Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Evaluación experimental de Neo4J para su aplicación en el dominio sanitario

Autor: Antonio Franco Romero

Tutor: Jorge Calvillo Arbizu

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Evaluación experimental de Neo4J para su aplicación en el dominio sanitario

Autor:
Antonio Franco Romero

Tutor:
Jorge Calvillo Arbizu
Profesor Permanente Laboral

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2025

Trabajo Fin de Grado: Evaluación experimental de Neo4J para su aplicación en el dominio sanitario

Autor: Antonio Franco Romero

Tutor: Jorge Calvillo Arbizu

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quiero expresar mi agradecimiento a mis padres por haberme brindado la oportunidad de estudiar con la tranquilidad de saber que cuento con su respaldo y a mi pareja que ha supuesto un apoyo siempre que lo he necesitado

Me gustaría mostrar mi más sincero agradecimiento a mi tutor el profesor Jorge Calvillo Arbizu, así como al resto de profesores del Grado de Ingeniería de las Tecnologías de Telecomunicación por compartir sus conocimientos y estar ahí siempre que lo he necesitado.

Antonio Franco Romero

Sevilla, 2025

Resumen

Este Trabajo de Fin de Grado tiene como objetivo principal analizar y evaluar el rendimiento de la base de datos basada en grafos Neo4J aplicada al sector sanitario, específicamente en entornos de monitorización de pacientes a través de dispositivos autónomos bajo el paradigma de *Internet of Medical Things* (IoMT).

Para el desarrollo del proyecto, se ha diseñado una solución empleando drivers de Java integrados con Neo4J, lo que permitió la conexión y manipulación de la base de datos de manera eficiente. Además, se utilizó la librería OSHI para llevar a cabo la monitorización del sistema, registrando el consumo de recursos (CPU y RAM) durante la ejecución de las pruebas.

El trabajo experimental incluyó escenarios con distintos volúmenes de datos y frecuencias de transmisión, simulando condiciones reales en entornos hospitalarios, así como pruebas de distintas consultas de diferente complejidad. Las pruebas realizadas permitieron analizar el rendimiento del sistema bajo diversas cargas de trabajo, identificando las capacidades y limitaciones de Neo4J en términos de escalabilidad y consumo de recursos.

Los resultados obtenidos reflejan que Neo4J es una herramienta altamente eficiente para gestionar bases de datos con gran densidad de relaciones, destacando en entornos con cargas moderadas debido a su capacidad de manejar datos interconectados y realizar consultas complejas de forma rápida y precisa. A pesar de estos resultados positivos, el sistema mostró ciertas limitaciones al enfrentarse a cargas elevadas, donde se evidenció un incremento progresivo en el consumo de CPU y RAM, lo que afectó la estabilidad general.

Como parte de las conclusiones, se proponen diversas líneas de trabajo futuro que buscan optimizar el rendimiento y ampliar la aplicabilidad del sistema. Entre estas líneas destacan la optimización del rendimiento a través de técnicas avanzadas de indexación, la implementación de un entorno distribuido y la integración con sistemas reales para validar el desempeño en condiciones operativas.

Este trabajo establece una base sólida para futuras investigaciones que busquen mejorar el almacenamiento y análisis de datos interconectados en el ámbito sanitario, contribuyendo al desarrollo de soluciones más eficientes y escalables en el sector de la salud.

Abstract

This Bachelor's Thesis aims to analyze and evaluate the performance of the Neo4J graph database applied to the healthcare sector, specifically in patient monitoring environments using autonomous devices under the Internet of Medical Things (IoMT) paradigm.

For the development of the project, a solution was designed using Java drivers integrated with Neo4J, enabling efficient connection and manipulation of the database. Additionally, the OSHI library was used for system monitoring, recording resource consumption (CPU and RAM) during test execution.

The experimental work included scenarios with different data volumes and transmission frequencies, simulating real conditions in hospital environments, as well as tests involving queries of varying complexity. The tests allowed for an analysis of system performance under diverse workloads, identifying the capabilities and limitations of Neo4J in terms of scalability and resource consumption.

The results obtained show that Neo4J is a highly efficient tool for managing databases with a high density of relationships, excelling in environments with moderate loads due to its ability to handle interconnected data and perform complex queries quickly and accurately. Despite these positive results, the system exhibited certain limitations when faced with high loads, showing a progressive increase in CPU and RAM consumption, which affected overall stability.

As part of the conclusions, various future research lines are proposed to optimize performance and expand the system's applicability. These include performance optimization through advanced indexing techniques, the implementation of a distributed environment, and integration with real systems to validate performance under operational conditions.

This work establishes a solid foundation for future research aimed at improving the storage and analysis of interconnected data in the healthcare sector, contributing to the development of more efficient and scalable solutions in the health industry.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Motivación

En el mundo actual, gracias al desarrollo de internet y los avances en las tecnologías, la velocidad de transmisión y el volumen de los datos transmitidos no solo se han incrementado exponencialmente, sino que siguen aumentando día a día. Ese alto volumen de datos transmitidos a alta velocidad debe ser recepcionado y almacenado para su posterior explotación. Es por eso que es importante desarrollar soluciones adecuadas para el almacenamiento de datos, teniendo en cuenta su conectividad, disponibilidad, seguridad y tiempos de respuesta.

En el ámbito sanitario, como en otros sectores, el número de dispositivos que operan de manera autónoma (por ejemplo, realizando mediciones en pacientes) es cada día mayor. Para estos dispositivos, merece la pena introducir el término IoMT (*Internet of Medical Things*), que refiere a la red de dispositivos físicos implicados en la asistencia de los pacientes, por ejemplo, siendo capaces de captar y transmitir información sobre los pacientes o su entorno en tiempo real. Dicha red de dispositivos permite realizar tratamientos telemáticamente y la monitorización de pacientes en tiempo real, entre otras capacidades [1].

Todos estos avances suponen numerosas fuentes de datos mandando información simultáneamente y en tiempo real a un servidor. El volumen de datos puede ser muy alto y, si este servidor no es capaz de soportarlo, podría saturarse y perder información, lo que supondría un cuello de botella para soluciones por ejemplo de monitorización de pacientes en tiempo real.

Es por ello que las características de las soluciones que recepcionen y almacenen datos de monitorización de pacientes en tiempo real son críticas para garantizar la escalabilidad de los sistemas y la conservación de los datos de salud de los pacientes. Por otro lado, el almacenamiento de los datos también es crucial para la posterior explotación de los datos y la generación de conocimiento a partir de ellos (descubrimiento de tendencias de riesgo, alarmas, etc.).

Las bases de datos relacionales se llevan usando en la industria desde los inicios de la revolución computacional. Sin embargo, poseen ciertas limitaciones debido a su rigidez estructural para ciertos ámbitos de aplicación. Las bases de datos basadas en grafos solucionan este problema, dando igual importancia a la estructura y las relaciones entre datos, que a los datos en sí. Esto supone solucionar los problemas de escalabilidad que tienen las bases de datos relacionales [2]. Entre las implementaciones actuales de bases de datos, Neo4J es una de las más reconocidas y utilizadas. Su capacidad para manejar grandes volúmenes de datos y realizar consultas complejas de manera eficiente la hace la elegida para aplicaciones con un gran volumen de datos y de relaciones. Su flexibilidad permite añadir relaciones nuevas sin mayor complejidad.

La motivación de este proyecto es, por tanto, analizar las prestaciones de una base de datos basada en grafos (concretamente, Neo4J) para el almacenamiento de datos de monitorización de pacientes.

1.2 Objetivos del proyecto

El objetivo principal del proyecto es el análisis de las prestaciones de Neo4J en diferentes escenarios de prueba modificando tanto el número de fuentes que transmiten datos como la frecuencia de envío, así como los tiempos

de respuesta del sistema cuando varía la cantidad de datos almacenados.

Este objetivo principal se concreta en los siguientes subobjetivos:

1. En primer lugar, estudio de Neo4j y su lenguaje de consulta (Cypher), analizando sus características, capacidades y limitaciones.
2. Posteriormente se realizará el diseño de la base de datos, indicando los tipos de nodos y las relaciones entre ellos. A partir de este análisis, se desplegará la base de datos, especificando los nodos, la información que contendrán, y las relaciones entre ellos. Se desarrollará un programa en java, que simule la inyección de datos en la base de datos. Este programa deberá ser capaz de variar tanto el número de fuentes de datos como la frecuencia de inyección.
3. Se realizarán las pruebas para ver cómo reacciona el sistema ante diferentes situaciones, tratando de encontrar su "punto de ruptura" (aquel en el que el sistema deja de funcionar correctamente).
4. Por último se analizarán los tiempos de respuesta ante diferentes consultas, variando el tamaño de la base de datos y la información que contiene. Esto evaluará tanto consultas simples como complejas, y su rendimiento bajo distintas cargas. El objetivo no sería solo encontrar el punto de saturación, si no ver en qué condiciones el sistema trabaja de manera óptima.

1.3 Plan de trabajo

En este apartado se van a detallar las tareas desarrolladas para alcanzar los objetivos de este proyecto. Incluyendo las horas de trabajo, estimadas y reales, para cada tarea.

Tarea 1: Búsqueda y obtención de la información. En esta actividad inicial se estudiará la documentación de la base de datos Neo4J.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Investigación de las herramientas a usar	40	50
Total	40	50

Tabla 1: Investigación de neo4J

Tarea 2: Instalación de los diferentes componentes que serán usados a lo largo del trabajo.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Instalación de software	2	3
Total	2	3

Tabla 2: Instalación de equipos

Tarea 3: Estudio del lenguaje cypher y funcionamiento de Neo4J. En esta tarea también se tiene en cuenta el estudio de los archivos de configuración de Neo4J.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Estudio del entorno de Neo4J	10	15
Estudio del lenguaje Cypher	15	25
Total	25	35

Tabla 3: Estudio de Neo4J y cypher

Tarea 4: Comparación y decisión de herramientas a utilizar. En esta tarea se analizará y decidirá el entorno para programar, así como el lenguaje de programación. También se buscarán herramientas para la medición de consumo CPU y RAM y tiempos de respuesta.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Comparación y elección de lenguaje de programación	10	12
Comparación y elección de entorno de programación	2	2
Comparación y elección de herramientas de medición de consumo.	5	8
Comparación y elección de herramientas para medición de tiempos de respuesta	2	2
Comparación y elección de herramientas para hacer gráficas	2	4
Total	21	28

Tabla 4: Elección de herramientas

Tarea 5: Estudio de las herramientas elegidas. Esta tarea incluye el estudio de la librería de Neo4J para Java y el estudio de la librería OSHI para medición de CPU y RAM.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Estudio de librería de Neo4J	10	15
Estudio de librería OSHI	5	9
Estudio de Tableau Desktop	5	8
Total	20	31

Tabla 5: Estudio de las herramientas

Tarea 6: Desarrollo de herramientas para la automatización de las pruebas. Tanto para pruebas de consumo de CPU y RAM como para envío de datos, creación de datos *dummies*, medición de tiempos, etc.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Programa para inserción de datos	25	35
Programa de consumo de CPU y RAM	10	16
Programa para medición de tiempos	10	12
Total	45	63

Tabla 6: Tiempos de programación

Tarea 7: Realización de pruebas. En esta tarea se incluyen las pruebas de CPU y RAM, así como las pruebas de

medición de tiempos.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Pruebas de consumo de CPU	30	40
Pruebas de consumo de RAM	30	35
Pruebas de tiempos de respuesta	30	45
Total	90	120

Tabla 7: Realización de pruebas

Tarea 8: Análisis de resultados. En esta tarea se analizarán todos los resultados obtenidos en las pruebas.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Análisis de consumo de CPU	10	15
Análisis de consumo de RAM	10	15
Análisis de tiempos de respuesta	10	20
Total	30	50

Tabla 8: Análisis de resultados

Tarea 9: Redacción de la memoria del trabajo.

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Redacción de la memoria	85	90
Total	85	90

Tabla 9: Redacción de la memoria

Tiempos totales

Tarea	Tiempo estimado (horas)	Tiempo real (horas)
Total	358	474

Tabla 10: Tiempos totales

2 ESTADO DEL ARTE

En este capítulo se explicará toda la teoría relacionada con el trabajo, así como las herramientas que se han utilizado para su desarrollo.

2.1. Conceptos clave

- Testing de software- El testing de software es una actividad que pretende evaluar el correcto funcionamiento de un sistema y su rendimiento. La evaluación se realiza a través de diferentes pruebas centrándose cada una en un aspecto concreto. En este caso nos centraremos en las siguientes pruebas:
    - Prueba de rendimiento: Ponen a prueba el rendimiento de software en tiempo de ejecución. Se centran en buscar los límites del sistema, ver los puntos en los que falla y por qué.
    - Pruebas de recuperación: Fuerzan el sistema a fallar de diversos modos, y verifica que la recuperación es adecuada.
- Cuello de botella- En inglés "*bottleneck*", se refiere al punto en el que la base de datos se ralentiza o sobrecarga. Tarda más tiempo que lo que se considera normal en realizar consultas o no admite más datos.

2.2. Internet of Medical Things (IoMT)

Antes de entrar a explicar IoMT, conviene poner este concepto en contexto explicando primero el término *Internet of Things* (IoT). El Internet de las Cosas (IoT) abarca una red de dispositivos, sistemas y sensores interconectados que colaboran y se comunican entre sí, aprovechando los avances en la potencia de computación, la miniaturización de componentes electrónicos y la evolución de las redes de internet. Una característica clave del IoT es la capacidad de los dispositivos para operar de forma autónoma, recopilando y procesando datos sin intervención humana, lo que permite optimizar procesos y mejorar la eficiencia en diversos entornos. Esta red es altamente escalable, lo que permite integrar nuevos dispositivos fácilmente, y muchos de ellos cuentan con la capacidad de aprender y adaptarse a su entorno a través de inteligencia artificial. Estas características hacen del IoT una herramienta clave para optimizar procesos, mejorar la eficiencia y fomentar la innovación en una amplia variedad de sectores.

La implementación de este tipo de red busca mejorar la calidad de vida y facilitar tareas. Abarcan un gran abanico de aplicaciones como "casas inteligentes" (por ejemplo, sensores y actuadores domóticos), "vehículos interconectados" (comunicación entre vehículos de forma autónoma), etc., pero la que nos importa es la aplicación al mundo sanitario [3].

Una plataforma IoMT es un “sistema inteligente” compuesto de sensores y circuitos electrónicos para obtener señales biomédicas de un paciente a través de la red para su posterior almacenamiento (temporal o permanente) [3].

La aplicación de IoT en sanidad, permite minimizar los errores humanos, y ayuda a los profesionales a diagnosticar las enfermedades más fácilmente gracias a la monitorización de las constantes vitales en tiempo real que proporciona esta red de dispositivos.

2.3. Bases de datos basadas en grafos

Las bases de datos basadas en grafos no son más que otra manera de representar y definir una base de datos. En este tipo de bases de datos, la información se guarda en nodos, relaciones y propiedades. Los datos se guardan en nodos, que pueden tener propiedades, y cada nodo se relaciona con otros nodos a través de las relaciones.

Dado que cada nodo en una base de datos basada en grafos tiene un enlace directo a los nodos con los que está relacionado, no es necesario crear índices adicionales para buscar o acceder a esas conexiones. Esto significa que, a diferencia de las bases de datos tradicionales, donde se deben generar índices para optimizar las búsquedas y las consultas, en las bases de datos de grafos la estructura misma facilita la navegación y recuperación de datos de manera eficiente. Cada nodo actúa como un punto de acceso inmediato a su red de relaciones, lo que simplifica la gestión y aceleración de las consultas complejas.

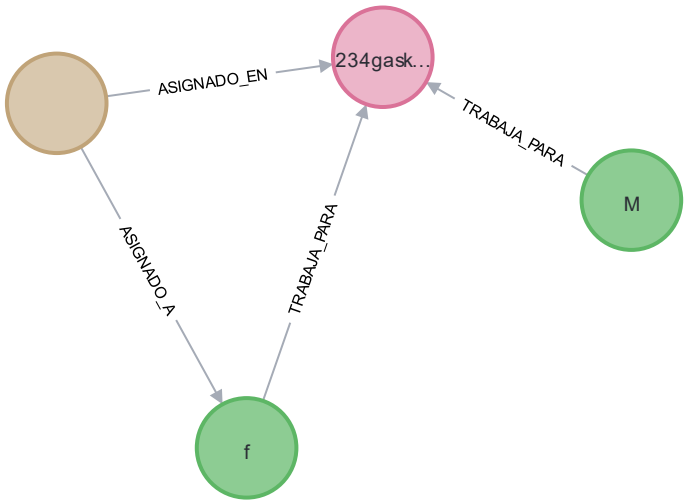


Figura 1: Visualización de nodos en Neo4j

En la figura se puede apreciar cómo se representaría la información en Neo4J, pero da una idea de las bases de datos basadas en grafos, en general. Vemos cada nodo con su información, y cómo se relacionan entre ellos. Tenemos 3 tipos de nodos en este ejemplo: médico (verde), hospital (rosa) y paciente (marrón). Un paciente estará asignado EN un hospital y A un médico y el médico trabajará para un hospital y tendrá asignados sus respectivos pacientes.

Este tipo de bases de datos se suele usar cuando las relaciones entre cada nodo cobran una vital importancia, por eso es por lo que su uso se ve mayoritariamente en redes sociales.

2.4. Recursos software utilizados

2.4.1. Neo4J

Neo4j es considerado el software de referencia en las bases de datos basadas en grafos y es una de las más utilizadas en áreas como la salud, el gobierno y el sector militar, entre otras. Es una base de datos de código abierto implementada en Java. Este software se lanzó en 2007 y tiene tres categorías: comunidad, gobierno y empresa. En este trabajo se utilizará la versión de comunidad. Neo4j permite a los usuarios modelar, almacenar y consultar datos altamente conectados de manera eficiente, lo que es esencial para aplicaciones que requieren

una representación rica de relaciones entre datos [2]. La arquitectura de Neo4j está optimizada para manejar millones de nodos y relaciones, proporcionando respuestas rápidas y escalabilidad horizontal.

Además, Neo4j ofrece una librería de Java conocida como Neo4j Java Driver, que permite a las aplicaciones Java conectarse y comunicarse con la base de datos Neo4j. Esta librería facilita la ejecución de consultas Cypher y la manipulación de datos, proporcionando una API intuitiva y fácil de usar. El driver de Java maneja la autenticación, la conexión y el manejo eficiente de sesiones, lo cual simplifica significativamente la integración de Neo4j en aplicaciones basadas en Java [4].

Neo4j Desktop es una herramienta integral diseñada para interactuar de manera eficiente con bases de datos Neo4j. Funciona como una interfaz gráfica de usuario (GUI) que facilita la administración, exploración y prueba de bases de datos Neo4j en un entorno local o remoto. Con Neo4j Desktop, los usuarios pueden crear y configurar bases de datos, ejecutar consultas en Cypher (el lenguaje de consulta de Neo4j), y visualizar los resultados en forma de grafos interactivos, lo que mejora la comprensión y análisis de las relaciones entre los datos.

2.4.2. Cypher

Cypher es el lenguaje de consulta declarativo utilizado por Neo4j. Similar al SQL en las bases de datos relacionales, Cypher está diseñado específicamente para trabajar con datos de grafos. Este lenguaje permite expresar consultas complejas de manera simple y legible, facilitando la extracción y manipulación de datos conectados. Cypher soporta patrones de búsqueda, correspondencia de nodos y relaciones, así como la actualización de datos. La capacidad de Cypher para manejar consultas sobre grafos hace que Neo4j sea una herramienta poderosa para el análisis de datos interconectados [5]. Además, Cypher proporciona soporte para operaciones de transacciones y tiene capacidades avanzadas para gestionar grandes volúmenes de datos.

Para entender mejor este lenguaje, veamos una consulta. Supongamos que estamos utilizando una base de datos Neo4j para modelar una red social y queremos encontrar a todas las personas que Alice conoce. Para lograrlo, utilizamos una consulta en Cypher como la siguiente:

```
MATCH (p:Person)-[:KNOWS]->(friend:Person)
WHERE p.name = 'Alice'
RETURN friend.name
```

Esta consulta tiene varios componentes que trabajan juntos para obtener el resultado deseado. En primer lugar, el comando MATCH se utiliza para especificar un patrón de búsqueda en el grafo. En este caso, el patrón está buscando nodos etiquetados como Person (que representamos como p) conectados por una relación llamada KNOWS hacia otros nodos, también etiquetados como Person (representados como friend).

El filtro WHERE p.name = 'Alice' se asegura de que la consulta solo considere los nodos que tienen un atributo name con el valor "Alice". Esto nos permite enfocarnos únicamente en las conexiones de Alice.

Finalmente, el comando RETURN friend.name se utiliza para devolver el nombre de las personas conectadas a Alice a través de la relación KNOWS.

Por ejemplo, si en la base de datos hay nodos para Alice, Bob y Carol, y existen relaciones de KNOWS entre Alice y estos dos, la consulta devolverá los nombres "Bob" y "Carol". Esto permite comprender rápidamente las conexiones de Alice en la red social.

La gran ventaja de Cypher es su legibilidad. Su diseño declarativo y centrado en patrones facilita la comprensión y el análisis de datos interconectados, haciendo que sea una herramienta poderosa para trabajar con bases de datos basadas en grafos como Neo4j.

2.4.3. Eclipse

Eclipse es un entorno de desarrollo integrado (IDE) ampliamente utilizado en la programación en Java y otros lenguajes. Es una plataforma de código abierto que ofrece un conjunto de herramientas extensibles para el desarrollo de software. Eclipse proporciona una interfaz amigable y diversas funcionalidades que ayudan en la

gestión del código, depuración y pruebas. En este proyecto, Eclipse se utiliza para desarrollar y gestionar scripts y aplicaciones que interactúan con Neo4j, facilitando la implementación de consultas y el análisis de datos de la base de datos [6]. Su ecosistema de plugins permite la integración con herramientas adicionales y frameworks, lo cual mejora la eficiencia del desarrollo.

2.4.4. Librería OSHI

OSHI (Operating System and Hardware Information) es una biblioteca de código abierto en Java que proporciona información sobre el sistema operativo y el hardware. OSHI es útil para obtener detalles del entorno en el que se ejecuta la aplicación, como la memoria disponible, el uso de CPU y otros recursos del sistema. Esta información es valiosa para monitorizar y optimizar el rendimiento de las aplicaciones que interactúan con la base de datos Neo4j. Utilizando OSHI es posible recopilar métricas y realizar ajustes para asegurar un rendimiento óptimo del sistema [7]. Además, OSHI facilita la detección de posibles cuellos de botella y permite implementar estrategias proactivas de gestión de recursos.

2.4.5. Tableau

Tableau es una potente herramienta de visualización de datos ampliamente utilizada en el análisis de grandes volúmenes de información. Su capacidad para convertir datos complejos en gráficos interactivos y dashboards facilita la comprensión y el análisis de patrones y tendencias. Tableau se integra con diversas fuentes de datos, incluyendo bases de datos relacionales, hojas de cálculo, y herramientas de big data, lo que permite a los usuarios extraer, transformar y visualizar datos de manera eficiente. Una de las características más destacadas de Tableau es su interfaz intuitiva, que permite a los usuarios crear visualizaciones sin necesidad de conocimientos avanzados en programación, lo que lo convierte en una herramienta accesible tanto para analistas de datos como para usuarios no técnicos. [8]

3 EVALUACIÓN EXPERIMENTAL

La evaluación experimental analizará el rendimiento de Neo4J (en su versión 1.5.9) cuando se somete a distintos escenarios. Los parámetros que probaremos serán el consumo de CPU y RAM mientras se añaden datos desde un diferente número de dispositivos y a diferente frecuencia y la variación del tiempo de consultas con la variación del número de nodos.

