Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE VALLADOLID

Grado en Ingeniería Informática
Mención en Computación

Estudio de rendimiento de bases de datos para series temporales aplicado a la construcción de modelos basados en datos para la mejora de eficiencia energética en edificios inteligentes

---

Autor: Patricia Aguado Labrador

Tutor: José Belarmino Pulido Junquera

A mi abuela Carmen,
por tu amor incondicional y sin límites,
por ser única e irreemplazable en mi corazón,
por tu creencia en mí que hizo que llegase a donde estoy ahora.
Un besito al cielo.

Agradecimientos

Este proyecto está firmado por un sólo autor, pero son muchas las personas que han contribuido en él para que salga adelante. Estoy agradecida con todas las personas que han formado parte de mi vida durante la realización de este trabajo, porque de alguna manera me han aportado aquello que estaba a su alcance en cada momento y han hecho que siguiera adelante cuando sentía que la vida me venía demasiado grande. En especial, quiero agradecer a mi tutor, Belarmino Pulido, por su apoyo y disposición para ayudarme y guiarme en esta etapa académica; gracias por la paciencia y la confianza depositada en mí.

Agradezco a mi familia por su apoyo, a mis amigos por su amor y fe en mi valía, a la vida por los baches que me ha puesto en el camino y que me han enseñado muchas cosas, y a mi psicóloga por darme las herramientas que necesitaba para convertirme en la persona que soy hoy en día.

Por último, quiero agradecerme a mí misma el esfuerzo puesto en cada caída para conseguir levantarme siendo más fuerte, haciéndome llegar a donde estoy ahora.

Resumen

Como resultado del crecimiento exponencial de los datos en numerosos ámbitos, los sistemas gestores de datos se han visto obligados a evolucionar para resolver los retos que plantea el "Big Data" o datos masivos. En el ámbito de la sostenibilidad, los edificios inteligentes generan grandes cantidades de datos constantemente y necesitan emplear métodos basados en datos para intentar mejorar su rendimiento y eficiencia energética en ausencia de modelos detallados. Para la construcción de modelos es necesario analizar datos que tienen asignadas etiquetas temporales, por ello para el presente trabajo contamos con datos de series temporales recopilados del edificio inteligente LUCIA de la Universidad de Valladolid.

En este proyecto nos planteamos estudiar distintos sistemas gestores de bases de datos temporales para usarlos posteriormente en sistemas de aprendizaje automático. Estudiamos las cualidades de cuatro sistemas gestores de bases de datos que existen hoy en día en el mercado y analizamos su rendimiento a través de distintas consultas que están relacionadas con consultas típicas de recuperación y análisis de datos. Además, se desarrolla una herramienta que ayuda a la visualización de los resultados.

Palabras clave: rendimiento bases de datos, bases de datos temporales, Big Data, series temporales, eficiencia energética, edificios inteligentes

Abstract

As a result of the exponential growth of data in many areas, data management systems have been forced to evolve to meet the challenges posed by Big Data. In the field of sustainability, smart buildings are constantly generating large amounts of data and they need to employ data-driven methods to try to improve their performance and energy efficiency in the absence of detailed models. In order to build models it is necessary to analyze data that have time labels assigned to them, so for the present work we use time series data collected from the LUCIA smart building of the University of Valladolid.

In this project we propose to study different temporal database management systems for later use in machine learning systems. We study the features of four database management systems that exist today in the market and analyze their performance through different queries that are related to typical data retrieval and analysis queries. Moreover, a tool is developed to help in the visualization of the results.

Key words: performance databases, time-based databases, Big Data, time series, energy efficiency, smart buildings

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Desde el punto de vista tecnológico, la revolución de los datos está transformado la sociedad a pasos agigantados. El desarrollo de herramientas de código abierto para el almacenamiento de grandes cantidades de datos, el auge de la utilización de dispositivos conectados a Internet y nosotros mismos, favorecemos cada día el aumento del volumen de datos mundial, el cual tiene actualmente un crecimiento exponencial.

