL.* Retrieved May 12, 2022, from Universidad Tecnológica de Pereira: https://repositorio.utp.edu.co/handle/11059/5119

Seco, R. R. (2017, January 11). ANÁLISE COMPARATIVA ENTRE O BANCO DE DADOS CASSANDRA (MODELO NOSQL) E O POSTGRESQL (MODELO RELACIONAL) EM DUAS DIFERENTES ORGANIZAÇÕES EMPRESARIAIS. *Colloquium Exactarum, 8*(2), 39–56. http://revistas.unoeste.br/index.php/ce/article/view/1324

Tatsis, K. (2022). *A quantitative study on the popularity and performance of SQL and NoSQL DBMS.* Retrieved 16 de May de 2022, from DiVA: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1640259

Vaish, G. (2013). Getting Started with NoSQL: Your guide to the world and technology of NoSQL. In G. Vaish. Packt Publishing.

Veronica, A., & Bernardino, J. (2013). NoSQL Databases: MongoDB vs Cassandra. *ACM Digital Library*, 14–22. https://doi.org/https://doi.org/10.1145/2494444.2494447

Yarabarla, S. (2015). Learning Apache Cassandra. In S. Yarabarla, *Learning Apache Cassandra* (p. 350). Packt Publishing,.

Yarabarla, S. (2017). *Learning Apache Cassandra* (Second Edition ed.). Packt Publishing.