3.1 Setup experimental.

Puesto que los resultados varían dependiendo del entorno, debemos describir la plataforma con la que se han realizado los experimentos. El hardware usado para la evaluación ha sido un ordenador portátil con las siguientes características:

- Ordenador: Lenovo IdeaPad S540
- 8GB de memoria RAM
- Intel(R) Core (TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
- Windows 11
- Neo4J versión 1.5.9
- Eclipse IDE 2023-12 (4.30.0)

3.2 Base de datos

Para poder llevar a cabo el análisis, se ha diseñado una base de datos apta para poder realizar las pruebas.

3.2.1 Estructura

3.2.1.1 Nodos

La base de datos cuenta con 6 tipos de nodos, diferenciados por etiquetas (cada nodo puede tener una o más etiquetas). Los tipos de nodos son los siguientes:

- :Person:Medico → Guarda los datos de un médico y tiene los siguientes atributos: **dni**[1], nombre, apellidos, edad, sexo y departamento.
- :Person:Paciente → Recopilará los datos de un paciente y tendrá como atributos **dni**, nombre, apellidos, edad, sexo y patología.

[1] Los atributos marcados en negrita son claves primarias.

- :Hospital → Registra la información de un hospital, guardando un **id**, nombre y dirección.
- :Dispositivo → Este es el dispositivo que realiza la monitorización de cada paciente y tendrá como atributos: **id**, modelo y fecha_inicio (fecha en la que se empezaron a monitorizar los datos).
- :Observación → Cada observación representa una medida de un dispositivo y contará con los siguientes atributos: dato (el valor de la medida), tipo_dato (qué está midiendo, e.g., frecuencia cardíaca, nivel de glucosa en sangre, etc.) y un ***timestamp*** que lo identificará.

La etiqueta Person se utiliza tanto para los médicos como para los pacientes, ya que tienen una clave primaria común, el dni. Se utiliza esta etiqueta para que un médico no pueda tener el mismo dni que un paciente.

3.2.1.2 Relaciones

La potencia de las bases de datos basadas en grafos se encuentra aquí, en las relaciones entre los nodos. Mientras que en las bases de datos relacionales habría que usar "claves externas", aquí se relacionan directamente unos nodos con otros, lo que hace mucho más eficiente y fácil la realización de consultas. Así es como se relacionan los nodos en el diseño que hemos realizado:

- Paciente → [Asignado_A] →Médico
  - Un paciente será asignado a un médico, de manera que el médico puede acceder a los datos del paciente.
- Paciente → [Asignado_En] → Hospital
  - El paciente también estará asignado a un hospital.
- Médico → [Trabaja_Para] → Hospital
  - Cada médico trabajará para cierto hospital.
- Paciente → [Monitorizado_Por] → Dispositivo
  - Un dispositivo monitoriza un paciente.
- Médico → [Consulta] → Dispositivo
  - Cada médico podrá acceder a los dispositivos de sus pacientes.
- Observación→ [Pertenece_A] → Paciente
  - Esta relación, indica a qué paciente pertenece dicha observación
- Observación → [Realizada por] → Dispositivo
  - Aquí se indica qué dispositivo realizó dicha observación.

Tras ver el diseño de la base de datos es interesante poner una imagen de ejemplo para poder visualizarlo más fácilmente:

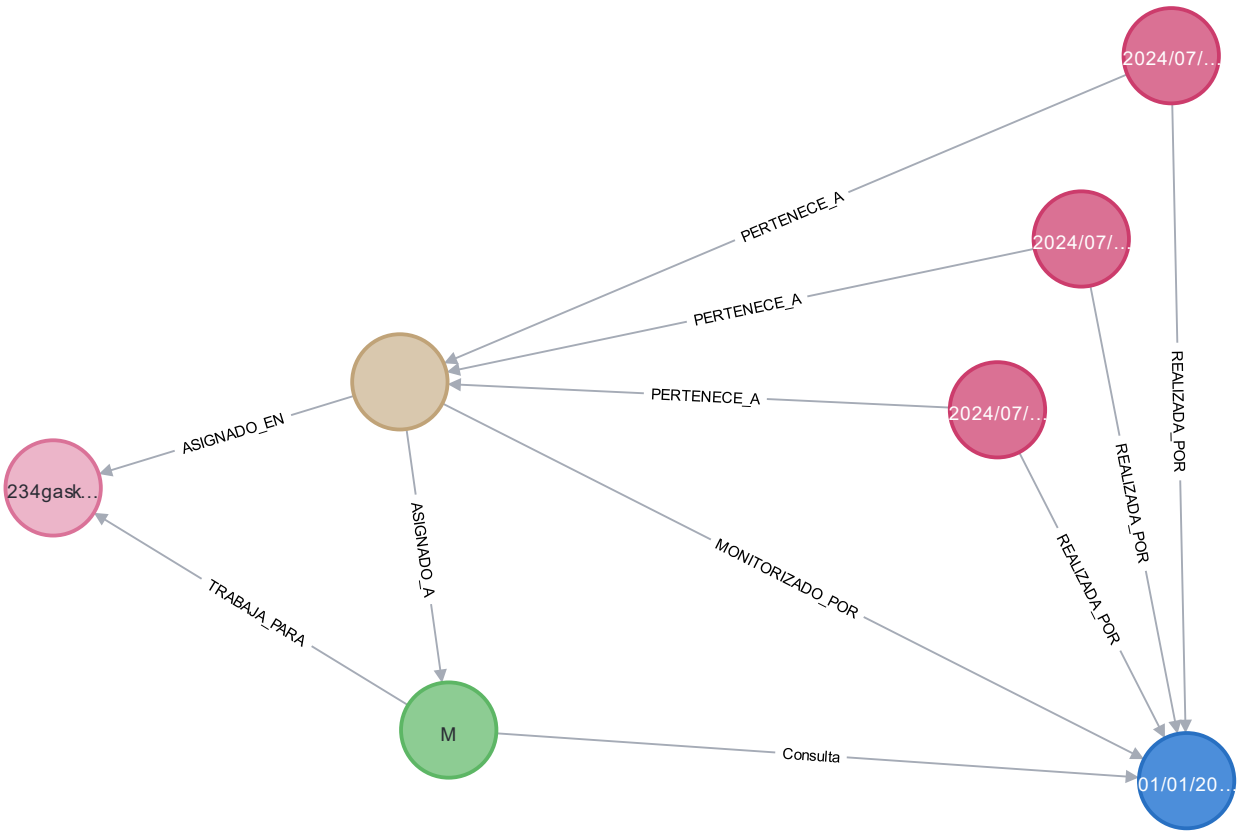


Figura 2: Nodos y relaciones de la base de datos

Comentarios sobre la imagen:

- El nodo rosa claro representa el hospital
- Los nodos rosa oscuro son las observaciones
- El nodo verde representa un médico
- El nodo azul sería un dispositivo
- El nodo marrón es el nodo paciente
- La dirección de las flechas en este caso no tiene importancia.

3.3 Proyecto en eclipse

Como se comentó previamente en los objetivos, la idea es mandar datos a distinta frecuencia y desde un variable número de dispositivos. El primer problema que nos encontramos fue crear archivos "dummies" (cada archivo representaría un dispositivo mandando datos) para la posterior inserción en la base de datos. En una primera instancia, para hacer pruebas, se realizó a mano un fichero del que coger los datos, pero esto no iba a ser suficiente para las pruebas que se tenían que realizar posteriormente. Por esta razón, se desarrolló un script que generaba los ficheros con los datos deseados.

3.3.1 Creación de archivos

El formato que queríamos conseguir era el siguiente:

```
dato1        Tipo1        12/07/2024 13:12   000000000B   000111a
```

donde la primera columna es el dato medido, la segunda columna el tipo de dato, y la tercera columna la marca del tiempo (atributos del nodo Observación). Las dos últimas columnas son para poder relacionar el nuevo nodo

Observación con los nodos Paciente y Dispositivo. La cuarta columna es el DNI del paciente al que pertenece la observación y la quinta el dispositivo que la realizó.

El script nos ha permitido crear todos los archivos que hemos necesitado, y con las líneas deseadas para realizar las pruebas. Creando hasta 349 archivos de 20.000 líneas cada uno, cosa que de forma manual hubiera sido imposible.

Este script ha sido creado en Eclipse con java, usando la librería java.io. El resultado es un archivo CSV para su posterior lectura por otro programa.

3.3.2 Envío de datos

Para la inserción de los datos, la idea era hacer un programa que mandara los datos previamente creados a la base de datos. Para ello, los creadores de Neo4J disponen de una librería en java, que proporciona las clases y los métodos necesarios para su conexión. Dado que el proyecto en eclipse era un proyecto Maven, se incluyó la dependencia de esta librería para su posterior uso como se muestra en la Figura 3.

```
<dependency>
 <groupId>org.neo4j.driver</groupId>
  <artifactId>neo4j-java-driver</artifactId>
  <version>5.12.0</version>
</dependency>
```

Figura 3: Dependencias

En un inicio se desarrolló todo el programa en el "main", pero de esa manera no podíamos simular varios dispositivos mandando datos simultáneamente, por lo que se decidió dividir el programa en hilos, cada hilo simulando un dispositivo.

3.3.2.1 Main

La base de datos proporciona una URL local, con el puerto en el que escucha. La base de datos restringe el acceso, por lo que para el acceso debemos proporcionar un nombre de usuario y una contraseña.

```
//Establecemos la conexion con la base de datos
Driver driver = GraphDatabase.driver(dbUri, AuthTokens.basic(dbUser, dbPassword));
driver.verifyConnectivity();
```

Figura 4: Inicialización del driver de Neo4J

Estas dos líneas crean el driver que proporciona la conexión a la base de datos y verifican la conexión del mismo. Si la base de datos estuviera apagada, la segunda línea pararía el programa.

Posteriormente el programa rescata todos los archivos creados previamente.

```
List<String> filepaths = getCsvFilesFromDirectory(directoryPath);
```

En este punto, el programa pregunta desde cuántos dispositivos queremos mandar los datos y a qué frecuencia. El número de dispositivos está limitado al número de archivos con datos.

```
for (int contador=0;contador<nDisp;contador++) {
    PruebaDemo pruebaDemo = new PruebaDemo(filepaths.get(contador), driver, freq, latch);
    thread = new Thread(pruebaDemo);
    System.out.println(contador);
    thread.start();
}

latch.await(); // Esperar a que todos los hilos terminen
```

Figura 5: Inicialización hilos

En este punto se crea un hilo por cada dispositivo, y la última línea que se ve abajo, hace que no termine el proceso principal hasta que todos los hilos hayan terminado.

3.3.2.2 Hilos

Cada hilo abrirá un archivo distinto, ya que si usaran los mismos archivos daría problemas al añadirlos y una observación podría pertenecer a más de un paciente y más de un dispositivo, cosa que en un caso real no debería pasar.

Una vez el hilo tiene el archivo abierto, ejecutará la siguiente sentencia Cypher:

```
(1) MATCH(p :Person {dni :$dni})
(2) MATCH(d: Dispositivo {id :$id})
(3) CREATE (obs1:Observacion {dato: $dato, tipo_dato: $tipo_dato, timeStamp: $timeStamp})
(4) MERGE (obs1)-[r:PERTENECE_A]->(p)
(5) MERGE(obs1)-[r2:REALIZADA_POR]->(d)
```

La sentencia (1) busca el nodo Person (que será un paciente) por el dni y lo almacena en una variable p. La sentencia (2) busca el Dispositivo por su id, y lo almacena en una variable d. (3) crea el nodo observación, con todos sus atributos. (4) y (5) crean las relaciones entre ambos nodos.

El programa ejecutaría estas instrucciones en bucle, pasando por todas las filas de los archivos, hasta quedarse sin filas. Así, cada paso añadiría un nuevo nodo Observación.

3.3.3 Insertar pacientes

La inserción de pacientes y dispositivos en la base de datos fue un paso crucial para garantizar la efectividad y la robustez del sistema en situaciones de prueba. Inicialmente, la base de datos estaba limitada a 10 nodos, una cantidad claramente insuficiente para simular el comportamiento del sistema bajo condiciones de alta carga o en escenarios que requieran un mayor nivel de complejidad en la interacción entre nodos. Para evitar esta limitación, se desarrolló un script automatizado capaz de generar y agregar nodos de manera masiva, incrementando significativamente el número de pacientes y dispositivos en la base de datos. Este proceso no solo amplió la capacidad de la base de datos para manejar un mayor volumen de información, sino que también permitió simular entornos mucho más realistas, donde se pueden evaluar aspectos críticos como la latencia en la transmisión de datos, la eficacia de las relaciones entre nodos y la capacidad de respuesta del sistema bajo condiciones extremas.

Además, la generación automática de estos nodos y sus relaciones es fundamental para las fases de prueba y validación del sistema. Aunque los datos generados sean artificiales, cumplen con el propósito esencial de proporcionar un entorno de simulación complejo y detallado, donde se pueden realizar experimentos controlados y medir con precisión el rendimiento y la estabilidad de la base de datos. El script no solo se limita a insertar nodos, sino que también establece las relaciones entre los pacientes y los dispositivos, replicando de manera más fiel el comportamiento real que se espera del sistema en producción. Al preparar la base de datos de esta manera, se asegura que el entorno esté listo para manejar pruebas a gran escala, permitiendo identificar y resolver posibles cuellos de botella o fallos antes de que el sistema sea desplegado en un entorno real. Esta preparación es esencial

para asegurar que, en situaciones reales, la base de datos funcione de manera eficiente y sin contratiempos, garantizando la integridad y la fiabilidad de los datos manejados.

3.3.4 Monitorización

La monitorización del rendimiento del sistema es un aspecto fundamental en cualquier proyecto que implique el procesamiento intensivo de datos, como es el caso de la simulación y envío de información en bases de datos. Durante el desarrollo, surgió la necesidad de observar en tiempo real el consumo de recursos del sistema, específicamente el uso de CPU y memoria RAM, para asegurar que el sistema pudiera manejar la carga sin comprometer su estabilidad y eficiencia.

Inicialmente, se buscó algún software de uso libre que permitiera medir estos parámetros, pero tras una exhaustiva búsqueda, no se encontró ninguna herramienta que cumpliera con los requisitos específicos del proyecto. Esto llevó a la exploración de alternativas, resultando en la elección de la librería OSHI (*Operating System and Hardware Information*) en Java. OSHI es una biblioteca de código abierto que permite acceder a información detallada sobre el hardware y el sistema operativo del dispositivo en el que se ejecuta la aplicación. Esta librería resultó ser la solución ideal para el monitoreo, dado que permite recopilar datos precisos sobre el uso de CPU y memoria, así como otros recursos del sistema.

Para implementar la monitorización, se desarrolló un script en un proyecto separado de Eclipse que se ejecuta en paralelo al script principal de envío de datos. Esto se hizo con el objetivo de medir el consumo de recursos en tiempo real mientras se ejecutan las pruebas. Este script se encarga de medir el porcentaje de CPU y RAM usados, mide cada medio segundo y los inserta en un archivo CSV. Archivo que posteriormente se abre en Tableau para poder visualizar las gráficas correspondientes.

3.4 Análisis de rendimiento de CPU y RAM

Para poder entender el comportamiento y saber la eficiencia que tiene Neo4j, hemos hecho este análisis bajo diferentes cargas de trabajo. En este apartado se examinará cómo la carga que le proporcionemos a la base de datos afecta a los recursos que demanda la misma.

Para llevar a cabo este análisis, se han realizado pruebas simulando diferentes volúmenes de datos, lo que nos llevará a ver el límite de la base de datos.

Este análisis no solo proporciona información sobre la eficiencia de Neo4j, sino que también ofrece claves sobre cómo optimizar la configuración del sistema y las consultas Cypher para mejorar el rendimiento. Se presentarán gráficos y tablas con los resultados obtenidos, seguidos de una discusión sobre las posibles optimizaciones y mejores prácticas para gestionar los recursos de manera eficiente

Para poner en contexto, y explicar la estructura que van a llevar a las gráficas, se muestran a continuación dos imágenes del consumo de CPU y RAM en la situación en que no se manda ningún tipo de dato. Esto nos permitirá conocer el punto de partida del sistema. En concreto podemos observar que el consumo de CPU en reposo sí es bajo, pero el de RAM es alto, aunque esté en reposo debido a que el equipo del setup tiene tan solo 8Gb de RAM.

Como se ve en la gráfica, el consumo medio es bastante alto, por lo que el consumo de RAM no se verá bien reflejado en este estudio. El eje horizontal indica el tiempo (en segundos) en el que se realizó la medida, y el eje vertical, el porcentaje de consumo (RAM o CPU, dependiendo del caso). En las gráficas de consumo de RAM además se ha añadido una línea de tendencia, indicando el consumo promedio.

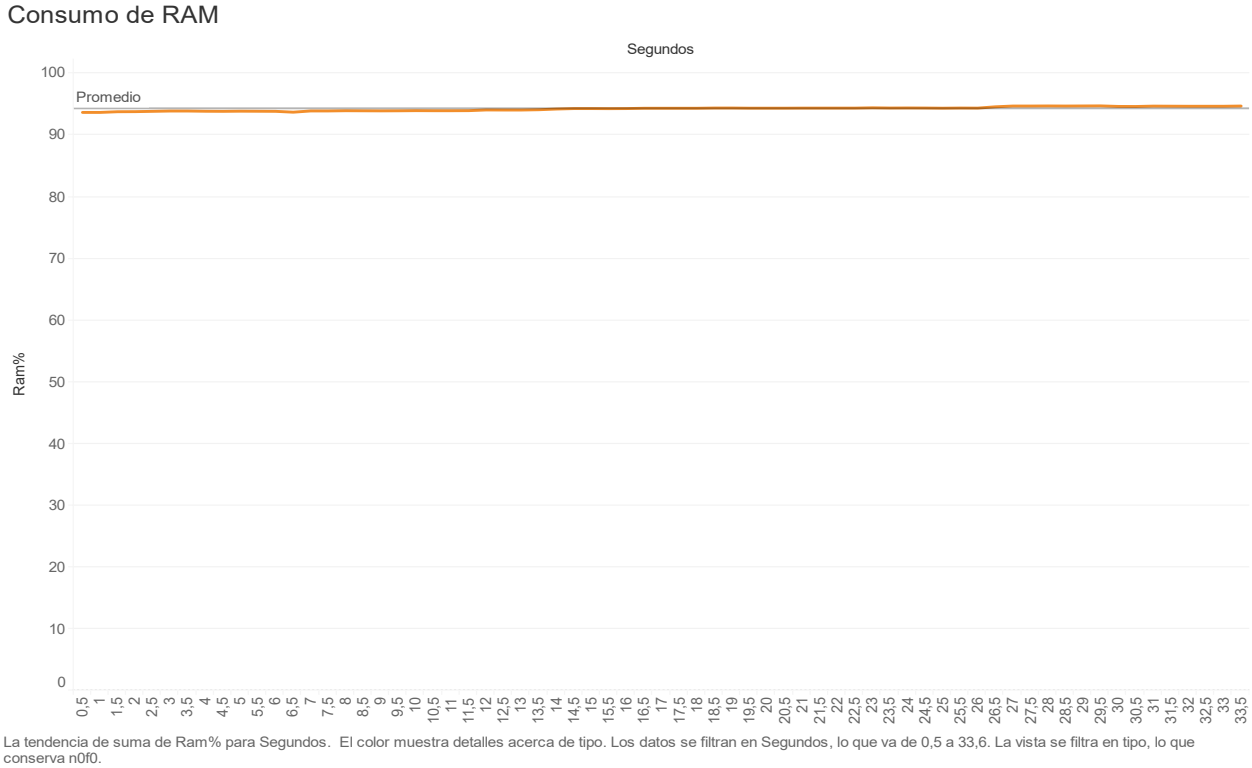


Figura 6: n0f0 RAM

Conviene aclarar la notación del título de las gráficas, como en n0f0. El número detrás de la "n" indica el número de dispositivos desde los que se mandan datos, mientras que el número detrás de la "f" indica la frecuencia [2] a la que se envían los mismos.

[2] Aunque se utilice el término "frecuencia" en el documento, nos referimos al tiempo transcurrido entre dos datos consecutivos para el mismo dispositivo.

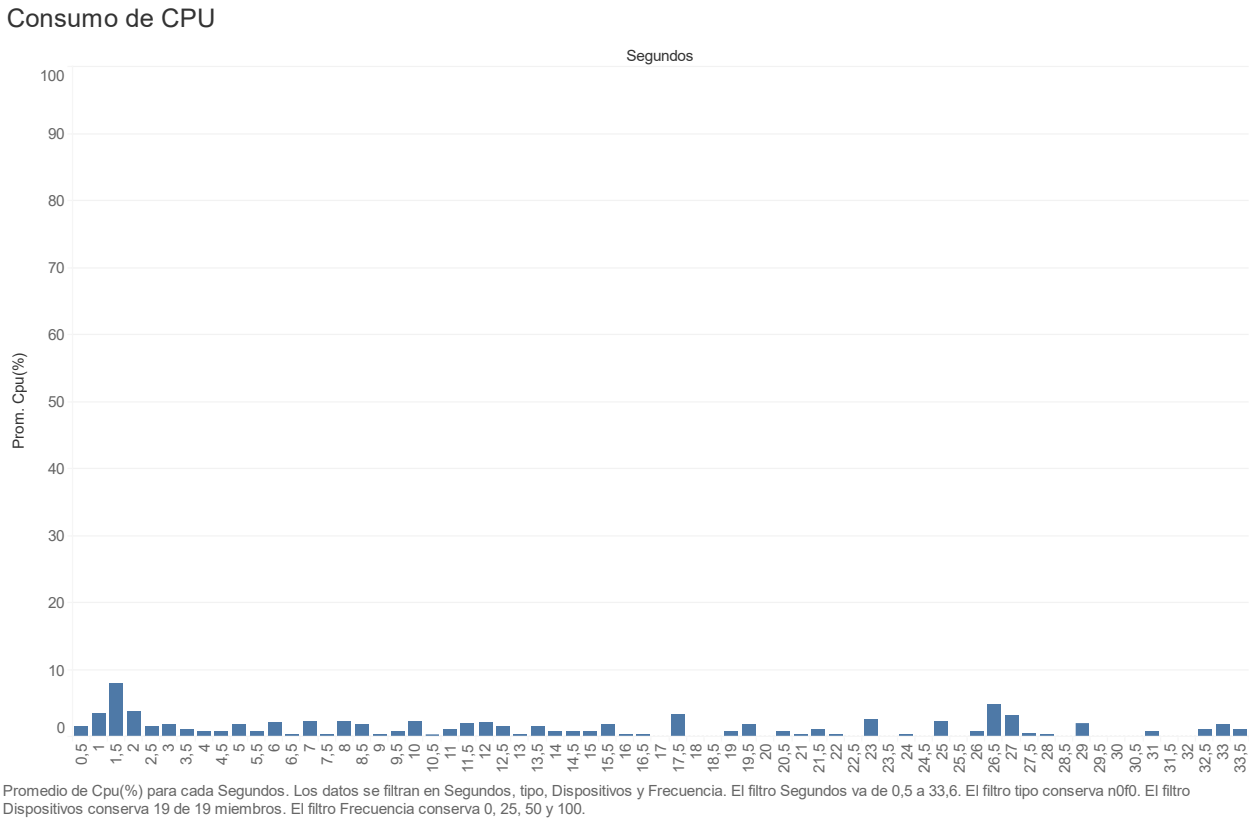


Figura 7: n0f0 CPU

La Figura 7 refleja lo que se esperaba: un consumo de CPU bajo en las condiciones iniciales de reposo sin actividad.