Los macrodatos, o más comúnmente conocidos como "Big Data"[33], son la clave de la transformación digital que estamos viviendo. Estos son conjuntos de datos de gran tamaño, variabilidad y velocidad de crecimiento que podemos encontrar en infinidad de ámbitos como, por ejemplo, en política haciendo posible el análisis de datos recogidos de los potenciales votantes; en salud facilitando el análisis de historiales médicos; en sostenibilidad mediante el tratamiento de datos recogidos de edificios; en empresas de cualquier sector donde clientes, productos y actividades de negocio hagan uso de internet y todo quede registrado digitalmente; así como, muchos procesos industriales que recopilan cientos o miles de datos cada instante de cada proceso.

Centrándonos en el ámbito de la sostenibilidad cabe decir que los edificios, los cuales son una parte fundamental de nuestra vida diaria, son los responsables del 40 % del consumo energético de la Unión Europea y del 36 % de las emisiones de gases de efecto invernadero[8]. A lo largo del tiempo, tanto la Unión Europea como la Organización de Naciones Unidas han desarrollado proyectos y legislaciones para facilitar el desarrollo sostenible. De hecho, la necesidad de mejorar la gestión energética de edificaciones y la posibilidad de disponer de cantidades ingentes de datos, obtenidos a través de dispositivos tecnológicos, presenta un abanico de posibilidades de cara a un futuro energético más sostenible.

La Comisión Europea puso en marcha el Pacto Verde Europeo en 2019, consistente en un conjunto de iniciativas para situar a la UE en el camino de la transición ecológica. En 2021, se presenta una nueva propuesta orientada a la armonización de las normas de eficiencia energética de los edificios y descarbonización del parque inmobiliario de la UE de aquí a 2050. De esta manera se quieren alcanzar diferentes metas[4]:

1. Todos los edificios nuevos tendrán cero emisiones a partir de 2030.
2. El 15 % del parque inmobiliario de cada Estado miembro con peor rendimiento deberá actualizar y mejorar su certificado de eficiencia.

3. Planes nacionales para abandonar progresivamente los combustibles fósiles en la calefacción y refrigeración de aquí a 2040.

4. Nuevas normas que fomenten el uso de las TIC (tecnologías de la información y las comunicaciones) y de las tecnologías inteligentes para garantizar el funcionamiento eficiente de los edificios.

En relación con el punto número cuatro, hoy en día el incremento de herramientas software hacen posible el almacenamiento y la comunicación de datos ya que, por ejemplo, sistemas SCADA (Supervisory Control And Data Acquisition) y sistemas de monitorización o BMS (Building Management Systems), nos brindan la posibilidad de procesar y almacenar grandes cantidades de datos en tiempo real. Además, disponemos de todos los avances que se han llevado a cabo en la informática y en especial en la ciencia de datos, la que nos permite analizar y extraer conocimiento de simples datos almacenados. Por otro lado, hay que tener en cuenta las dificultades existentes en torno al análisis de datos o a la aplicación de técnicas de aprendizaje automático sobre estos, ya que podemos encontrarnos con datos inconsistentes, falta de datos precisos o un tamaño de datos que condicione los recursos necesarios o los algoritmos que se puedan emplear.

Se plantea entonces el problema de la calidad de datos y del almacenamiento de estos. A la hora de recopilar macrodatos sobre un edificio podemos encontrarnos con datos de distintas fuentes y de diversos tipos, así como localizar valores ausentes o valores atípicos, por lo que el preprocesado de estos será una parte fundamental en el tratamiento del big data. Por otro lado, debemos contar con una tecnología de almacenamiento eficiente que sea capaz de lidiar con amplios volúmenes de datos, ciberataques, compatibilidad de los diferentes tipos de datos o la llegada de una cantidad mayor en el futuro.