Antes de proceder al análisis, es preciso comentar que, cuando la tasa de envío es alta puede dar problemas de concurrencia a la hora de escribir los datos. Se ha obviado este problema a la hora de hacer las pruebas y los resultados, ya que el número de datos perdidos era menor al 1% y se ha considerado que su efecto en el rendimiento es despreciable.

3.4.1 Consumo de CPU con frecuencia 100 ms

El primer bloque de pruebas se llevó a cabo con una frecuencia de 100 ms por dato, y se incrementó el número de dispositivos de manera gradual. En un principio, se escogió un aumento del número de dispositivos conservador, haciéndolo de forma incremental para evitar sobrecargar el sistema rápidamente. Sin embargo, pronto fue evidente que, con esta frecuencia, se necesitaba un número considerable de dispositivos para que el impacto en el consumo de CPU fuera notable.

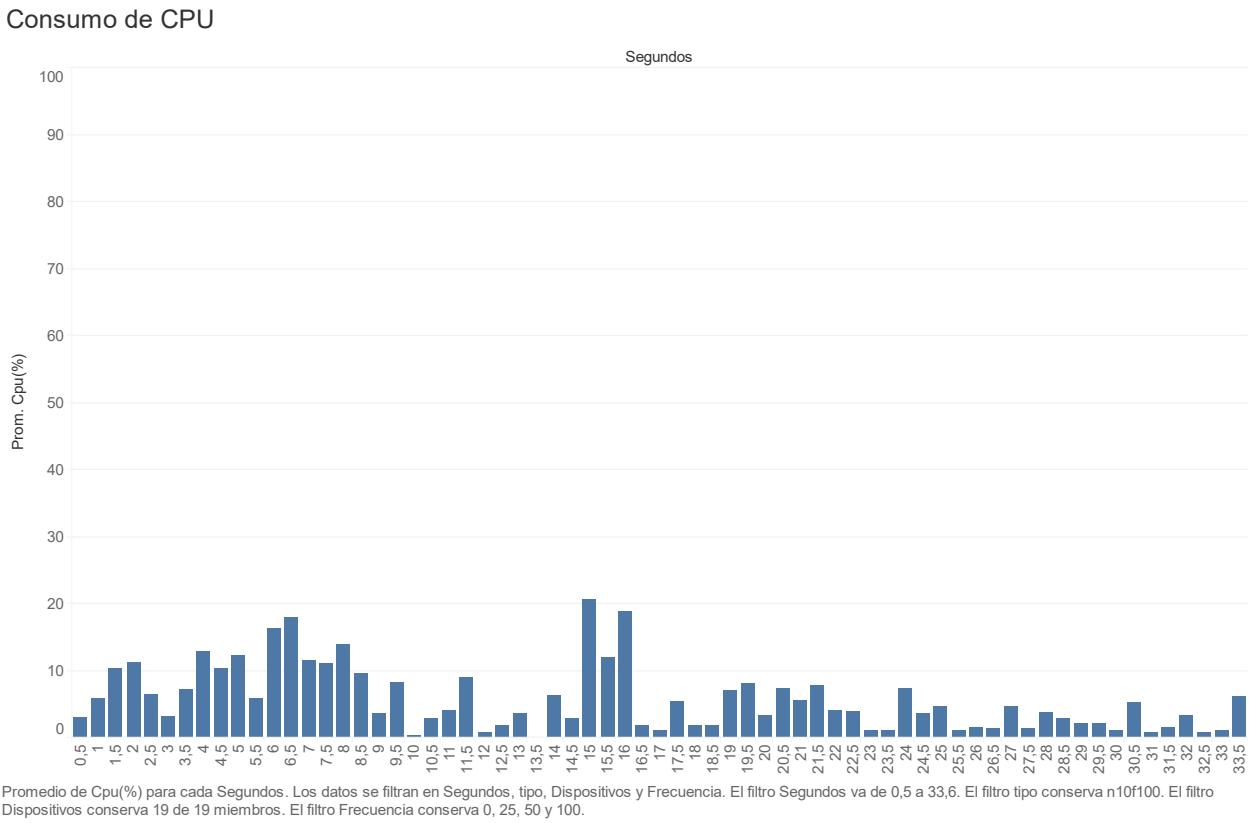


Figura 8: n10f100 CPU

En comparación con la gráfica anterior, se evidencia que, de media, consume más CPU (1,21% vs 5,7%) y llegando a picos más altos (7,88% vs 20,56%). Pero nada significante, un cambio normal dado que antes la base de datos no estaba trabajando prácticamente.

Las pruebas con 15 y 20 dispositivos daban picos similares 17,28% y 20,87, respectivamente. Lo que indica que todavía esto no suponía un esfuerzo para Neo4J porque además el consumo medio de CPU bajó a 3,68% con 20 dispositivos.

Al elevar los dispositivos a 25 manteniendo la frecuencia en 100 ms se pudieron observar comportamientos interesantes (Figura 9).

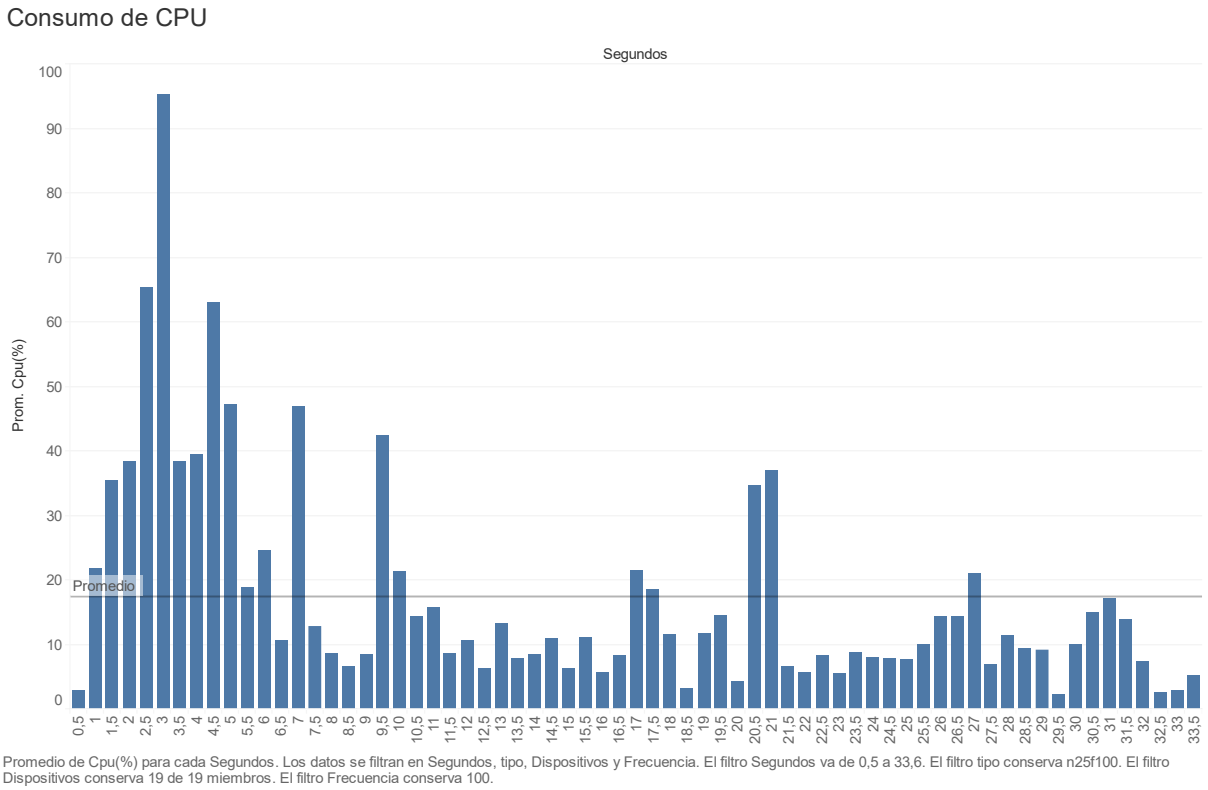


Figura 9: n25f100 CPU

Es notable ver el pico que hay en el segundo 3 de estar transmitiendo datos, sin embargo, es algo instantáneo ya que posteriormente vuelve a unos valores más razonables. Es interesante, ya que en la Figura 10 con 30 dispositivos no llega a un pico tan alto, ni se acerca directamente. Alcanza un máximo de 53,8%, insignificante en comparación con los 95,29% alcanzados en el anterior gráfico.

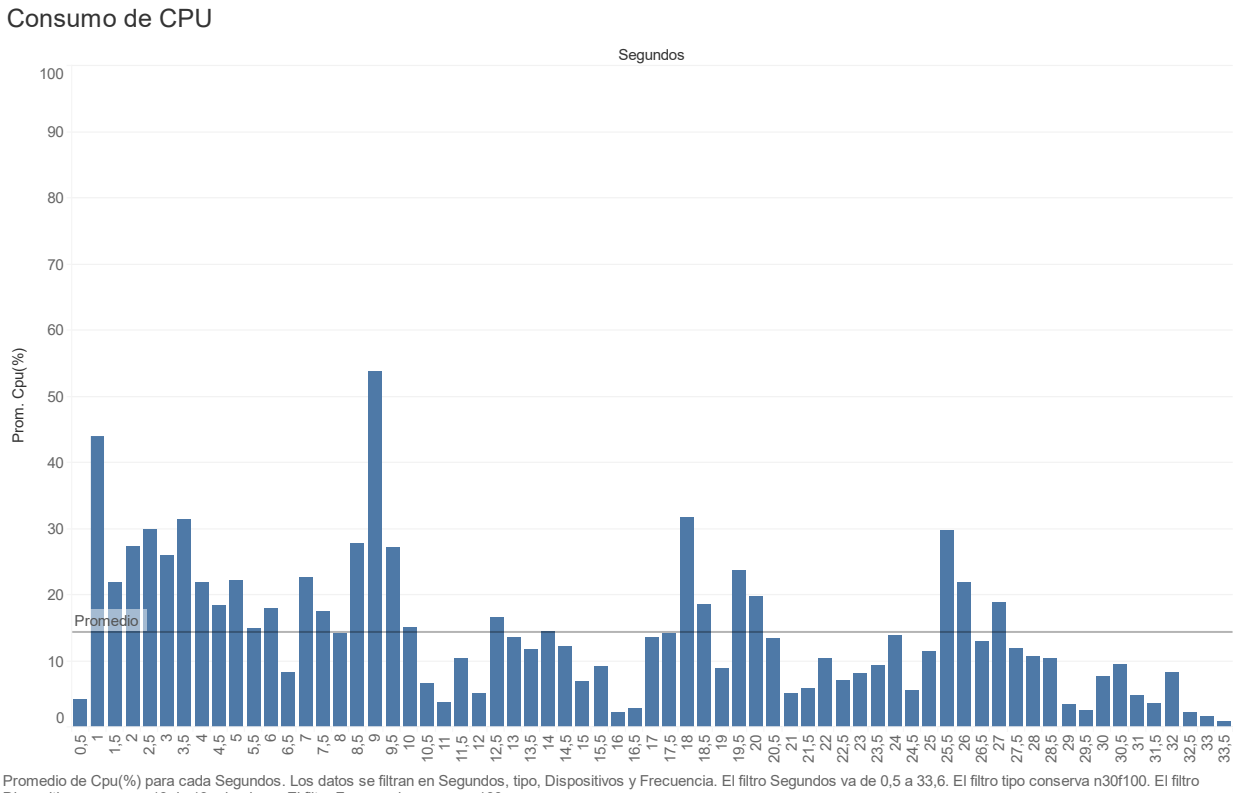


Figura 10: n30f100 CPU

En la Figura 11 se representa el consumo de CPU con 100 dispositivos y misma frecuencia, sin obtener cambios notables. Se puede comprobar como sigue aceptando bien la cantidad de datos, sin llegar a saturar demasiado.

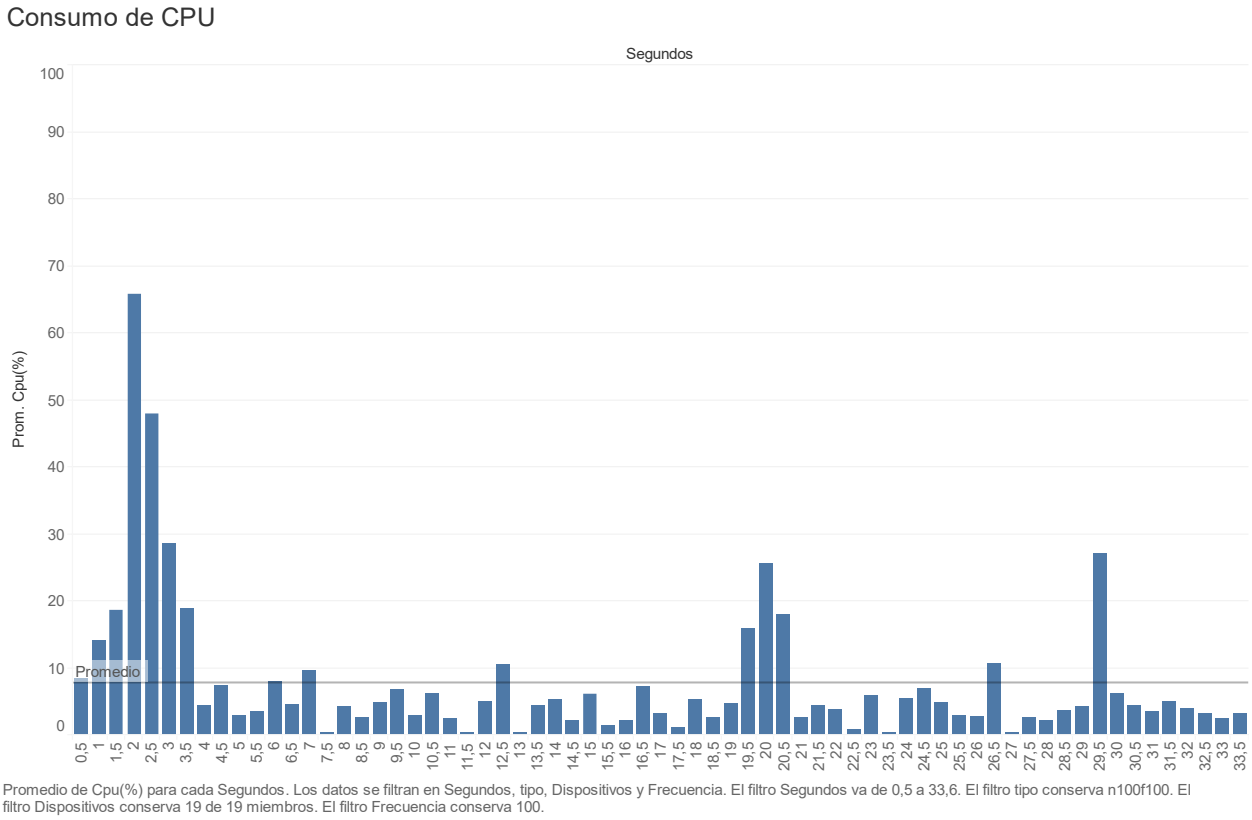
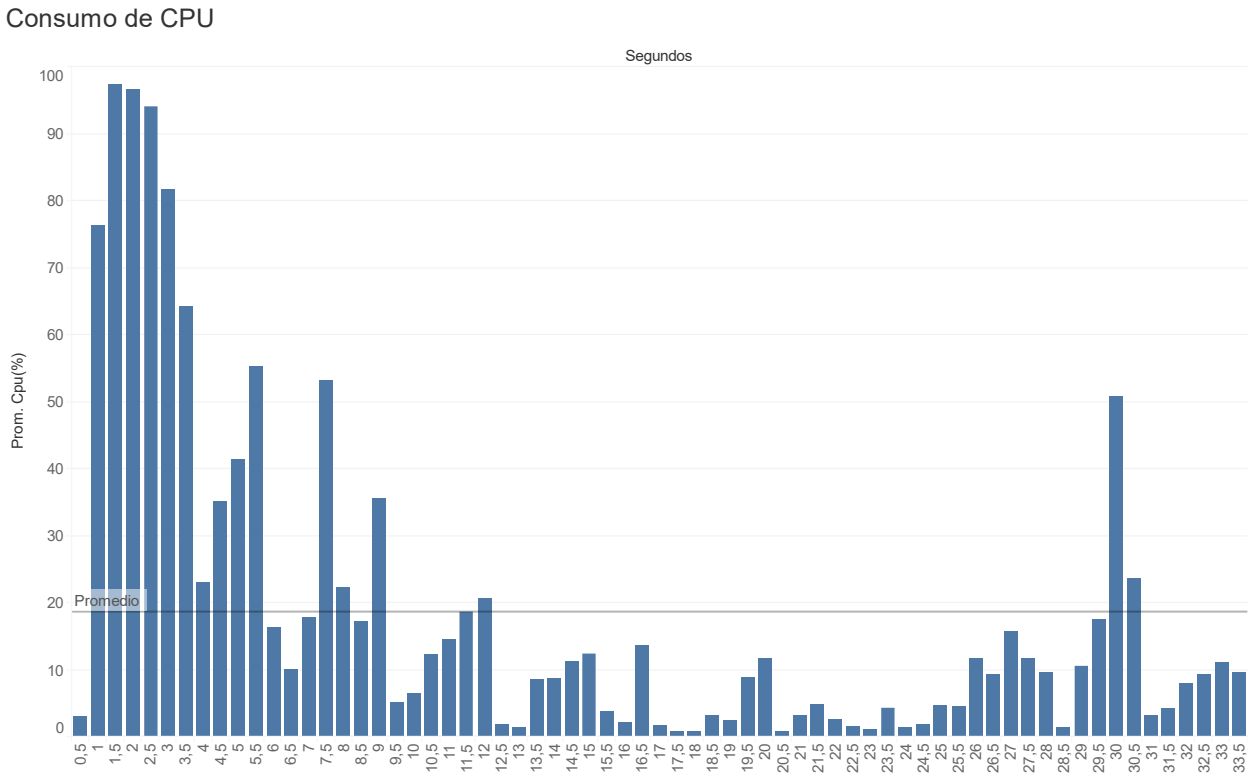


Figura 11: n100f100 CPU

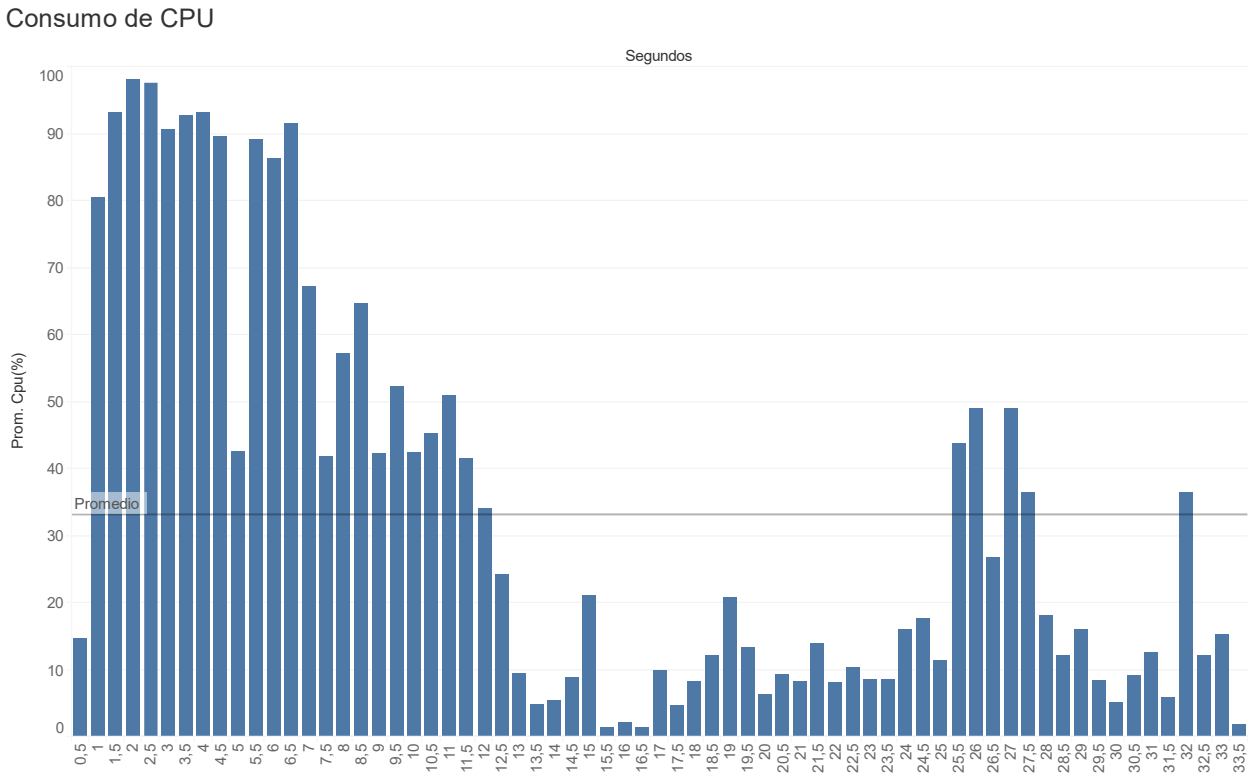
La situación comienza a cambiar cuando aumentamos el número de dispositivos a 149 (Figura 12) ya que es la primera vez que se mantiene por encima de 75% alrededor de 3 segundos, llegando a un pico de 97,33% de uso de CPU, algo que se podría considerar peligroso en el caso en el que estuviéramos hablando de servidores.

Posteriormente, observamos con 200 dispositivos un pico de 97,88%, pero un pico momentáneo que se estabiliza rápidamente (Figura 13).



Promedio de Cpu(%) para cada Segundos. Los datos se filtran en Segundos, tipo, Dispositivos y Frecuencia. El filtro Segundos va de 0,5 a 33,6. El filtro tipo conserva n149f100. El filtro Dispositivos conserva 19 de 19 miembros. El filtro Frecuencia conserva 100.

Figura 12: n149f100 CPU



Promedio de Cpu(%) para cada Segundos. Los datos se filtran en Segundos, tipo, Dispositivos y Frecuencia. El filtro Segundos va de 0,5 a 33,6. El filtro tipo conserva n300f100. El filtro Dispositivos conserva 19 de 19 miembros. El filtro Frecuencia conserva 100.

Figura 13: n200f100

Cuando se alcanzan los 300 dispositivos sí se puede empezar a observar un cuello de botella, en el que se mantiene el consumo de CPU por encima del 80% durante 6,5 segundos.

En la primera etapa de pruebas, vemos que la base de datos empieza a saturar seriamente con 300 dispositivos, lo que son en total unos 3000 datos enviados por segundo. Cada dato enviado es un nuevo nodo creado y dos relaciones nuevas.

3.4.2 Consumo de CPU con frecuencia 50 ms

La segunda parte de las pruebas se realizaron mandando un dato cada 50 ms, es decir, cada dispositivo manda 20 datos por segundo.

Empezamos con 49 dispositivos (Figura 14). Dada la estabilidad del consumo, obviamos las pruebas con menos dispositivos, ya que no se ve nada interesante en los resultados. En la Figura 15 podemos observar un pico de 62,57% pero que rápidamente disminuye, conservando un promedio de 5,89%. Tiene sentido que aguante la base de datos a pesar del incremento de frecuencia, ya que ahora mismo se estarían mandado 980 datos por segundo[3] cuando, anteriormente el problema empezó a los 3000 d/s.