En la actualidad y debido a la necesidad de almacenar datos para su posterior utilización, se han llevado a cabo grandes avances en el desarrollo de diversos tipos de bases de datos, en función de las necesidades que vayamos a tener. En el caso que hemos expuesto anteriormente, a la hora de recopilar datos de un edificio nos encontraremos con que estos están ligados a la variable tiempo, por lo que debemos contar con bases de datos capaces de trabajar de manera eficiente con este tipo de datos, a los que se les suele denominar series temporales.

1.1. Contexto

Para la realización de este trabajo de fin de grado, cuyos objetivos se exponen en el siguiente apartado, contaré con un conjunto de datos pertenecientes al edificio LUCIA (Lanzadera Universitaria de Centros de Investigación Aplicada), situado en el Campus Miguel Delibes de la Universidad de Valladolid. El marco de trabajo que engloba este proyecto es el GIR GSI (Grupo de Investigación Reconocido Grupo de Sistemas Inteligentes) dentro de la línea de investigación en “Tecnologías Big Data” y “Técnicas de aprendizaje automático” para la gestión eficiente de energía en edificios inteligentes.

El edificio LUCIA se diseñó con el fin de verificar hipótesis sobre las que se asienta la evaluación medioambiental de edificios e investigar aspectos sociales de la edificación sostenible

y del uso únicamente de energías renovables. Las estrategias principales para convertir al LUCIA en un referente en eficiencia energética y arquitectura sostenible se basan en un diseño bioclimático del edificio (cuenta con un 46 % de superficie acristalada), utilización de biomasa como energía primaria y cogeneración de ésta para climatización y producción de electricidad, recuperación de agua y gestión de residuos en la fase de construcción, uso y demolición[58].

Gracias al enfoque llevado a cabo se consigue una reducción de más del 50 % en demanda energética, un ahorro del 45 % en el gasto de iluminación y un ahorro superior al 60 % en el gasto de energía del propio edificio y un 30 % en los edificios anexos.

En la Figura 1.1 podemos ver las secciones del edificio y algunas de sus principales peculiaridades como pueden ser los lucernarios, la cubierta vegetal que permite el filtrado de agua de lluvia o el ascensor con regeneración energética. El edificio consta de cuatro plantas[58]:

- Planta sótano, donde se encuentran distintas instalaciones como pueden ser el intercambiador geotérmico o el multiciclón.
- Planta baja, donde podemos encontrar el centro de procesado de datos y laboratorios dedicados al desarrollo de la Sociedad Digital del Conocimiento.
- Primera planta, dedicada a estudios genéticos y prevención de metabolopatías.
- Segunda planta, destinada a la investigación de nutrición, alimentación y dietética.

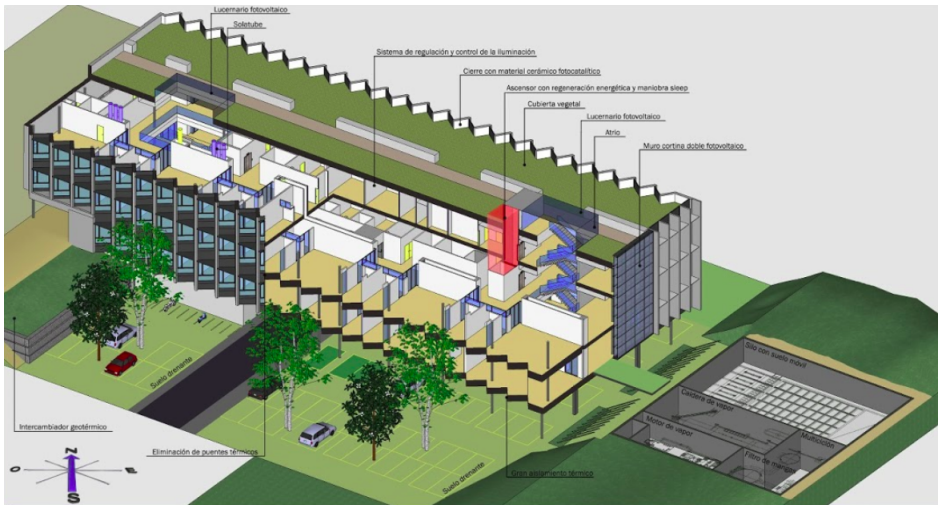


Figura 1.1: Secciones edificio LUCIA (Fuente: blog LUCIA)