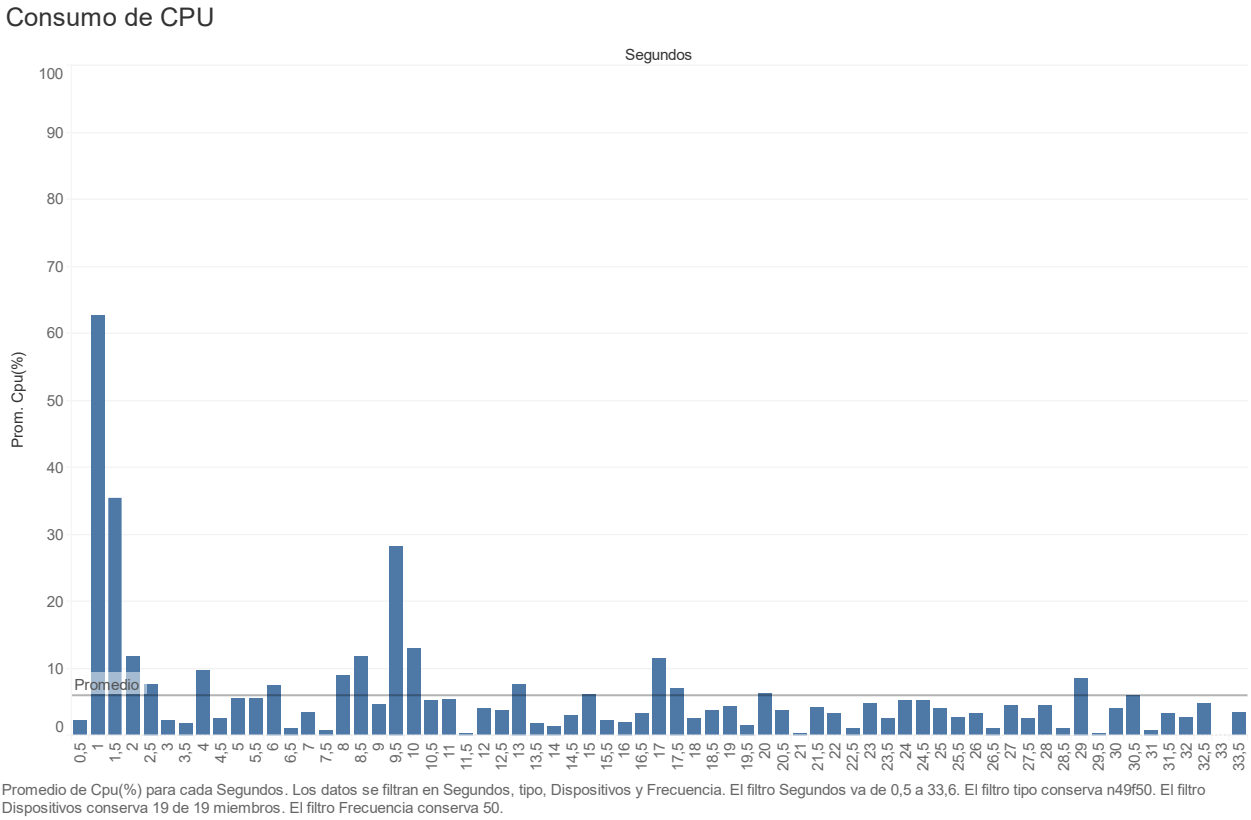


Figura 14: n49f50 CPU

[3] A partir de este momento, se le abreviará datos por segundo a "d/s" para una mayor brevedad.

Siguiendo con el siguiente gráfico:

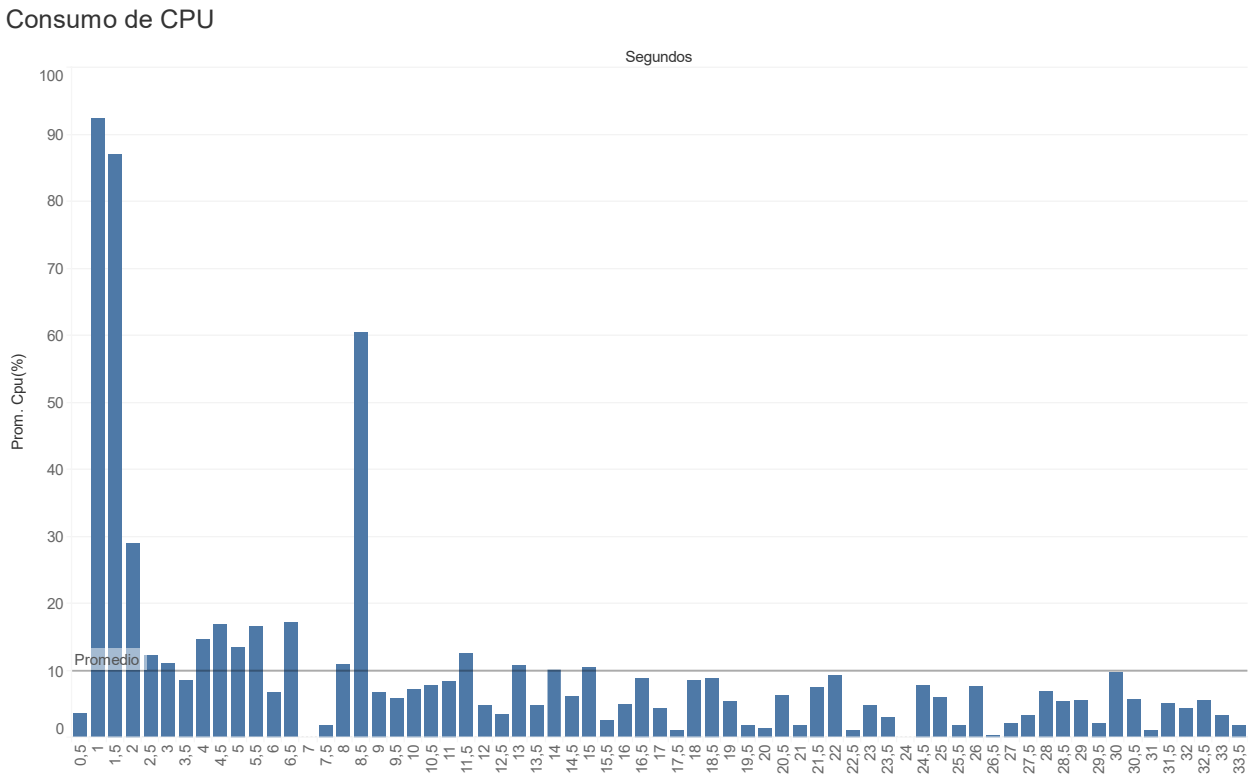


Figura 15: n100f50 CPU

Los datos que nos presenta este gráfico nos permiten ver dos picos altos de 92% y 86%, respectivamente, pero que no perduran en el tiempo. Salvo el otro pico de 60%, la base de datos soporta esta cantidad de dispositivos y esa frecuencia de envío.

La siguiente prueba nos permite ver dos picos en vez de uno, lo que indica que Neo4j empieza a necesitar más

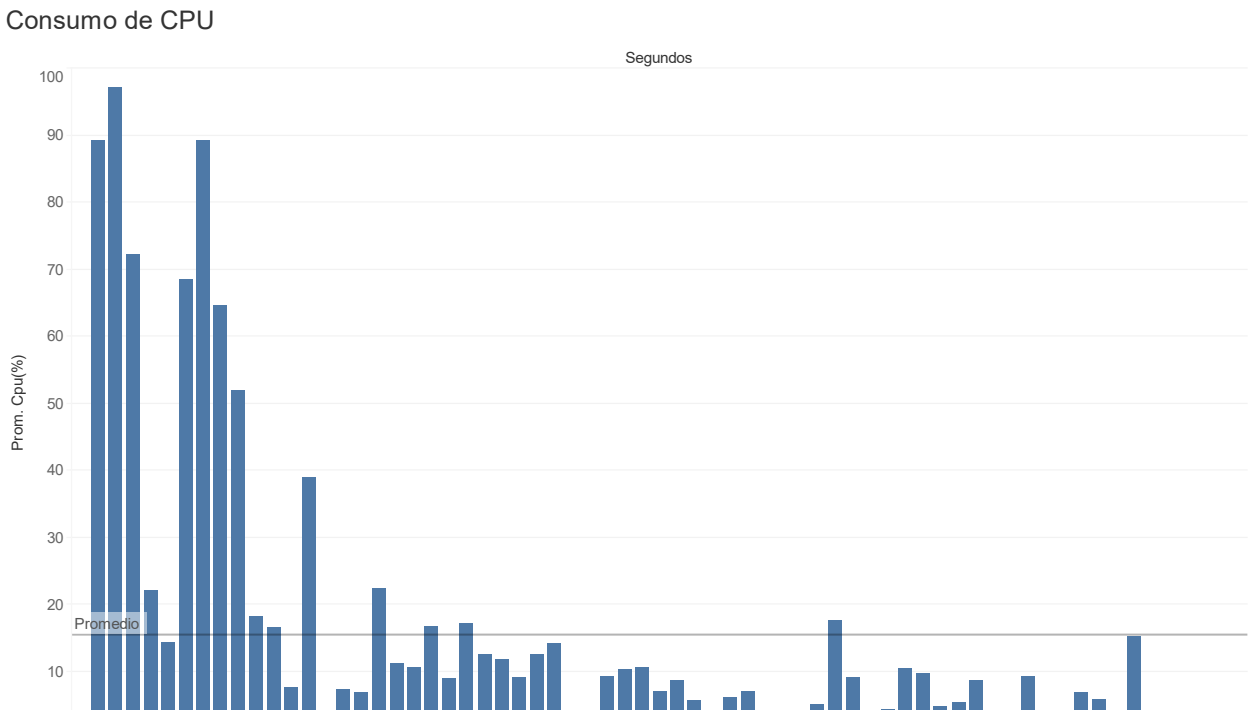


Figura 16: n149f50 CPU

recursos para poder procesar la información (Figura 16).

En este punto casi hemos alcanzado el punto de ruptura anterior, mandando 2980 d/s. Pero parece ser que a esta frecuencia el sistema aguanta mejor.

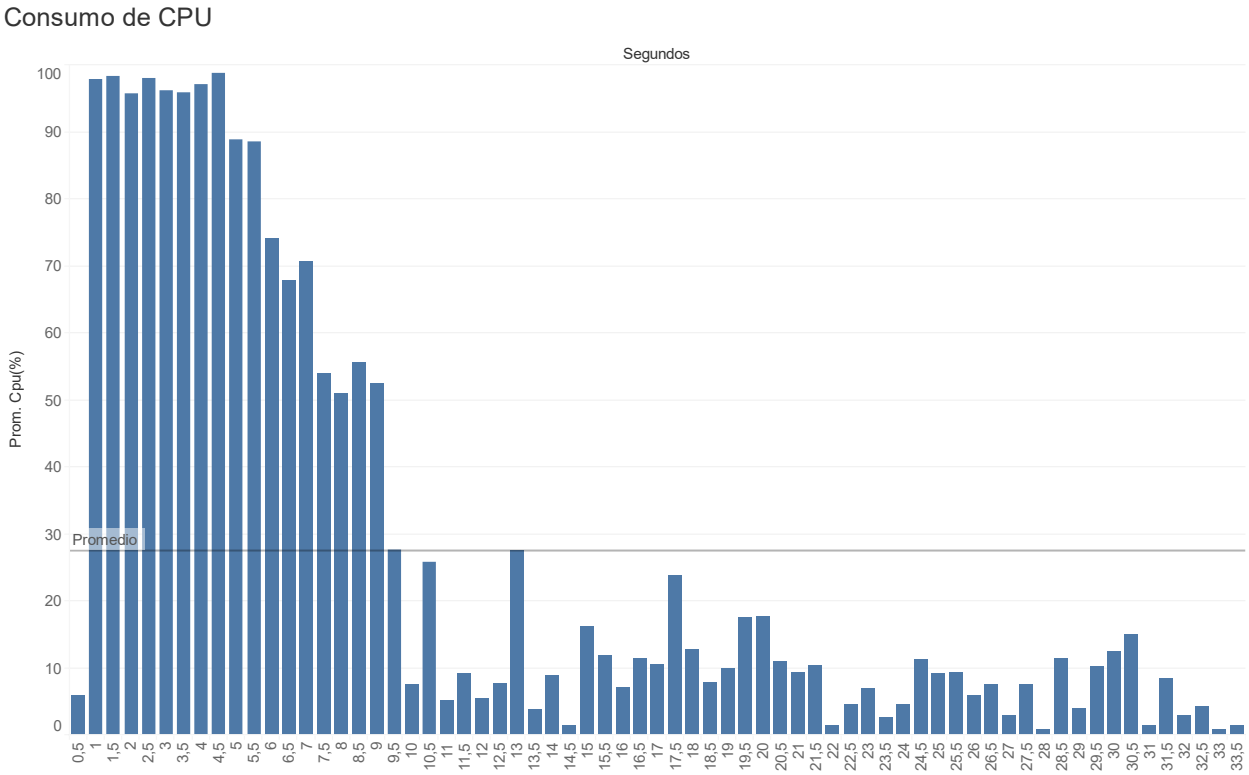


Figura 17: n249f50 CPU

Pero llegados a este punto sí que alcanzamos un punto crítico en el sistema. Con una tasa de 4980 d/s el sistema se pasa 4,5 segundos por encima del 95% y vemos que posteriormente la recuperación es mucho más lenta de lo que venía siendo en las anteriores pruebas.

3.4.3 Consumo de CPU con frecuencia 25 ms

La tercera, y última frecuencia probada, es un dato cada 25 ms. Lo que se traduce a que cada dispositivo manda 40 d/s.

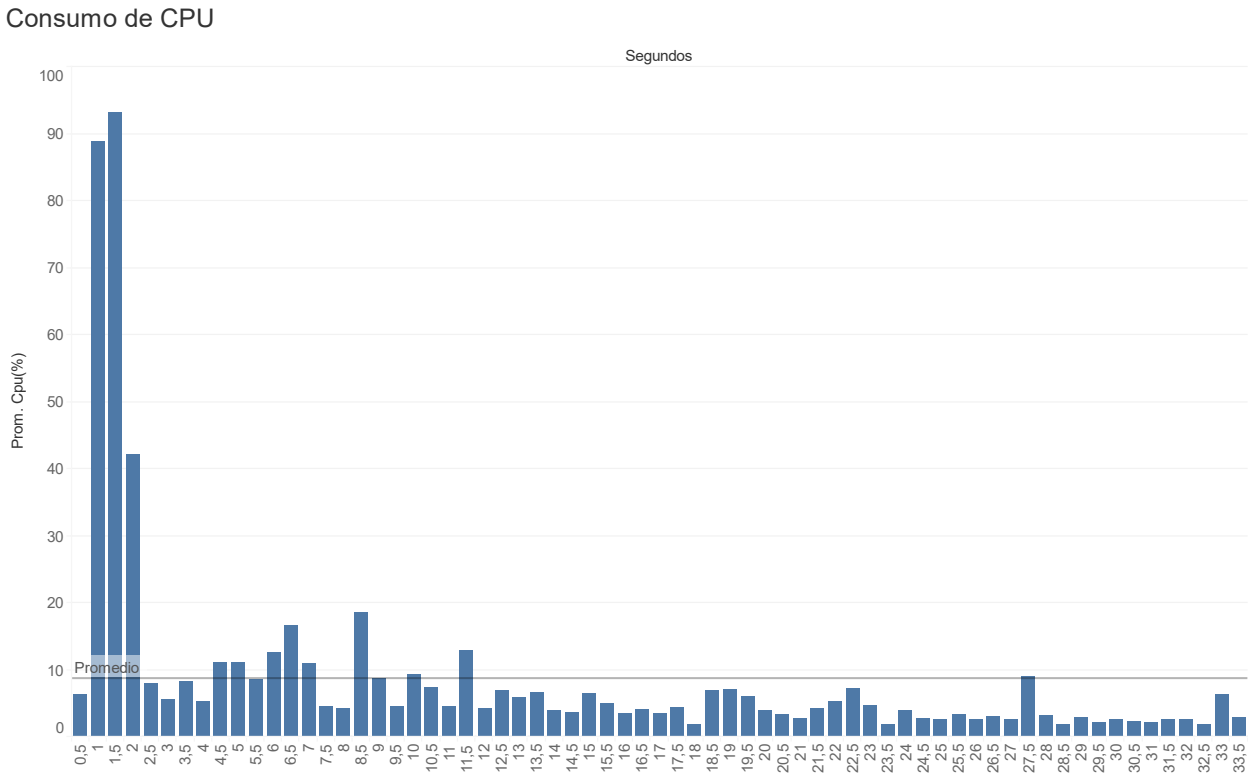


Figura 18: n50f25 CPU

Vemos en la Figura 18 un comienzo mucho más agresivo, ya con 50 dispositivos vemos unos máximos bastante altos ya, lo que concuerda con las gráficas anteriores ya que en este punto son 2000 d/s lo que se está mandando.

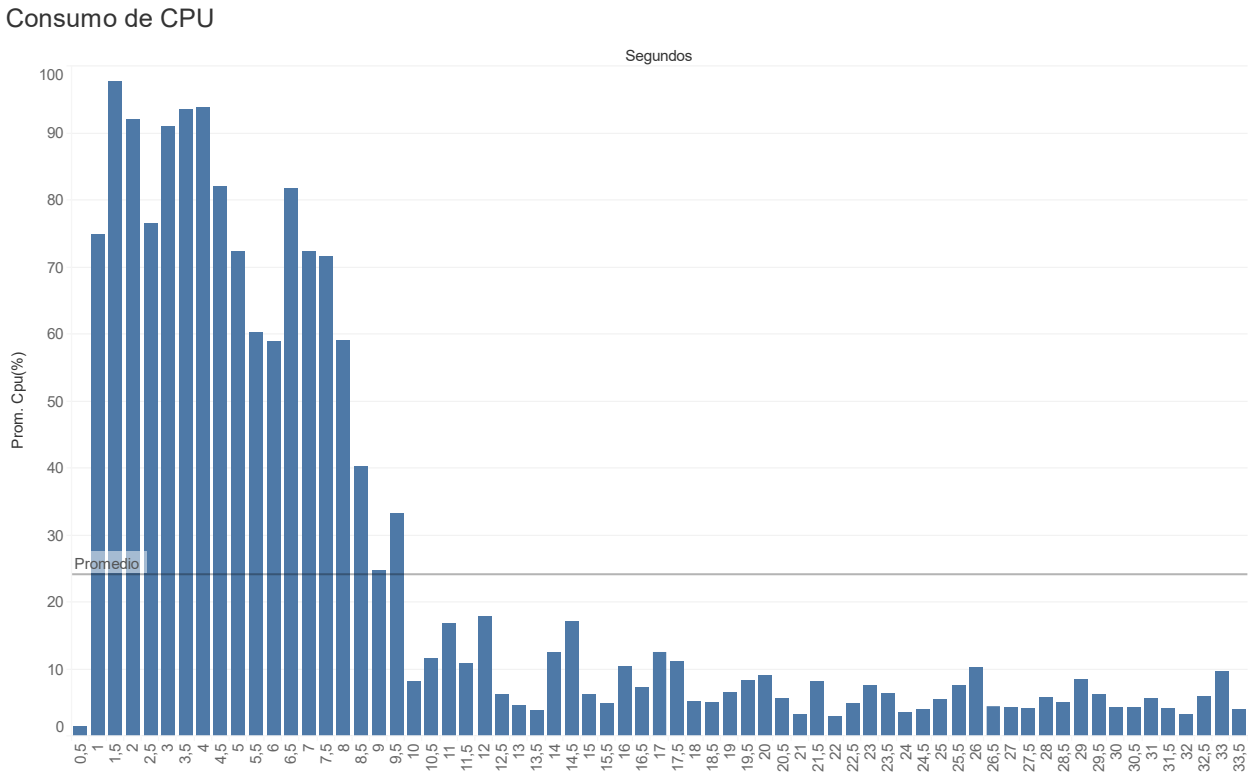


Figura 19: n149f25 CPU

Con 149 dispositivos podemos decir sin problema que se ha llegado prácticamente al límite de la base de datos, con 5960 d/s se observa que el sistema exige una gran cantidad de CPU durante los primeros 7,5 segundos. Que hemos llegado al límite se confirma con la Figura 20, en la que vemos un primer pico que pasa posteriormente a casi no consumir apenas CPU durante unos 9,5 segundos para después llegar a un consumo de CPU

inadmisible. Se puede observar cómo la base de datos satura, y, por lo que se entiende debió reiniciarse o parar de recibir esos datos mandados.

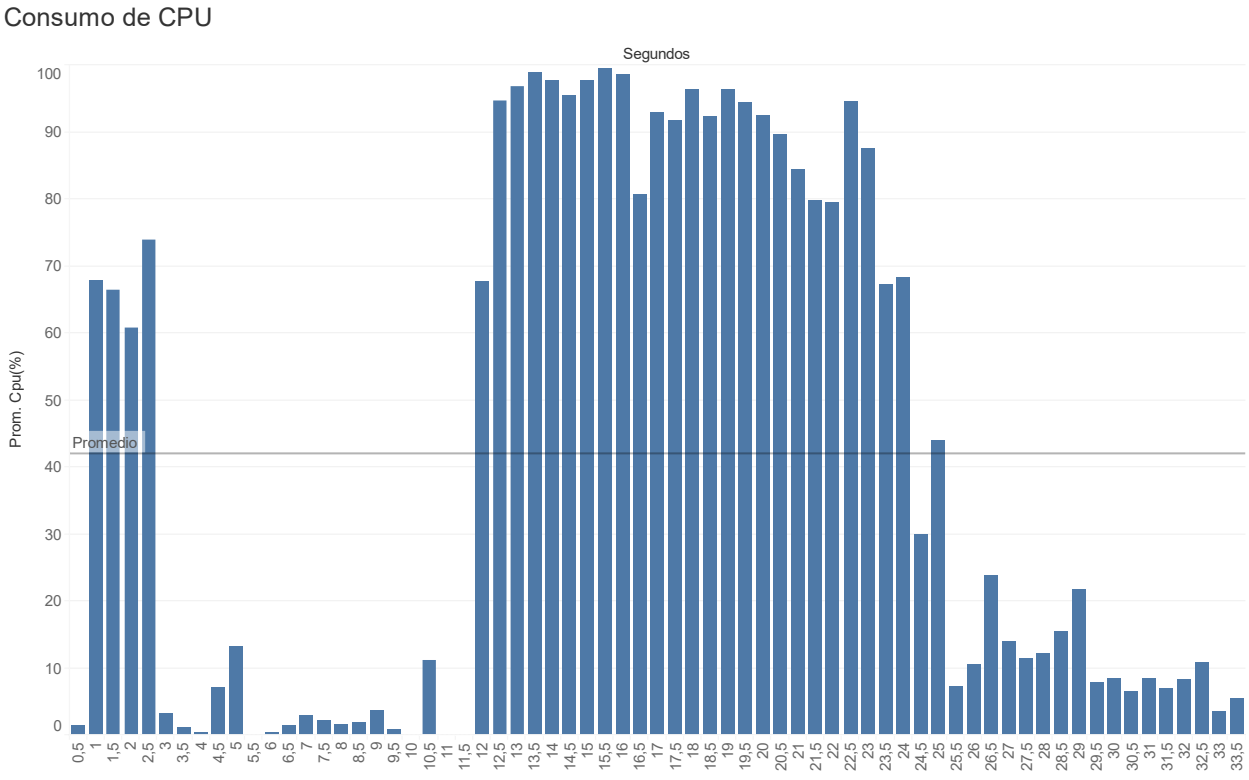


Figura 20: n200f25 CPU

Durante la realización de las pruebas, nos surgió la duda de si el alto consumo de CPU inicial podría deberse al arranque de los hilos para el envío de datos, por lo que se hicimos una prueba para comprobarlo. Esta prueba consiste en 150 dispositivos mandando un dato cada 3 segundos, por lo que si el arranque de los hilos consumiera más de lo normal se debería ver reflejado en un alto consumo en los primeros instantes de la gráfica. En la Figura 21 se ve cómo el consumo en los instantes iniciales no pasa del 10%, por lo que se puede deducir que el arranque de hilos no está causando este consumo inicial. No es hasta el segundo 2,5 y 3 que llega el primer pico significante (coincidiendo con el tiempo que se está mandando los datos).

Esta gráfica confirma la teoría de que la base de datos se adapta con el tiempo a los datos mandados, ya que las primeras inserciones de datos consumen bastante CPU, pero cada vez que se manda el sistema se adapta y logra minimizar los recursos para procesar los datos.

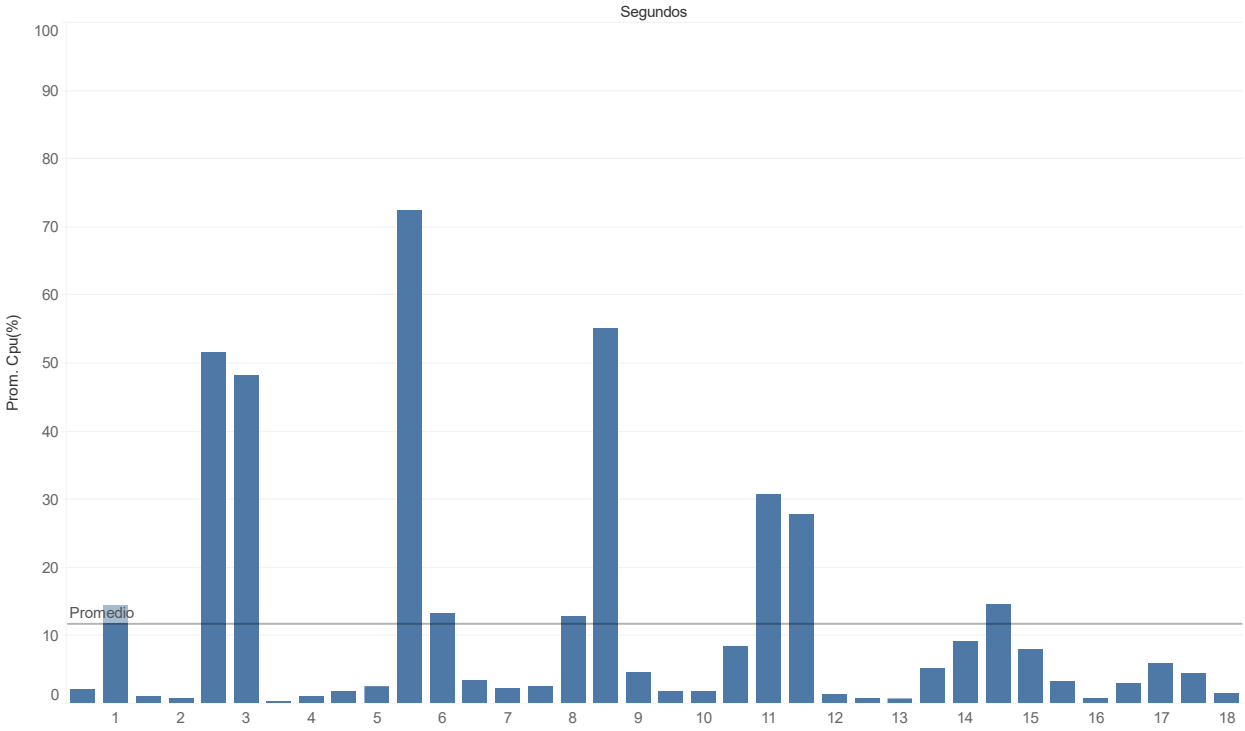


Figura 21: Prueba inicio de hilos.

3.4.4 Resumen sobre los resultados del consumo de CPU

Las pruebas realizadas revelan de manera clara las capacidades y limitaciones del sistema Neo4j bajo diferentes cargas de trabajo, representadas por la variación en la frecuencia de envío de datos y el número de dispositivos conectados. A lo largo de la experimentación, se observó que el sistema podía manejar de manera eficiente un número creciente de dispositivos sin un impacto significativo en el consumo de CPU, hasta que se alcanzaron umbrales críticos que llevaron al sistema a un cuello de botella.

Inicialmente, con una frecuencia de 100 ms por dato, el sistema manejó adecuadamente hasta 300 dispositivos, alcanzando picos de consumo de CPU por encima del 80%, pero aún sin llegar a saturarse por completo. Sin embargo, cuando la frecuencia aumentó a 50 ms y 25 ms por dato, se empezaron a manifestar signos claros de sobrecarga. Con 249 dispositivos y una tasa de 4980 datos por segundo, el sistema alcanzó un punto crítico, manteniendo un consumo de CPU por encima del 95% durante varios segundos y mostrando una recuperación mucho más lenta que en pruebas anteriores.

El experimento más agresivo, con una frecuencia de 25 ms y hasta 149 dispositivos, confirmó que el sistema llegó a su límite. Con una tasa de 5960 datos por segundo, se observó un consumo de CPU extremadamente alto, acompañado de un comportamiento errático en el que el sistema, después de un pico inicial, casi deja de consumir CPU, sugiriendo un reinicio o una interrupción en el procesamiento de datos.

Hemos de aclarar que estas pruebas son solo del consumo de CPU mientras se envían los datos, no se han tenido en cuenta todavía tiempos de consultas ni navegación por la base de datos con diferentes cantidades de datos. Pero estas gráficas nos dan una idea de los datos que Neo4J puede aceptar, más adelante veremos cómo podemos consultar estos datos y si merece la pena recopilar tantos datos o la base de datos se ralentizará de manera que impida un uso normal.

3.4.2 Consumo de RAM

Dado que el consumo de RAM es tan alto, no conviene ver las gráficas igual que con la CPU, ya que no se apreciarían cambios. Para las siguientes gráficas se ha limitado el eje vertical (que indica el consumo de RAM) a un mínimo de 70 y un máximo de 100, ya que los cambios son leves y este cambio permite que se aprecien mejor. Cabe destacar también que se ha cambiado la medida en el eje horizontal, indicando esta vez el número de dispositivos en vez del tiempo. De cada número de dispositivos se ha extraído el uso de RAM medio.

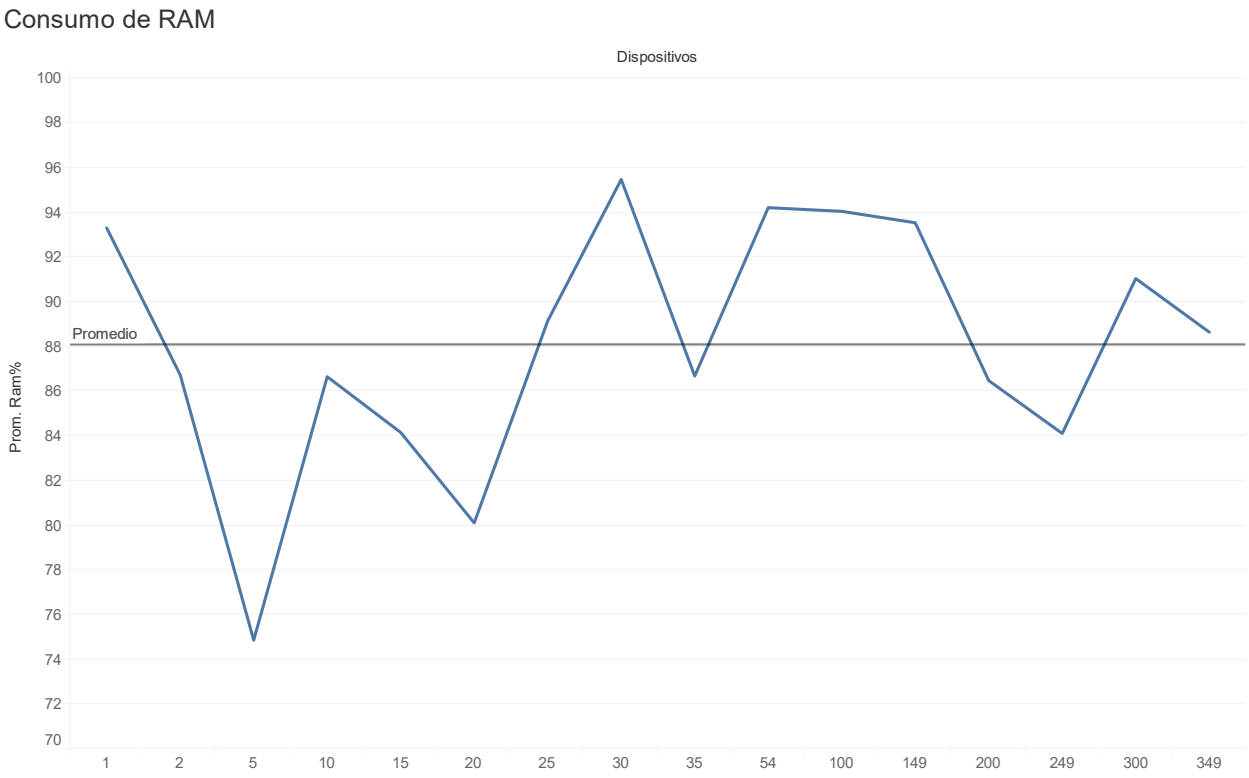


Figura 22: f100 RAM

Empezando de nuevo con la frecuencia a 100 ms, la Figura 22 revela que el consumo de RAM aumenta de manera irregular a medida que se incrementa el número de dispositivos, con algunos picos significativos, llegando a un máximo de 96% con 30 dispositivos, y posteriores estabilizaciones. Estos resultados sugieren que el sistema está manejando la carga de memoria de manera eficiente en la mayoría de los casos, aunque existen ciertos puntos donde la demanda de RAM es considerablemente mayor, debido a momentos de alta concurrencia o procesamiento intensivo de datos.

La Figura 23, con una frecuencia ahora de 50 ms, muestra un comportamiento más estable del consumo de RAM en comparación con la anterior, con el uso de RAM manteniéndose mayormente en torno al 88%, pero aun presentando algunas fluctuaciones. Inicialmente, el consumo de RAM comienza ligeramente por encima del promedio (88-89%) con un menor número de dispositivos (10 a 49) y luego desciende a medida que aumenta el número de dispositivos.

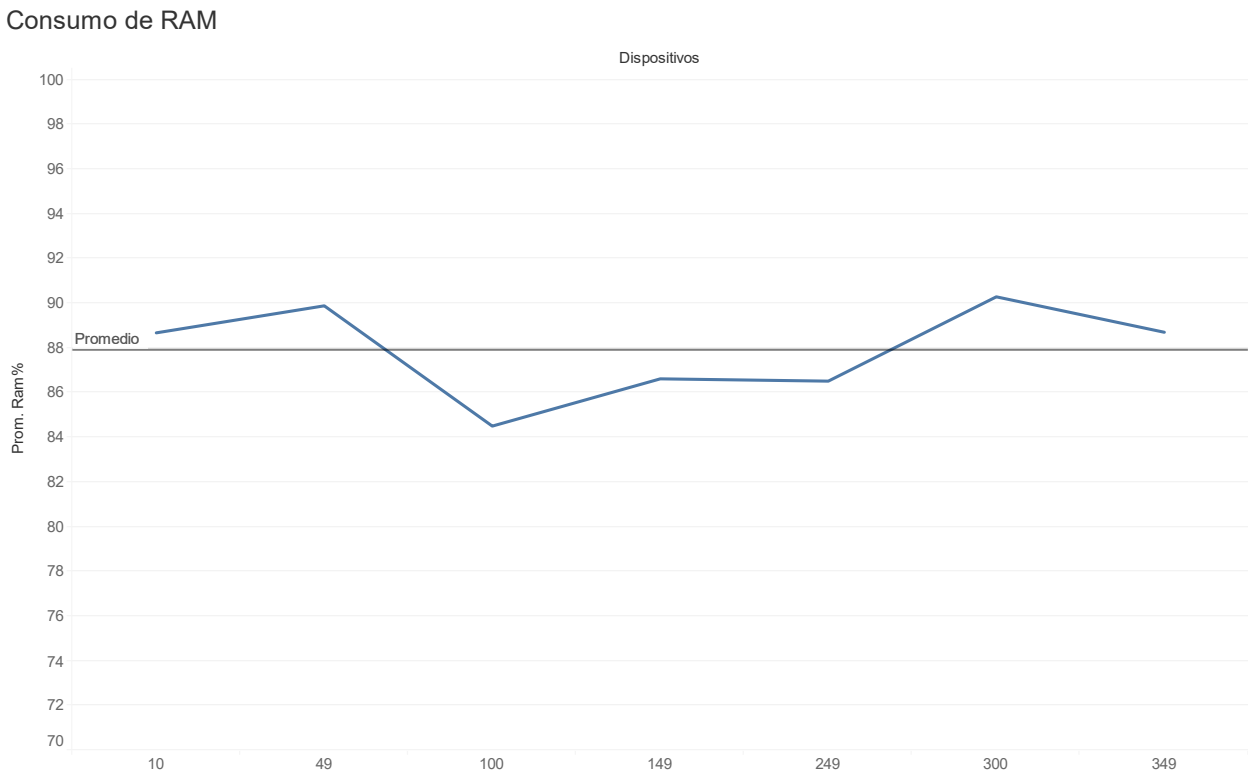


Figura 23: f50 RAM

A medida que se incrementan los dispositivos hasta 149, el consumo de RAM alcanza su punto más bajo alrededor del 84% y se mantiene estable en el rango de 84%-86% hasta llegar a 249 dispositivos. Con 300 dispositivos, el uso de RAM vuelve a aumentar, acercándose al 89%, antes de disminuir ligeramente hacia los 349 dispositivos.

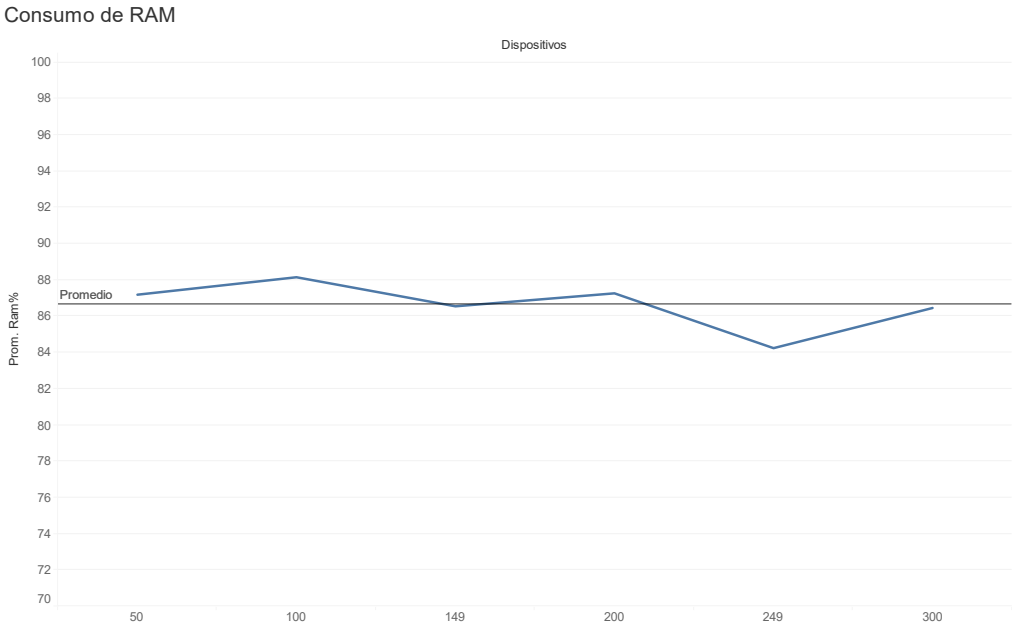


Figura 24: f25 RAM

La Figura 24 para una frecuencia de 25 ms muestra fluctuaciones en el consumo de RAM con un aumento irregular a medida que se incrementa el número de dispositivos. Inicialmente, se observa un descenso en el consumo de RAM con pocos dispositivos, seguido de un aumento y posterior disminución. El consumo de RAM

alcanza su punto más alto (96%) con 30 dispositivos. El uso de RAM se estabiliza alrededor del 80-88%.

Según lo visto con el uso de RAM, parece que el sistema maneja mejor las cargas constantes y rápidas, provocando menos fluctuación en el uso de la memoria.

3.4.3 Resultados combinados

Adicionalmente a lo expuesto en los apartados anteriores, es interesante ver el cambio de CPU y RAM en una misma gráfica para una frecuencia dada. Para la Figura 25 se ha filtrado la frecuencia a 25 milisegundos, ya que es el experimento más crítico, sobre todo para la CPU. La línea naranja es el uso de RAM y la línea azul el uso de CPU (véase leyenda).

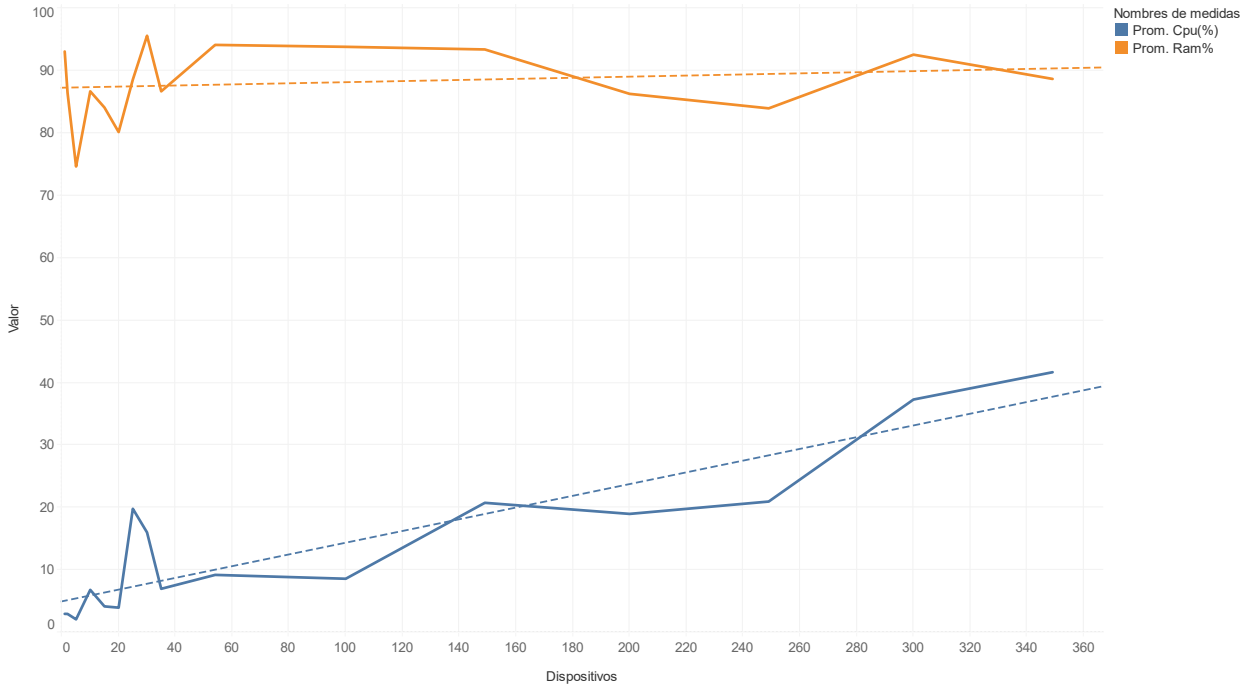


Figura 25: CPU vs RAM f100

Podemos observar que el uso de la RAM se mantiene relativamente estable, con un promedio cercano al 90% independientemente del número de dispositivos (como ya se vio previamente).

El comportamiento de la CPU muestra una tendencia ascendente que refleja la creciente carga de procesamiento a medida que aumenta el número de dispositivos. En los primeros 100 dispositivos, el uso de CPU se mantiene relativamente bajo, pero con algunas fluctuaciones iniciales. Esto podría deberse a la variabilidad en la gestión de las primeras conexiones o al manejo de picos de datos cuando se incrementa el número de dispositivos.

A partir de aproximadamente 200 dispositivos, el uso de CPU empieza a aumentar de manera más constante y pronunciada. Este crecimiento indica que la CPU está realizando más cálculos y manejando un mayor volumen de datos a medida que se conectan más dispositivos. La tendencia ascendente continua sugiere que la CPU se va acercando a su capacidad máxima de procesamiento, lo que podría eventualmente llevar a un cuello de botella si el número de dispositivos sigue aumentando sin un incremento en la capacidad de procesamiento o una optimización en la gestión de datos. Esto resalta la importancia de monitorear y gestionar eficientemente el uso de CPU en sistemas que dependen de la conexión y procesamiento de datos en tiempo real de múltiples dispositivos.

Se aprecia también en esta gráfica cómo los picos de CPU y RAM suben y bajan paralelamente. Se aprecia bien desde los 150 dispositivos hasta los 300, cómo hay un valle en ambas gráficas, y los picos iniciales que poseen la misma forma.

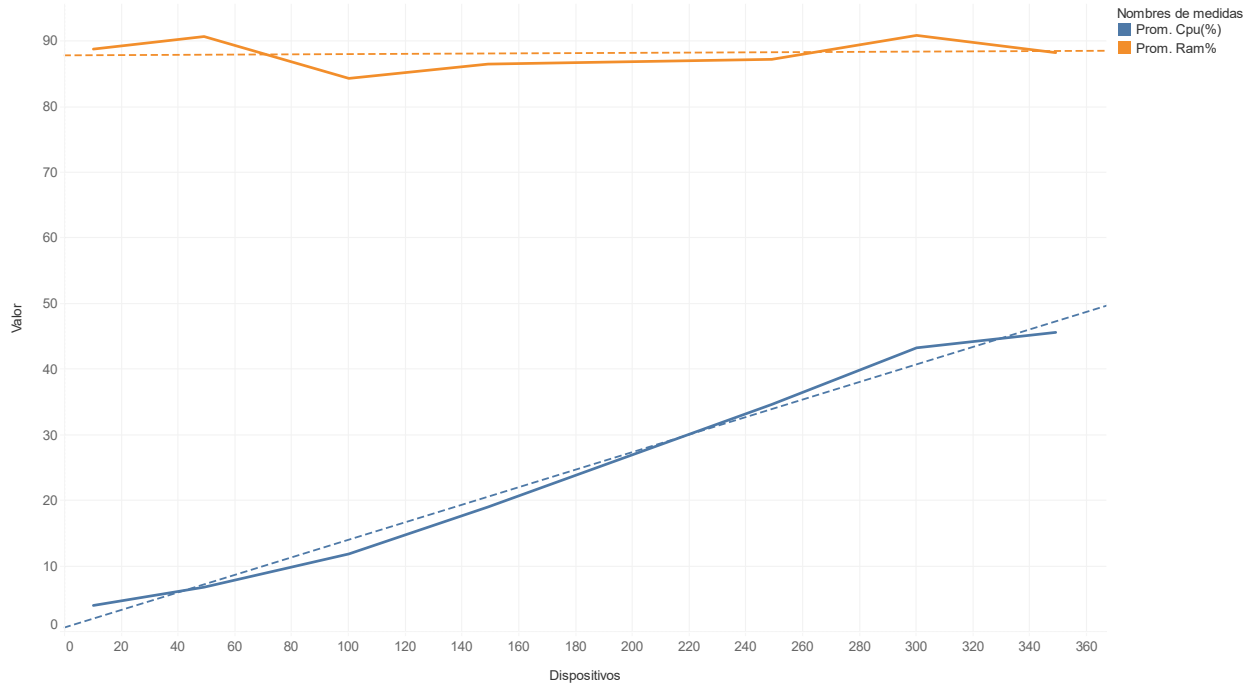


Figura 26: CPU vs RAM f50

En la Figura 26, para una frecuencia de 50 ms, se observa que la línea azul, que representa el uso promedio de CPU, muestra un incremento continuo y más suave en comparación con la gráfica anterior. Este comportamiento indica que, a medida que aumenta el número de dispositivos, la carga de procesamiento en la CPU sigue creciendo de manera lineal. El patrón refleja un aumento consistente en la demanda de procesamiento, probablemente debido al procesamiento de los datos que cada dispositivo está enviando en intervalos cortos.