Además, destacar que el edificio consiguió en el pasado 98 puntos LEED Platinum (Leadership in Energy and Enviromental Desing), 4,51 puntos sobre 5 en la certificación verde de la Green Building Council España (GBCe) en 2015[58] y, recientemente, consiguió la certificación internacional de protección contra el contagio del virus COVID-19 por la Organización Mundial para la Seguridad y Salud en Espacios Interiores[5].

Los datos recogidos por en el edificio se almacenan en el BMS del LUCIA. Estos consisten en varios cientos de medidas, como se comentará más adelante, que tienen la característica de

representar series temporales de datos. Para implementar distintos algoritmos de aprendizaje automático será necesario analizar y recuperar parte de estos datos por lo que nos planteamos cuál sería la arquitectura de bases de datos más adecuada para ellos.

1.2. Objetivos

El presente trabajo de fin de grado tiene como objetivo analizar las prestaciones en términos de tiempo de acceso y requisitos de almacenamiento de sistemas de adquisición y almacenamiento de datos relativos a un edificio inteligente como es el LUCIA. El proyecto constará de varias tareas enfocadas al estudio de diferentes bases de datos temporales para su uso en tiempo real, las cuales son:

- **Estudiar las prestaciones de distintas bases de datos temporales**: se elegirán varias bases de dados enfocadas a almacenar datos ligados a la variable tiempo y se estudiarán sus características y configuraciones.
- **Analizar y almacenar datos relacionados con el consumo energético**: se estudiarán los datos almacenados sobre el edificio LUCIA y se se estudiará su almacenamiento en diferentes bases de datos enfocadas a las series de tiempo. Se realizarán pruebas de rendimiento con el fin de elegir la más adecuada para los datos disponibles en este trabajo.
- **Crear una herramienta de visualización para los datos energéticos del edificio**: se realizará una representación gráfica del rendimiento obtenido en los experimentos y de los resultados de las pruebas realizadas en cada base de datos.

1.3. Estructura de la memoria

A continuación, se presentan los 9 capítulos en los que se divide la memoria de este trabajo:

- **Capítulo 1 - Introducción**: en el capítulo actual presentaremos el proyecto, daremos un contexto y explicaremos los objetivos de este trabajo de fin de grado y la estructura de la memoria.
- **Capítulo 2 - Planificación**: en este capítulo hablaremos sobre temas relacionados con la gestión del proyecto como son la metodología utilizada, la planificación temporal, la gestión de riesgos y la estimación económica.
- **Capítulo 3 - Caso de estudio**: en este capítulo presentaremos el caso de estudio de este trabajo, profundizando en la organización del edificio inteligente del que proceden los datos utilizados en las pruebas de rendimiento.
- **Capítulo 4 - Estado del arte**: en este capítulo ahondamos un poco en el contexto de este trabajo y desarrollamos la evolución de las bases de datos presentando los diferentes tipos que han surgido desde su aparición hasta la actualidad.