Es importante notar que, aunque el incremento es lineal, la tendencia apunta a una carga significativa a medida que se alcanzan los 360 dispositivos, llegando a un uso de CPU alrededor del 40%. Este incremento sugiere que, si se continúa aumentando el número de dispositivos, el sistema podría acercarse a un punto donde la capacidad de la CPU podría volverse un factor limitante, requiriendo posibles optimizaciones o mejoras en la capacidad de procesamiento para evitar problemas de rendimiento.

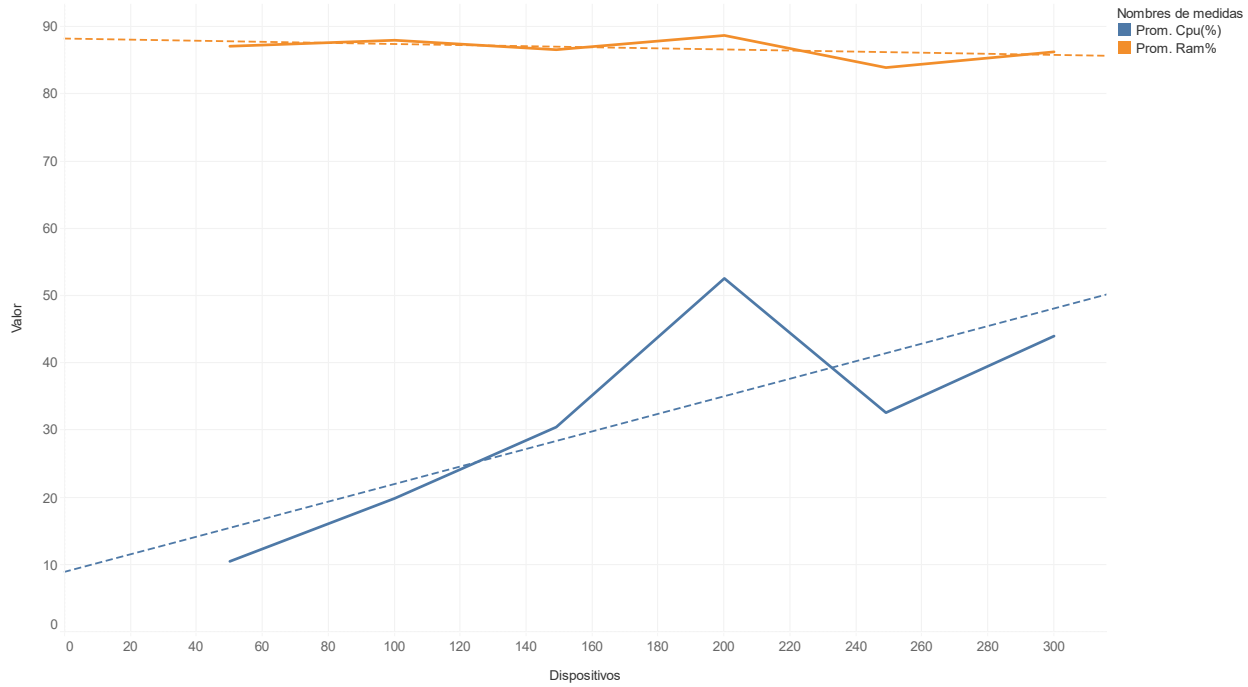


Figura 27: CPU vs RAM f25

En la Figura 27, la CPU muestra un comportamiento más irregular en comparación con las gráficas anteriores. Si bien existe una tendencia general ascendente en el uso de la CPU a medida que aumenta el número de dispositivos, también se observan picos y caídas significativas, especialmente alrededor de los 180 dispositivos y luego una disminución alrededor de los 220 dispositivos. Este patrón, se debe a lo que vimos previamente en el consumo de CPU, que al saturarse la base de datos se detiene y disminuye drásticamente el consumo de CPU.

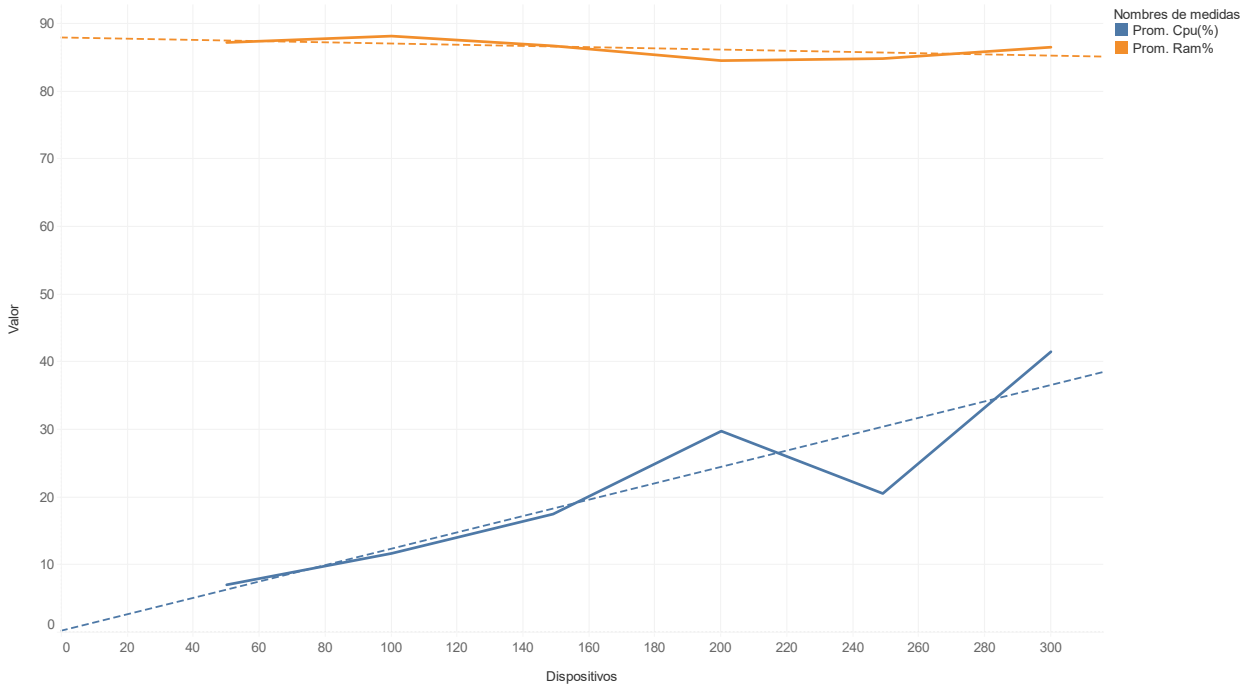


Figura 28: CPU vs RAM f25_v2

Lo que se comentaba previamente se puede apreciar en la gráfica de la Figura 28, la cual es la misma que la Figura 27, pero aumentando los segundos de medición (pasando de 25 segundos a 50). Aquí se puede ver más clara la tendencia, y que a medida que se mandan más datos desde más dispositivos el sistema tiende a saturar. Pero los picos y caídas indican que el sistema podría estar experimentando momentos de alta demanda seguidos de momentos de alivio. Esto podría ser resultado de cómo se gestionan las tareas en la CPU, quizás debido a un algoritmo de distribución de carga que no distribuye de manera uniforme en todos los casos.

Podemos ver también los resultados en una tabla (Figura 29). La tabla refleja la media de CPU y RAM a medida que aumenta el número de dispositivos. Hay que tener en cuenta que con un mismo número de dispositivos se han hecho pruebas a distintas frecuencias y esta tabla recoge la media de todas las medidas.

En la primera columna se refleja el número de dispositivos, la segunda la media de CPU y la tercera la media de RAM.

Dispositiv..	Prom. Cpu(%)	Prom. Ram%
0	0,78	90,71
1	2,13	92,68
2	1,50	86,83
5	1,29	76,60
10	3,20	86,22
15	2,87	83,08
20	2,67	80,48
25	9,26	90,41
30	7,61	94,62
35	3,90	89,67
49	3,66	88,84
50	4,26	87,60
54	5,53	95,08
100	6,06	89,61
149	10,03	89,27
200	12,10	83,66
249	11,76	85,42
300	16,16	87,92
349	15,87	87,74

Figura 29: Tabla de medias CPU y RAM

Como hemos visto durante todo el análisis, el promedio de CPU aumenta poco a poco de manera lógica, mientras que el promedio de RAM va variando sin un criterio aparente. Para visualizar eso mejor, podemos ayudarnos de las dos gráficas siguientes.

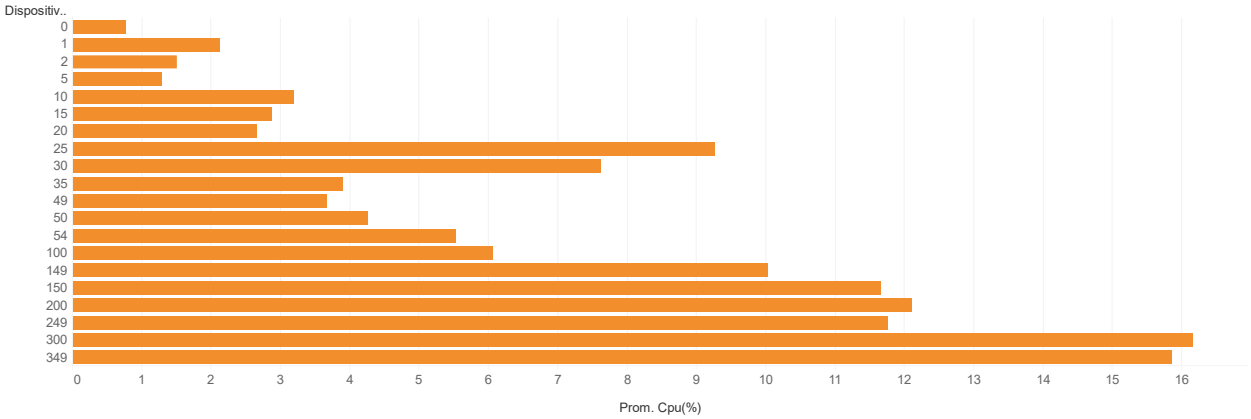


Figura 30: Media de CPU vs dispositivos

Como hemos comentado a partir de la tabla, la media de CPU va aumentado acorde al número de dispositivos, salvo con 25 y 30 dispositivos. Esto puede deberse a que en esos casos haya más muestras de mayor frecuencia.

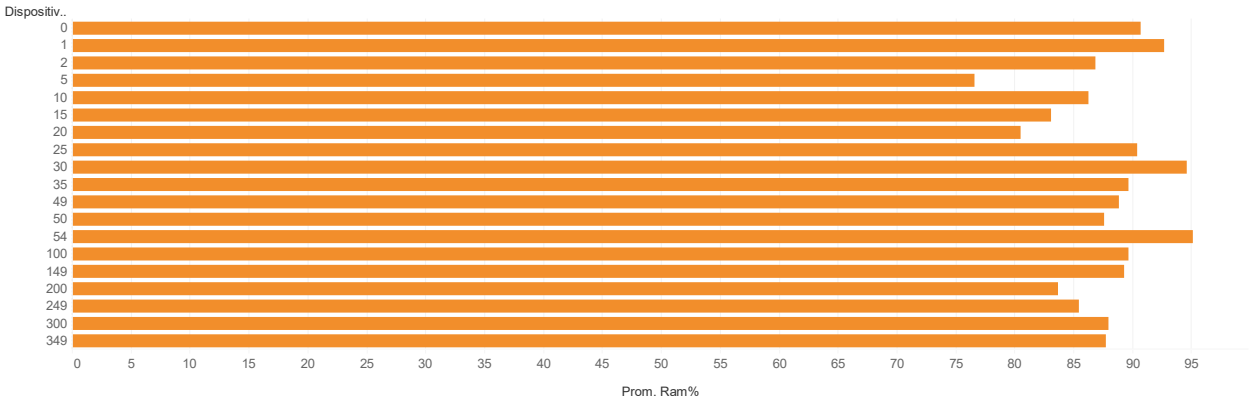


Figura 31: Media de RAM vs dispositivos.

En cuanto a la media de RAM se ve de nuevo mucha aleatoriedad, siempre un consumo alto pero sin un crecimiento acorde al número de dispositivos.

3.4.4 Conclusiones sobre los experimentos de RAM y CPU

Las pruebas realizadas han revelado de manera clara y detallada las capacidades y limitaciones del sistema Neo4j cuando se enfrenta a diferentes cargas de trabajo, caracterizadas por variaciones en la frecuencia de envío de datos y el número de dispositivos conectados. A continuación, se resumen las observaciones y conclusiones más relevantes de este análisis.

Frecuencia de 100 ms por dato:

- Capacidad Inicial del Sistema: Al principio, el sistema maneja adecuadamente hasta 300 dispositivos con una frecuencia de 100 ms por dato. Durante estas pruebas, se observan picos de consumo de CPU por encima del 80%, pero el sistema no llega a saturarse.
- Primeros Signos de Sobrecarga: Se identifican picos aislados, como con 25 dispositivos, donde se alcanza un 95,29% de uso de CPU, pero estos picos son momentáneos y no indican una saturación sostenida.
- Conclusión: A esta frecuencia, el sistema muestra una robustez considerable, gestionando eficientemente hasta 300 dispositivos con un volumen de 3000 datos por segundo.

Frecuencia de 50 ms por dato:

- Incremento de la Demanda: Al reducir la frecuencia a 50 ms, el volumen de datos se incrementa, alcanzando hasta 4980 datos por segundo con 249 dispositivos.
- Punto Crítico Identificado: A este nivel de carga, el sistema muestra signos claros de estrés, con la CPU manteniéndose por encima del 95% durante varios segundos y una recuperación más lenta.
- Conclusión: Aunque el sistema aún maneja la carga, se aproxima a su límite operativo, sugiriendo que una mayor optimización o un incremento en la capacidad de procesamiento podría ser necesario para evitar problemas de rendimiento.

Frecuencia de 25 ms por dato:

- Sobrecarga del Sistema: Esta prueba, la más agresiva, revela que el sistema se enfrenta a su límite máximo con 149 dispositivos enviando datos a 25 ms. El consumo de CPU se eleva a niveles insostenibles, y se observa un comportamiento errático, con períodos de casi inactividad que sugieren un reinicio o interrupción en el procesamiento.
- Estabilidad de la RAM: A pesar de las variaciones en la carga de la CPU, el consumo de RAM se mantiene relativamente estable en todas las pruebas, lo que indica que la memoria no es el principal factor limitante en este escenario.
- Conclusión: A frecuencias tan elevadas como 25 ms por dato, el sistema alcanza rápidamente un cuello

de botella, mostrando que la capacidad de procesamiento es insuficiente para manejar cargas tan altas de manera sostenida.

Comparación General:

- Uso de CPU vs RAM: Se ha observado que, mientras la CPU presenta un comportamiento más variable y susceptible a la carga de trabajo, la RAM mantiene un consumo alto pero estable. Esto sugiere que, en estas condiciones de prueba, la CPU es el recurso que primero alcanza su límite, actuando como el principal cuello de botella del sistema.
- Optimización Necesaria: Dado que el sistema muestra una capacidad de recuperación más lenta a medida que aumenta la carga, sería necesario considerar optimizaciones en la gestión de tareas, la distribución de carga o incluso mejoras en el hardware si se pretende escalar el número de dispositivos conectados o la frecuencia de envío de datos.

3.5 Análisis del tiempo de respuesta

Para analizar los tiempos de respuesta, se realizó un programa en Java que permitió automatizar las consultas y a su vez añadir más datos a la base de datos tras una consulta (ayudándonos del código previamente creado PruebaDemo.java).

El nuevo código es TestTime.java. La función principal y la que más nos interesa para estas pruebas es 'executeQuery':

```
public long executeQuery(String query) {
    try (Session session = driver.session()) {
        long startTime = System.currentTimeMillis();
        session.run(query);
        long endTime = System.currentTimeMillis();
        return endTime - startTime;  // Devuelve el tiempo en milisegundos
    }
}
```

Figura 32: Función de ejecución de consultas

Esta función ejecuta la consulta proporcionada como parámetro, midiendo el tiempo antes y después de ejecutarla.

Las consultas que se han probado han sido las siguientes:

Nº de la consulta	Consulta	Descripción
1	MATCH c = shortestPath((p:Person:Paciente {dni: 'dniEjemplo100'})-[:ASIGNADO_A*..]->(m:Person:Medico)) RETURN length(c) AS pathLength, nodes(c) AS pathNodes ORDER BY pathLength DESC LIMIT 10	Esta consulta fue diseñada en un primer momento pero descartada posteriormente. Se mantiene aquí para no variar la numeración del resto.
2	MATCH (p:Person:Paciente) RETURN p.dni AS pacienteDNI, d.modelo AS dispositivoModelo, count(o) AS cantidadObservaciones ORDER BY cantidadObservaciones DESC	Recupera el DNI de cada paciente, el modelo del dispositivo que usa y la cantidad de observaciones registradas para ese paciente. Luego, ordena los resultados en orden

		descendente según la cantidad de observaciones, mostrando primero a los pacientes con más observaciones.
3	MATCH(m:Person:Medico)<-[:ASIGNADO_A]-(p:Person:Paciente) WITH m, count(p) AS numeroDePacientes ORDER BY numeroDePacientes DESC LIMIT 5 RETURN m.nombre AS medicoNombre, numeroDePacientes	Identifica a los cinco médicos con más pacientes asignados. Cuenta la cantidad de pacientes asignados a cada médico, ordena los resultados en orden descendente y muestra el nombre del médico junto con el número de pacientes que tiene.
4	MATCH(o:Observacion)-[r:PERTENECE_A]->(p:Paciente) WHERE toInteger(o.dato) > 560 RETURN p	Devuelve los pacientes cuyas observaciones superen un cierto umbral
5	MATCH (o:Observacion) WHERE datetime(o.timeStamp) > datetime('2024-10-27T11:35:21') RETURN o	Devuelve todos los datos recopilados después de cierta fecha
6	MATCH (n) RETURN n	Devuelve todos los nodos
7	Igual a consulta 4	
8	Igual a consulta 5	
9	Igual a consulta 6	
10	MATCH (a)-[r:PERTENECE_A]->(b) WITH r LIMIT 200000 DELETE r	Elimina 200.000 relaciones PERTENECE_A
11	MATCH (c)-[r2:REALIZADA_POR]->(d) " WITH r2 LIMIT 200000 DELETE r2	Elimina 200.000 relaciones REALIZADA_POR
12	MATCH (a2:Observacion) WITH a2 LIMIT 200000 DELETE a2	Elimina 200.000 nodos Observación
13	Igual a consulta 4	
14	Igual a consulta 5	

15	Igual a consulta 6	
16	Igual a consulta 4	
17	Igual a consulta 5	
18	Igual a consulta 6	
19	MATCH (p1:Person:Paciente)<-[:PERTENECE_A]-(o1:Observacion), (p2:Person:Paciente)<-[:PERTENECE_A]-(o2:Observacion) WHERE o1.dato=o2.dato RETURN p1.dni AS Dni1, p2.dni AS Dni2	Devuelve el dni de los Pacientes que tengan nodos Observación con el mismo valor en el campo dato

Tabla 11: Consultas utilizadas

Es de notar cómo se han repetido las consultas 4, 5 y 6 varias veces. Esto se ha hecho porque se han probado varios métodos a la hora de realizar las pruebas.

3.5.1 Método 1

El primer método que se usará para analizar los tiempos de respuesta es el siguiente:

- Se ejecutan las consultas desde el proyecto en eclipse. Cada consulta es medida en términos de tiempo de ejecución (ms), capturando los resultados para su posterior análisis.
- Justo después, se ejecuta pruebaDemo, para generar nodos y relaciones en la base de datos.
- Ahora, con más nodos y relaciones que en la iteración anterior, vuelta a empezar.

Los resultados de las pruebas 2 y 3 fueron:

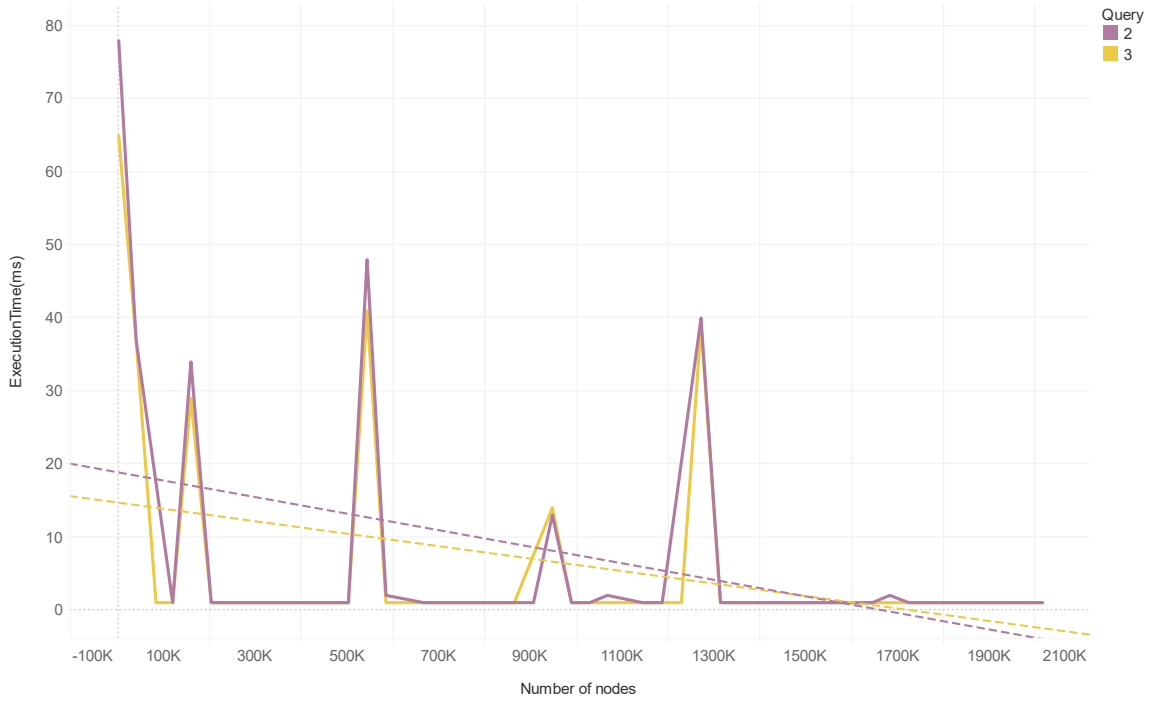


Figura 33: Tiempos de respuesta consultas 2 y 3

En la Figura 33, en el eje Y tenemos el tiempo de ejecución en ms y en el eje X el número de nodos. En la esquina superior derecha tenemos la leyenda que indican el color de las consultas. Las próximas gráficas tendrán la misma estructura que esta, salvo que se indique lo contrario.

Retomando de nuevo los resultados, las consultas 2 y 3 llegan a su máximo en la primera consulta, con apenas 778 nodos. El máximo es de 78ms para la consulta 2 y de 65 para la consulta 2.

A medida que el número de nodos aumenta, los tiempos de ejecución se sitúan mayormente en 0, 1 y 2 ms, salvo en algunos picos puntuales que sí que superan los 10 ms en la consulta. Esto motivó a hacer pruebas con otro tipo de consultas y llegando hasta un mayor número de nodos, para ver si se mantenía esta tendencia.

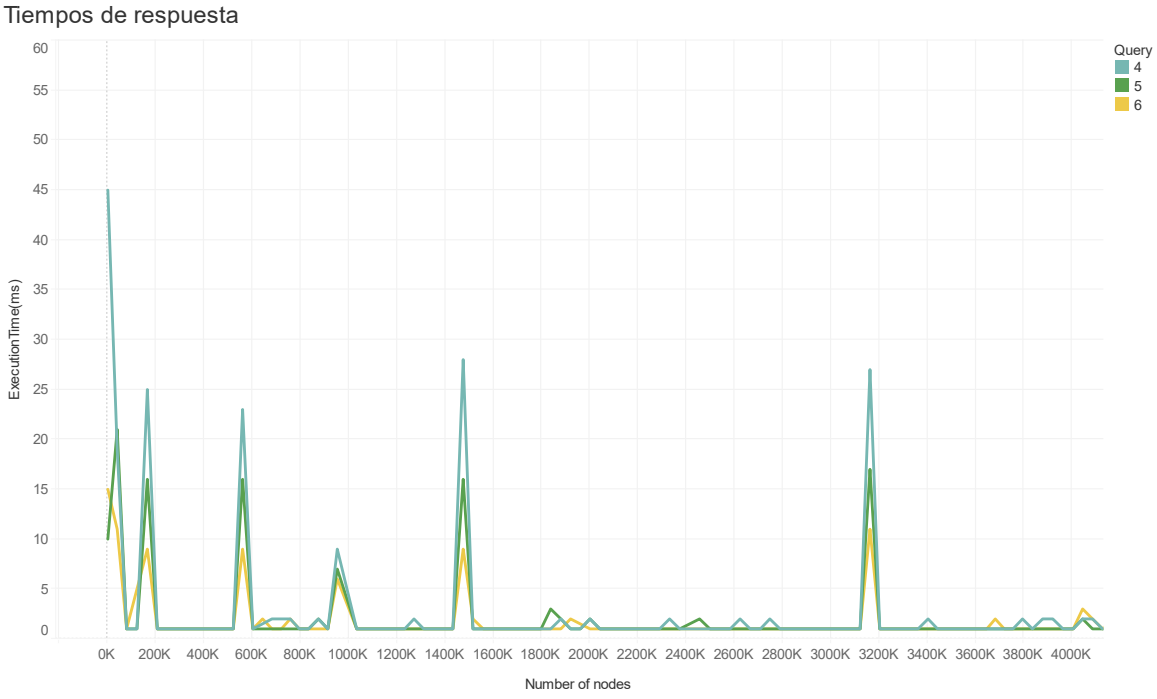


Figura 34: Tiempos de respuesta consultas 4,5 y 6

La Figura 34 muestra los resultados de las consultas 4, 5 y 6. A pesar de llegar a un mayor número de nodos, el

patrón se repite, mayormente obtendremos tiempos despreciables como resultado de la ejecución de las consultas. Decidimos resumir los datos en una tabla, para una mejor compresión. El desglose queda de la siguiente manera: la primera columna indica el tipo de consulta y la segunda el número de consultas de ese tipo que se han realizado. Los tiempos de respuesta se agrupan en las columnas siguientes.

Consulta	Total medidas	0 ms	1 ms	2 ms	otros
4	150	9	109	22	10
5	150	22	111	8	9
6	150	31	98	12	9

Tabla 12: Resultados método 1

Podemos ver como las medidas de 0, 1 y 2 ms son las dominantes entre todos los casos, suponiendo el 93% para la consulta número 4, el 94% para las consultas número 5 y 6. Mientras que todas las medidas que toman un tiempo superior raramente se repiten.

Algo que llama la atención a simple vista es que los picos se encuentran todos ubicados en el mismo momento, lo que da que pensar que la memoria caché está jugando un papel en estos resultados. Es por eso por lo que se diseñó y experimentó con un segundo método para la realización de estas pruebas.

3.5.2 Método 2

Estas pruebas siguen el mismo procedimiento a las pruebas anteriores, pero la memoria caché se redujo de 50MB a 10MB, con la intención de ver si el tiempo se veía influenciado con esta reducción.

Pero, como podemos ver en la figura 35 el patrón que sigue es el mismo.

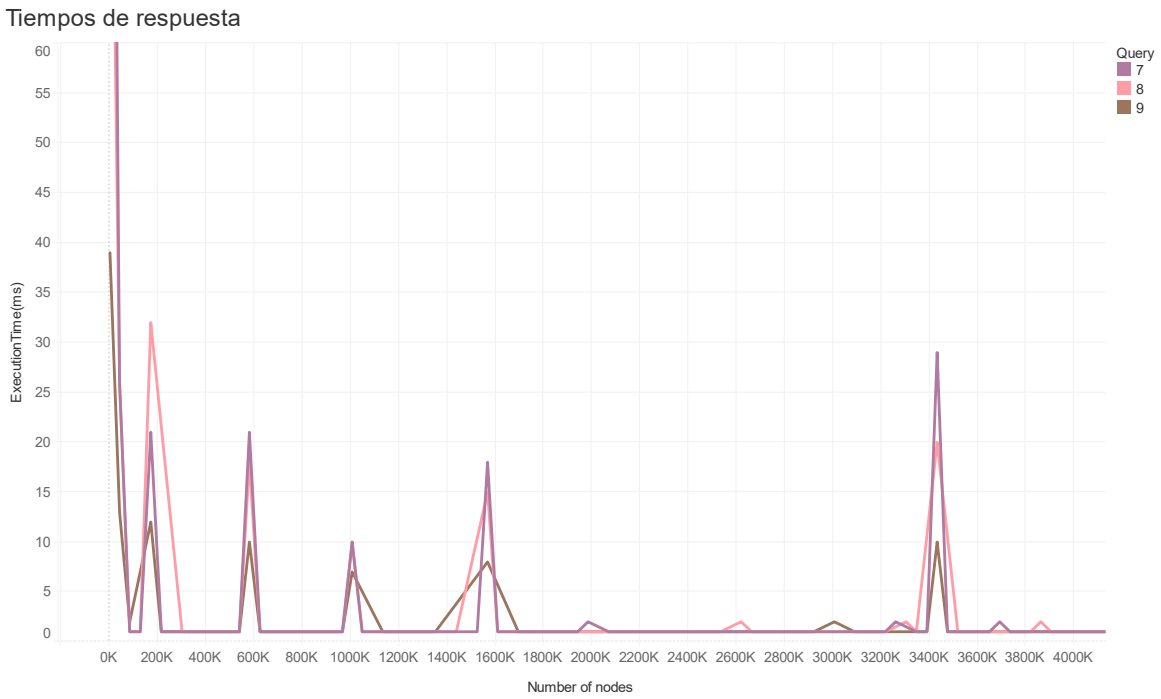


Figura 35: Tiempos de respuesta consultas 7,8 y 9

Podemos comprobar que los picos siguen coincidiendo de unas consultas con otras, y, si vemos la tabla que analizamos el caso anterior el resultado sigue siendo parecido:

Consulta	Total medidas	0 ms	1 ms	2 ms	otros

7	200	57	131	4	8
8	200	69	118	5	8
9	200	78	111	3	8

Tabla 13: Resultados método 2

Siendo el porcentaje de medidas de 0, 1 y 2ms suponen un 96% para las tres consultas. Lo que sigue siendo un resultado que tampoco proporciona mucha información.

Algo que si proporciona algo más de información, es la comparación de las consultas con distinta memoria caché:

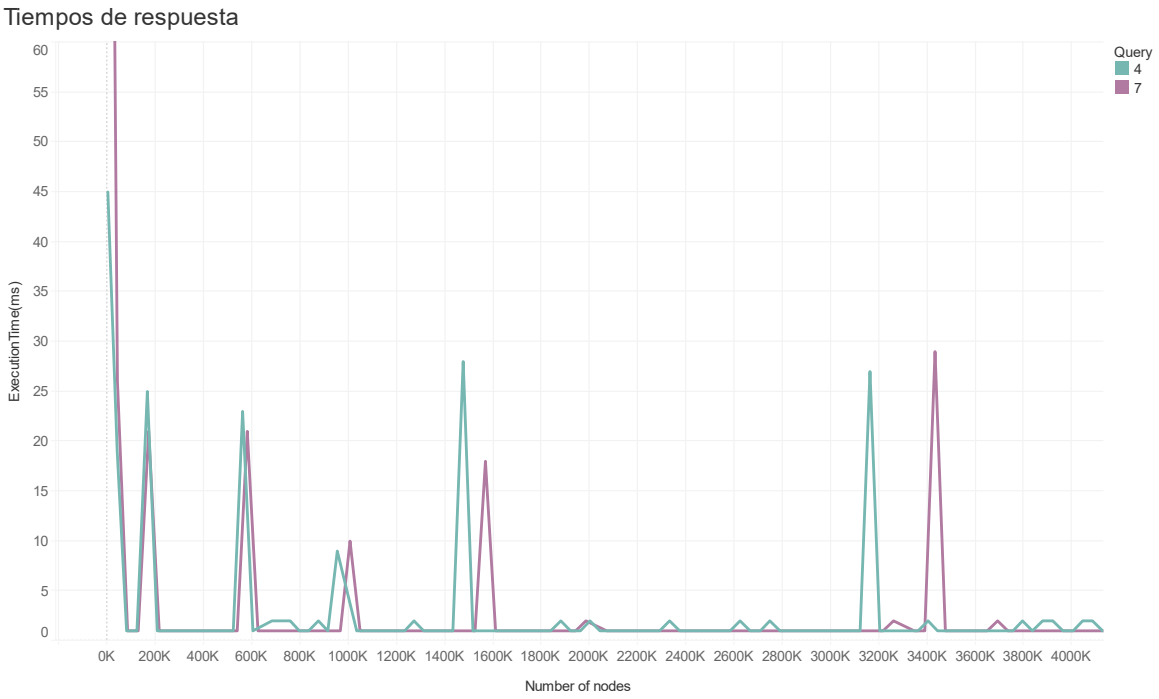


Figura 36: Comparación tiempos de respuesta consultas 4 y 7

Si comparamos las consultas 4 y 7, teniendo que cuenta que son la misma consulta, pero con distinta memoria caché, se observa perfectamente cómo la posición de los picos cambia, pero el tiempo de las consultas sigue siendo bastante parecido. Así que, sabemos que esto sí que se debe a la memoria caché.

La información recopilada, nos llevará a los métodos 3 y 4 en los que intentamos obtener unos resultados que nos permitan un mejor análisis.

3.5.3 Método 3

Tras analizar los resultados anteriores e investigar el lenguaje Cypher, encontramos una sentencia que podría ayudarnos en nuestro caso, para poder analizar el tiempo de cada consulta sin que influya la memoria caché. Este método seguirá el mismo procedimiento que el método 1, con la adición de la sentencia 'CALL db.clearQueryCaches();' tras añadir los nodos y relaciones a la base de datos. Este método se utilizó para las pruebas 13, 14 y 15.

Para la Figura 37 se ha usado en el eje X una escala logarítmica, debido a que se alcanzó un alto número de nodos y esto permite una mejor visualización la gráfica. A pesar de estar usando la sentencia para borrar la caché, los tiempos parece que también están siguiendo un patrón, con muchas subidas y bajadas en los tiempos. Lo que nos da la sensación al ver esta gráfica es que los tiempos a los que sube y a los que baja parece que están rondando los mismos valores.

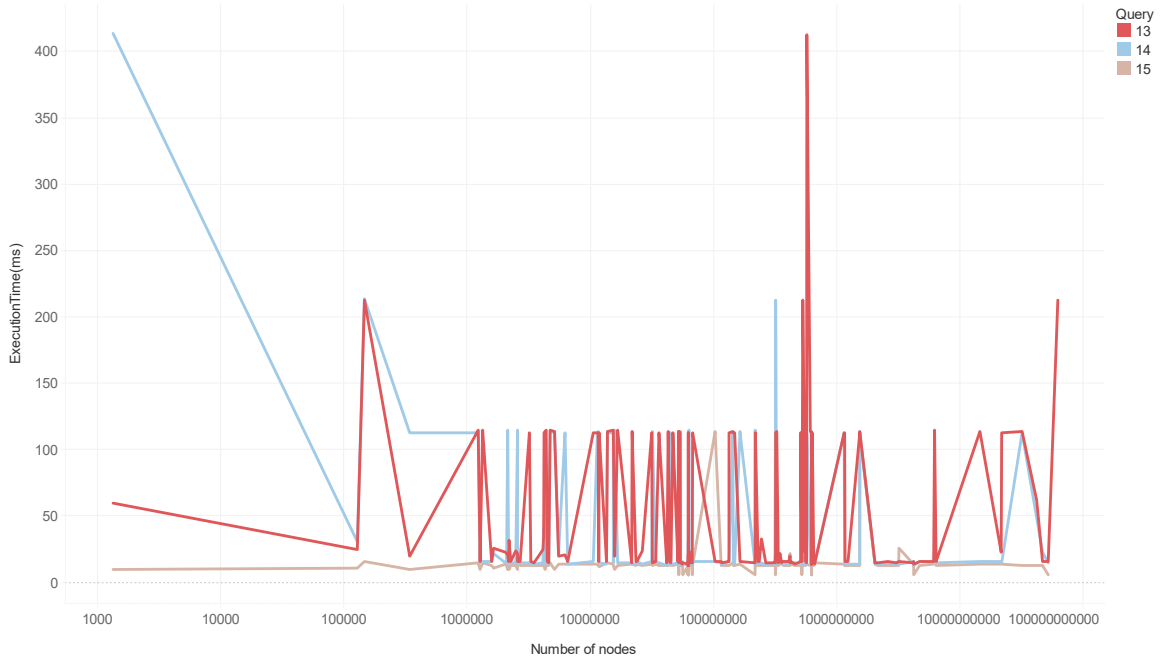


Figura 37: Tiempos de respuesta consultas 13, 14 y 15

Aunque ya no se repiten los tiempos de consulta de 0, 1 y 2 ms como pasaba antes, también aquí podemos encontrar patrones en estas gráficas. Para cada consulta se han hecho un total de 200 medidas, los valores más repetidos han sido:

- En la consulta 13, se repiten los siguientes valores
  - 14 ms → 16 veces
  - 15 ms → 43 veces
  - 16 ms → 55 veces
  - 113 ms → 26 veces
  - 114 ms → 14 veces
- En la consulta 14
  - Los tiempos 14 ms, 15 ms y 16 ms se repiten 71, 49 y 26 veces, respectivamente.
  - Los retardos más altos en este caso se repiten menos, repitiéndose 113 ms 16 veces y 114 ms 7 veces.
- En la consulta 15
  - Los tiempos 14 ms, 15 ms y 16 ms se repiten 104, 40 y 12 veces, respectivamente.
  - En cuanto tiempos altos, aquí no se repiten. Lo que sí podemos ver que no veíamos en los otros casos es que 6 ms se repiten 22 veces y 10 ms otras 10 veces.

3.5.4 Método 4

El último método que desarrollamos para ver si podíamos encontrar unos datos más fiables, consiste en lo siguiente:

- Se usó el código del primer método, con el añadido de una instrucción para pausar la ejecución del programa tras aumentar el número de nodos (después de ejecutar PruebaDemo).
- Una vez parado el programa, se reinicia la base de datos manualmente,
- Tras reiniciar la base de datos, se reinicia el programa y se repite esto con todas las muestras deseadas.

Este método se utilizó para las consultas 16 a 19.

Obviamente, con este método limitamos mucho el número de pruebas a hacer, ya que la base de datos no permite un reinicio desde un código externo, por lo que los reinicios tras cada consulta son manuales. A pesar de esto, se han podido sacar los resultados que mostraremos a continuación.

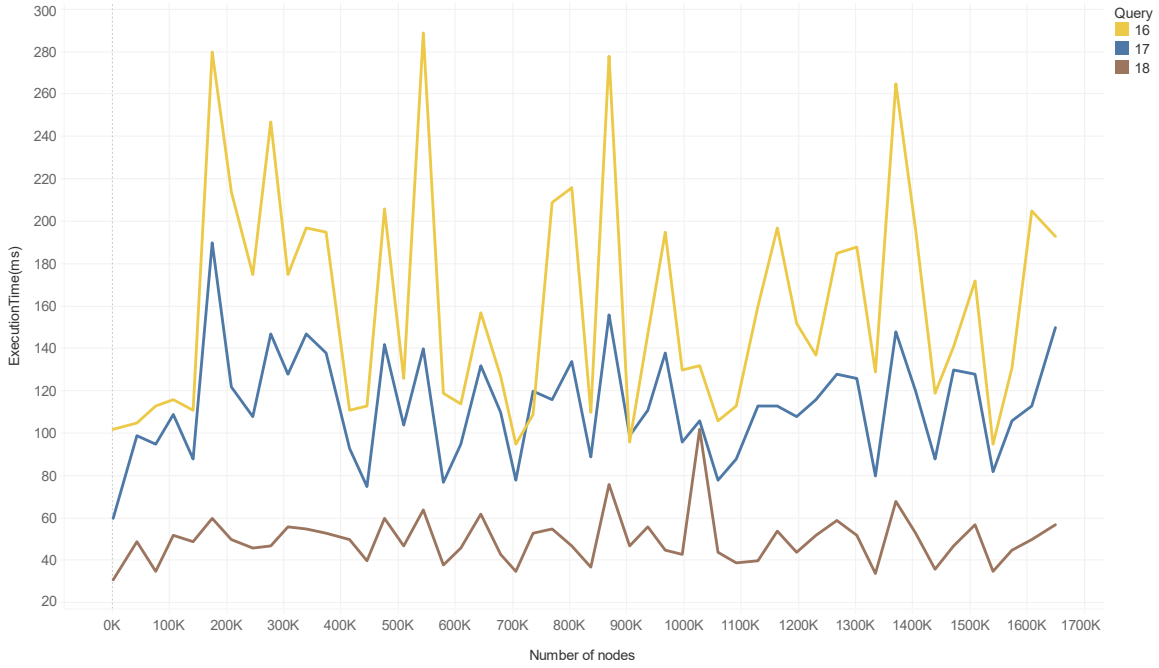


Figura 38: Tiempos de respuesta consultas 16, 17 y 18

En la Figura 38 sí que podemos ver tiempos más variables, y no se repite ninguno más de 3 veces cosa que sí tiene sentido.

A pesar de todo esto, no vemos una tendencia a crecer a medida que aumenta el número de nodos. Habiendo llegado a 1.700.000 nodos y más de 3 millones de relaciones, no se ve un incremento en los tiempos de respuesta. De hecho los máximos son:

- La consulta 16 llega a 289 ms como máximo con 543.147 nodos, lejos del máximo número de nodos.
- La consulta 17 alcanza su máximo en 190 ms, con 173.937 nodos en la base de datos.
- Y la consulta 18, llega a 102 ms con 1.026.494 nodos, algo más que los máximos anteriores.

Con el objetivo de recabar más información, decidimos probar con una consulta más, la consulta 19:

```
 MATCH (p1:Person:Paciente)<-[:PERTENECE_A]-
(o1:Observacion), (p2:Person:Paciente)<-[:PERTENECE_A]-
(o2:Observacion)
 WHERE o1.dato=o2.dato
 RETURN p1.dni AS Dni1, p2.dni AS Dni2
```

Esta consulta es más compleja que las anteriores, dado que no solo tiene que hacer las consultas, si no comparar nodos entre sí. Como vemos en la Figura 39, de media el tiempo es mayor que el de la consulta 16, llegando a un máximo de 1,558 segundos, mucho mayor que los máximos anteriores, pero posterior a ese máximo, los tiempos se estabilizan y vuelven a bajar, con esta nueva consulta tampoco vemos un crecimiento junto al número de nodos.

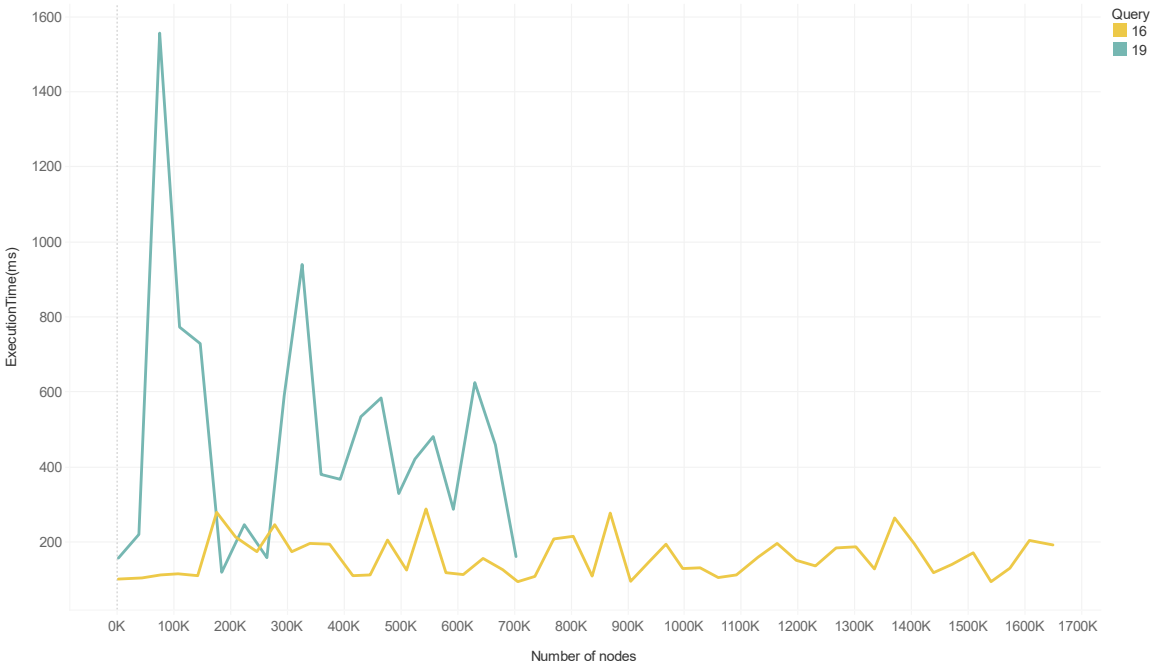


Figura 39: Comparación tiempos de respuesta consultas 16 y 19

3.5.5 Borrado de nodos y relaciones

Durante todo el trabajo se han añadido millones de nodos y relaciones a la base de datos. En un principio, cuando no se añadían tantos nodos, era posible borrarlos manualmente, pero llegó un límite que requería de mucho tiempo. Por esto, se decidió añadir una parte del código que los borrara, y llegados a esta parte del trabajo consideramos interesante medir el tiempo de estas consultas y ver cómo el sistema reacciona a estas consultas.

Las consultas de borrado se han limitado a 200.000 nodos / relaciones por cada una, ya que la base de datos da problema de memoria cuando se intentan borrar más.

Empezaremos mostrando las gráficas del borrado de las relaciones, ya que, se deben eliminar antes que los nodos porque estos no se pueden borrar hasta que se hayan quedado sin relaciones.

Como vemos en la Figura 40, vemos cómo la consulta de borrado 10 sí que tiene una tendencia a disminuir los tiempos, pero nada importante. Sin embargo, la consulta 11 no se ve que mantiene e incluso sube un poco los tiempos al disminuir el número de relaciones.

La media de estas consultas es mucho mayor que las consultas previas, siendo la media del borrado de las relaciones de 540 ms, cuando las otras consultas en muchos casos ni si quiera llegaba a esos niveles.

En estas consultas la memoria caché no interviene, ya que el borrado de las relaciones y/o nodos es puro procesamiento.

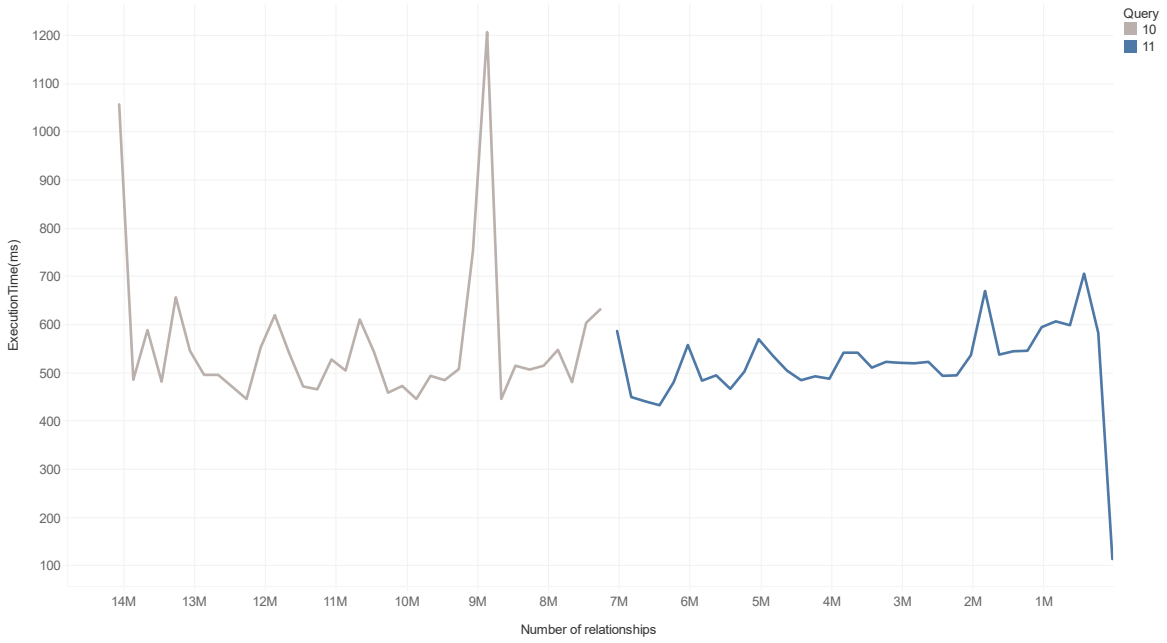


Figura 40: Borrado de relaciones

Donde sí podemos ver una disminución del tiempo de borrado es con los nodos.

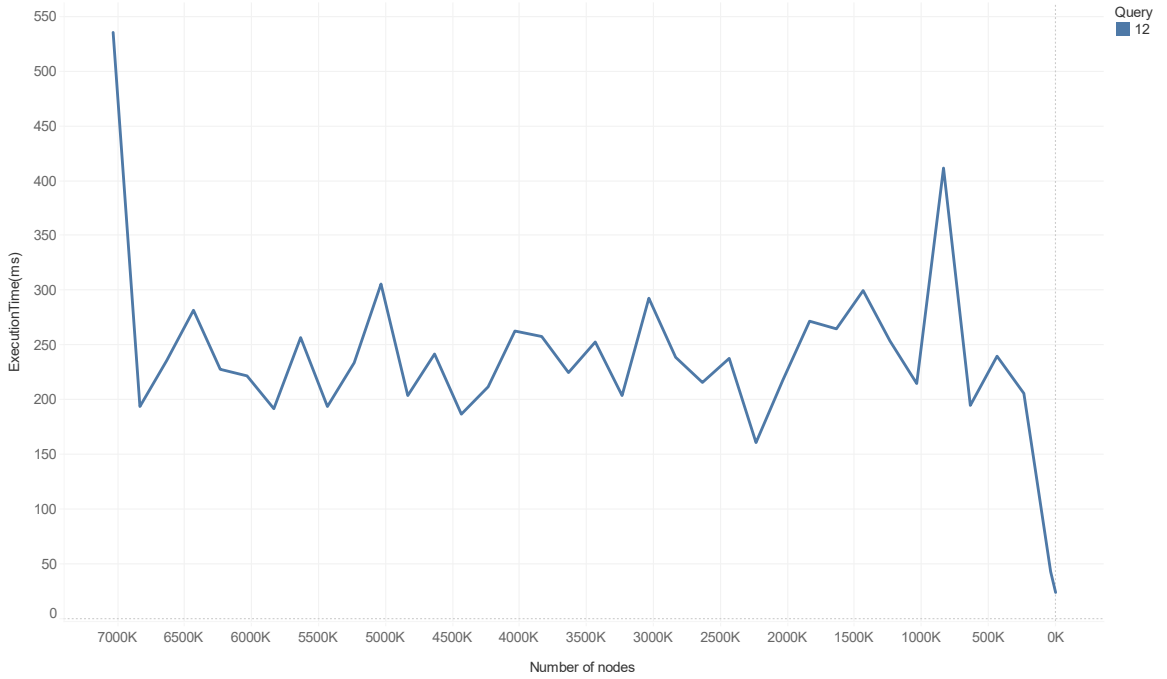


Figura 41: Borrado de nodos

En la Figura 41 se ve que se mantienen los tiempos, pero mirando la media, en este caso la media de borrado de nodos es de 235 ms, menos de la mitad que el borrado de relaciones. Esto puede ser debido a la menor carga que tiene la base de datos al tener todas las relaciones borradas.

3.5.6 Entorno gráfico

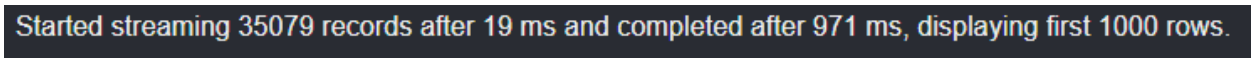
Durante el desarrollo de pruebas de rendimiento sobre una base de datos Neo4j, se observó una discrepancia significativa en los tiempos de ejecución de consultas dependiendo del entorno utilizado. Las mismas consultas arrojaron resultados con tiempos de ejecución menores cuando se realizaban a través de un programa en Java en comparación con las pruebas ejecutadas directamente en Neo4j Desktop. Este fenómeno puede ser explicado

por diversos factores debidos a la naturaleza del entorno de ejecución y la representación visual de los datos.

Para ilustrar esta discrepancia, se realizaron dos pruebas utilizando consultas básicas sobre una base de datos con un gran volumen de nodos:

```
Match(n)
Return n
```

Esta consulta recupera todos los nodos de la base de datos y los muestra en la interfaz de Neo4j Desktop. Tarda 19 ms para recuperar la información, pero 971 ms para mostrar los resultados visualmente.

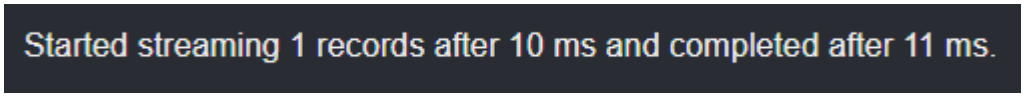


Algo entendible, ya que el número de nodos es alto, y le estamos pidiendo que muestre una gran cantidad de ellos. Sin embargo, si queremos contar el número de nodos, el tiempo de consulta total disminuye, al no tener que mostrar una gran cantidad de datos.

Entonces si ejecutamos la consulta:

```
MATCH (n)
RETURN count(n) AS numberOfNodes
```

A diferencia de la anterior, esta consulta solo cuenta el número de nodos sin visualizarlos. Esta consulta tarda 11 ms, significativamente menor que la consulta de recuperación y visualización.



El entorno de Neo4j Desktop está diseñado no solo para ejecutar consultas, sino también para proporcionar una representación visual de los resultados. Esta visualización implica renderizar gráficamente una gran cantidad de nodos y relaciones, lo que puede ser intensivo en términos de recursos y afectar el tiempo total de ejecución.

En contraste, una aplicación en Java que ejecuta la misma consulta a través del driver de Neo4j (Bolt) simplemente recibe los datos en formato de texto o JSON, sin necesidad de renderizarlos visualmente. Este ahorro de recursos se refleja directamente en tiempos de respuesta más rápidos.

La discrepancia en los tiempos de ejecución entre Neo4j Desktop y programas externos es un fenómeno esperado debido a las diferencias en cómo se manejan y representan los resultados. Comprender estas diferencias es fundamental para optimizar el rendimiento de las consultas y diseñar aplicaciones más eficientes basadas en Neo4j.

3.5.7 Conclusiones

El análisis detallado de diferentes métodos de consulta permitió evaluar con mayor precisión las condiciones que afectan al rendimiento. En el primer método, las consultas mostraron tiempos de ejecución muy bajos (entre 0 y 2 ms) en la mayoría de los casos, con algunos picos aislados. Estos resultados llevaron a la hipótesis de que la memoria caché influía en el rendimiento. Sin embargo, al reducir la memoria caché en el segundo método, el patrón se mantuvo, lo que indicaba que, si bien la caché afecta el rendimiento, no es el único factor determinante.

Para abordar esta cuestión, el tercer método incorporó la limpieza explícita de la caché mediante la sentencia CALL db.clearQueryCaches(), lo que permitió eliminar el impacto de la caché en los tiempos de ejecución. A pesar de ello, las consultas continuaron mostrando patrones repetitivos, aunque con tiempos ligeramente más altos que en los métodos anteriores. Este comportamiento sugiere que otros factores, como la organización de los datos y la estructura de las consultas, también influyen significativamente en el rendimiento. Las consultas más complejas, que implicaban comparaciones de nodos, presentaron tiempos de ejecución mayores.

El cuarto método introdujo reinicios manuales de la base de datos para eliminar cualquier influencia de la caché y observar el rendimiento en condiciones más "limpias". Este método reveló tiempos más variables y picos

menos predecibles, reflejando con mayor fidelidad el rendimiento real sin intervención de caché. A pesar de haber alcanzado un alto volumen de nodos y relaciones, no se observó una tendencia clara de incremento en los tiempos de ejecución, lo que indica que Neo4j maneja grandes volúmenes de datos de manera eficiente en términos de lectura y escritura.

El análisis de la consulta 19 revela un aspecto crucial sobre el rendimiento en Neo4j: las consultas que requieren comparaciones directas entre nodos tienden a tener tiempos de ejecución significativamente mayores. A diferencia de otras consultas que simplemente recorren nodos o relaciones, la consulta 19 implica una comparación de datos entre nodos Observación para identificar aquellos que comparten atributos similares. Este tipo de operación genera una carga adicional en el sistema, ya que Neo4j debe evaluar múltiples nodos y relaciones de manera iterativa, lo que aumenta considerablemente el tiempo de procesamiento.

El hecho de que la consulta 19 haya registrado tiempos máximos superiores a 1,5 segundos refleja la complejidad inherente a las consultas que requieren emparejar nodos basados en condiciones específicas. Este comportamiento subraya la importancia de diseñar cuidadosamente tanto la estructura de la base de datos como las consultas en sí.

Este caso demuestra que, a medida que aumenta la complejidad de las consultas, la optimización del modelo de datos se vuelve esencial. Diseñar una estructura de base de datos que favorezca la ejecución rápida de comparaciones y emparejamientos puede marcar la diferencia entre una consulta que tarda milisegundos y otra que tarda varios segundos

La consulta 19 sirve como recordatorio de que el rendimiento en Neo4j no depende únicamente del volumen de datos, sino también de cómo estos datos están organizados y de la forma en que se formulan las consultas. En bases de datos con grandes volúmenes de nodos y relaciones, una pequeña optimización en la estructura puede traducirse en mejoras significativas en el tiempo de respuesta.

Las pruebas de eliminación de nodos y relaciones mostraron resultados interesantes. Se identificó que el borrado de relaciones tiene un impacto mayor en el rendimiento que el borrado de nodos, con tiempos promedio más altos (540 ms frente a 235 ms). Esto sugiere que, en Neo4j, la gestión de relaciones conlleva una carga computacional superior, lo que debe ser tenido en cuenta durante operaciones de mantenimiento masivo. A medida que disminuye el número de relaciones, el tiempo de borrado de nodos tiende a estabilizarse, reflejando una menor carga sobre el sistema.

El análisis de rendimiento realizado sobre Neo4j ha permitido identificar diversas variables que influyen en los tiempos de ejecución de las consultas, revelando que el entorno donde se ejecutan tiene un papel fundamental en los resultados obtenidos. A lo largo de las pruebas, se evidenció que las mismas consultas presentaban tiempos de ejecución considerablemente menores cuando se realizaban desde un programa en Java, en comparación con las pruebas llevadas a cabo directamente en Neo4j Desktop. Esta disparidad se explica, principalmente, por la diferencia en la representación de resultados: mientras Neo4j Desktop debe renderizar gráficamente los nodos y relaciones, el programa en Java simplemente procesa y recibe los datos en formato texto o JSON, eliminando la carga visual asociada.

4 CONCLUSIONES Y LÍNEAS FUTURAS

4.1 Conclusiones

El presente trabajo ha permitido evaluar de forma experimental el rendimiento de la base de datos Neo4J en un entorno simulado de monitorización de pacientes, representando escenarios propios del ámbito sanitario. A lo largo del trabajo se han realizado pruebas de inserción masiva de datos, simulando lo que sería el flujo de información que se genera en sistemas IoMT. Las conclusiones que podemos sacar del trabajo realizado se exponen a continuación.

4.1.1 Consumo de CPU

A medida que se incrementó el número de dispositivos simulados y la frecuencia de envío de datos, se observó un aumento progresivo en el consumo de CPU. Si bien el sistema mantuvo un rendimiento estable durante la mayoría de las pruebas, con tasas superiores a 3000 datos por segundo comenzó a mostrar signos de saturación.

Es importante destacar, que a pesar de que haya llegado a saturarse en muchos casos, el sistema siempre se adapta y con el paso del tiempo reduce drásticamente el consumo de CPU, lo que indica que el sistema podría perder algunos datos en los primeros segundos, pero con el tiempo se estabiliza y recepciona los datos sin problema.

La figura 42 representa la media de todas las pruebas con respecto al paso del tiempo. No filtra por dispositivos ni frecuencia.

Si nos fijamos en la figura 42, se observa lo que estamos comentando, con el paso del tiempo el consumo de CPU tiende a disminuir.

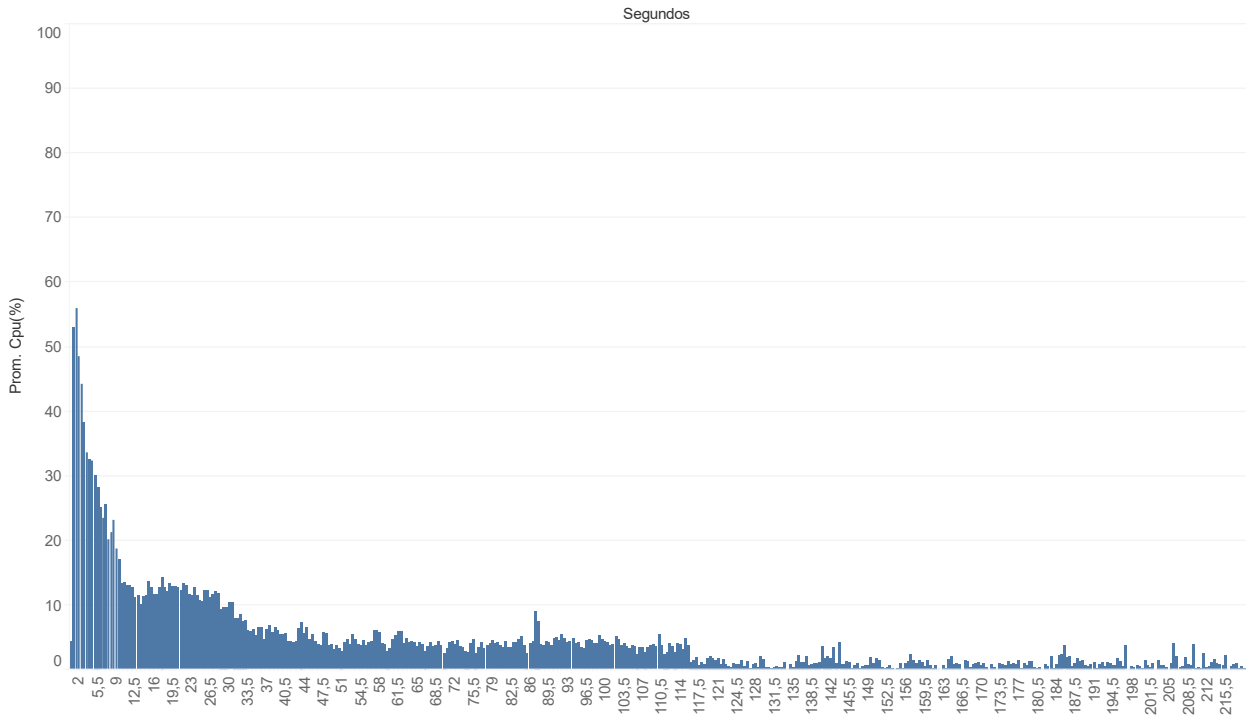


Figura 42: Media de CPU con el paso del tiempo.

En pruebas más agresivas, el sistema alcanzó picos superiores al 95% de uso de CPU, provocando cuellos de

botella y ralentización en la inserción de datos. El mayor cuello de botella encontrado fue con 200 dispositivos a frecuencia de 25 ms, en la gráfica vimos que se saturó al principio y de repente el consumo cayó por completo durante unos 5 segundos para posteriormente volver a tener un consumo alto. En ello se identifica un cuello de botella.

Este comportamiento evidencia la necesidad de optimizar el uso de recursos y mejorar la capacidad de Neo4J para manejar cargas de trabajo más pesadas sin comprometer la integridad de los datos.

4.1.2 Consumo de memoria

El uso de memoria RAM se mantuvo elevado de forma constante, lo que sugiere que Neo4J realiza una gestión agresiva de memoria para optimizar el rendimiento de las consultas. Sin embargo, esta estrategia puede convertirse en una limitación en sistemas con recursos de hardware limitados. A pesar de este elevado consumo, la base de datos mantuvo su estabilidad, lo que indica que Neo4J prioriza la disponibilidad y velocidad de las consultas por encima del ahorro de memoria.

4.1.3 Manejo de consultas

Las pruebas de consulta arrojaron resultados positivos incluso con bases de datos de gran tamaño, lo que subraya la capacidad de Neo4j para optimizar búsquedas y realizar consultas complejas de forma eficiente. Este comportamiento permite su aplicación en entornos críticos donde el acceso a información en tiempo real es vital. Las consultas que involucran múltiples relaciones entre nodos se ejecutaron con rapidez, lo que sugiere que el motor de búsqueda de Neo4j está bien adaptado para trabajar con datos densamente conectados.

Neo4j ha demostrado ser una herramienta robusta para gestionar grandes volúmenes de datos interconectados, ofreciendo tiempos de respuesta adecuados en escenarios de baja a media carga de trabajo. La eficiencia de las consultas basadas en relaciones complejas resalta la idoneidad de este tipo de base de datos para sistemas donde las conexiones entre entidades son cruciales. La estructura de grafos facilita la recuperación de datos y el análisis de redes complejas, lo que permite detectar patrones y correlaciones que serían difíciles de identificar mediante bases de datos relacionales tradicionales.

El mayor inconveniente en cuanto a las consultas sería el entorno gráfico de Neo4j, que añade un retraso de hasta 100 veces el tiempo de la consulta en los casos que se necesita representar una gran cantidad de nodos y relaciones. Esta restricción desaparece cuando se prescinde del entorno gráfico.

En general, Neo4j ha demostrado ser una opción viable y eficiente para sistemas sanitarios que requieren el análisis y almacenamiento de datos interrelacionados. No obstante, es necesario considerar la escalabilidad y el consumo de recursos en implementaciones a gran escala. Este trabajo proporciona una base sólida para futuras investigaciones que busquen mejorar la infraestructura de bases de datos en el ámbito sanitario.

4.2 Líneas futuras

A partir de los resultados obtenidos, se identifican diversas líneas futuras que podrían contribuir a mejorar y ampliar el alcance del proyecto:

- Rendimiento:

Para llevar más al límite la base de datos sería interesante hacer una base de datos con más relaciones, es decir, tener una estructura más compleja, además de probar consultas más complejas. Todo esto con el fin de llevar más al límite la base de datos.

- Paralelización y distribución de carga:

Una de las mayores restricciones que ha tenido este trabajo ha sido el uso de un solo dispositivo. Tanto la base de datos como los “dispositivos” estaban alojados en el mismo ordenador. Para investigaciones futuras sería

interesante alojar la base de datos en un Docker, de esta manera tendría más capacidad de computación que la que tendría en un ordenador.

Sería interesante también implementar un entorno distribuido que permita el procesamiento en paralelo de los datos, de manera que no se localicen todos los hilos desde un mismo dispositivo. Esta distribución de carga podría llevar a un mayor número de dispositivo mandando desde distintas localizaciones, esto se acerca más a la realidad y el rendimiento mejoraría.

- Integración con sistemas reales de IoMT:

Para comprobar definitivamente su posible implementación, la realización de pruebas de campo en un entorno real sería indispensable. Realizar pruebas con dispositivos médicos reales mandando los datos a la base de datos (idealmente virtualizada en un Docker) facilitaría la validación del sistema en situaciones operativas reales, identificando posibles limitaciones que en un entorno simulado no podríamos detectar.

- Estudio de soluciones híbridas:

En el caso de que algún estudio futuro no dé los resultados deseados, o no sean óptimos, otro posible estudio sería la combinación de bases de datos basadas en grafos con otros sistemas relacionales o NoSQL, lo que podría ofrecer un equilibrio entre el rendimiento y la flexibilidad, permitiendo almacenar datos críticos de forma eficiente y realizar consultas complejas sobre grafos. El desarrollo de arquitecturas híbridas permitiría aprovechar lo mejor de ambos mundos, utilizando bases de datos relacionales para datos estructurados y Neo4J para datos con relaciones densas.

Estas líneas futuras no solo contribuirán a mejorar el rendimiento del sistema, sino que también ampliarán su aplicabilidad en entornos críticos donde la monitorización y el análisis de datos juegan un papel fundamental en la toma de decisiones médicas.

REFERENCIAS

[1] Rose, K., Eldridge, S., & Chapin, L. (2015). The internet of things: An overview. The internet society (ISOC), 80(15), 1-53.

[2] Robinson, I., Webber, J., & Eifrem, E. (2015). Graph Databases: New Opportunities for Connected Data. O'Reilly Media.

[3] Vishnu, S., Ramson, S. J., & Jegan, R. (2020, March). Internet of medical things (IoMT)-An overview. In 2020 5th international conference on devices, circuits and systems (ICDCS) (pp. 101-104). IEEE.

[4] Neo4j, Inc. (2021). Neo4j Java Driver 4.3 - User Manual. Retrieved from https://neo4j.com/docs/java-manual/current/

[5] Partner, C. (2014). Neo4j in Action. Manning Publications.

[6] Gamma, E., & Beck, K. (1999). Contributing to Eclipse: Principles, Patterns, and Plugins. Addison-Wesley Professional.

[7] Smith, W. (2014). OSHI: Operating System and Hardware Information. Available: https://github.com/OSHI/OSHI

[8] "Tableau Software: Data Visualization for Business Intelligence," Tableau Software, 2023. [Online]. Available: https://www.tableau.com/