- **Capítulo 5 - Datos**: en este capítulo se realiza una descripción de los datos con los que trabajaremos, su estructura y sus características, así como el proceso de limpieza que se lleva a cabo con ellos.
- **Capítulo 6 - Diseño e implementación**: en este capítulo se realiza una descripción de las bases de datos que utilizaremos, se describen las decisiones a la hora de implementar el caso de estudio y se presenta la estructura general que tendrán los datos en las pruebas.
- **Capítulo 7 - Estudio de rendimiento de bases de datos**: en este capítulo se enuncian las métricas de evaluación de rendimiento que vamos a utilizar y se analizan los resultados obtenidos mediante tablas y gráficos comparativos.
- **Capítulo 8 - Herramienta de visualización desarrollada**: en este capítulo se presenta el proceso de análisis, de modelado del sistema y de diseño de la aplicación web desarrollada para la visualización de los resultados.
- **Capítulo 9 - Conclusiones y trabajo futuro**: en este capítulo se analiza el resultado global del proyecto exponiendo las conclusiones a las que hemos llegado y las posibles mejoras de este trabajo.

Capítulo 2

Planificación

2.1. Metodología y ciclo de vida del proyecto

A la hora de planificar el proyecto en la fase inicial, decidimos utilizar una metodología tradicional empleando el modelo en cascada[46] para llevar a cabo los objetivos planteados en la Sección 1.2. Como vemos en la Figura 2.1 el desarrollo de la metodología en cascada es lineal y secuencial, aunque se podrían producir bucles de retroalimentación si a lo largo del trabajo fuese necesario realizar cambios en los requisitos iniciales teniendo en cuenta que es raro que un proyecto real siga el flujo secuencial propuesto. Teniendo en cuenta que al trabajo inicialmente se le otorgan un número de horas determinado para su ejecución y que tenemos unos objetivos bien definidos, aplicaremos el paradigma incremental para no desviarnos de los objetivos principales y obtener una retroalimentación por parte del tutor del trabajo.

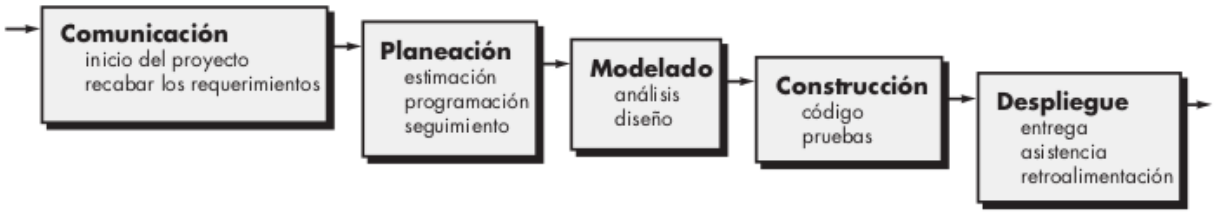


Figura 2.1: Modelo de la cascada (Fuente: capítulo 2. Modelos del proceso[46])

Dentro de esta metodología en cascada, nos basaremos también en un ciclo de vida clásico representado en la Figura 2.2 que sirve para definir las fases en las que se va a dividir nuestro proyecto desde el inicio hasta el cierre del mismo. Las siete fases de nuestro proyecto serán:

- Planificación: el objetivo de esta fase es realizar un análisis del proyecto, determinar las actividades que tenemos que llevar a cabo, analizar los riesgos que pueden surgir y estimar los costes para estudiar la viabilidad del proyecto.
- Estudio: el objetivo de esta fase es adquirir o profundizar en los conocimientos necesarios sobre los temas que vayamos a tratar y las herramientas que vayamos utilizar para desarrollar el proyecto.

- Análisis: el objetivo de esta fase es realizar un análisis de los datos con los que vamos a trabajar y del caso de estudio para proporcionar una base estable para la parte de diseño e implementación.
- Diseño: el objetivo de esta fase es revisar los requisitos y transformarlos en un diseño concreto y específico que pueda ser implementado.
- Implementación: el objetivo de esta fase es desarrollar el software y optimizarlo para realizar las pruebas de rendimiento planteadas.
- Validación: el objetivo de esta fase es la de comprobar que el software desarrollado cumple con los requisitos establecidos y que se han conseguido los objetivos planteados en la fase inicial del proyecto.
- Documentación: el objetivo de esta fase es documentar cada paso seguido en la elaboración del proyecto y generar el manual del instalador.

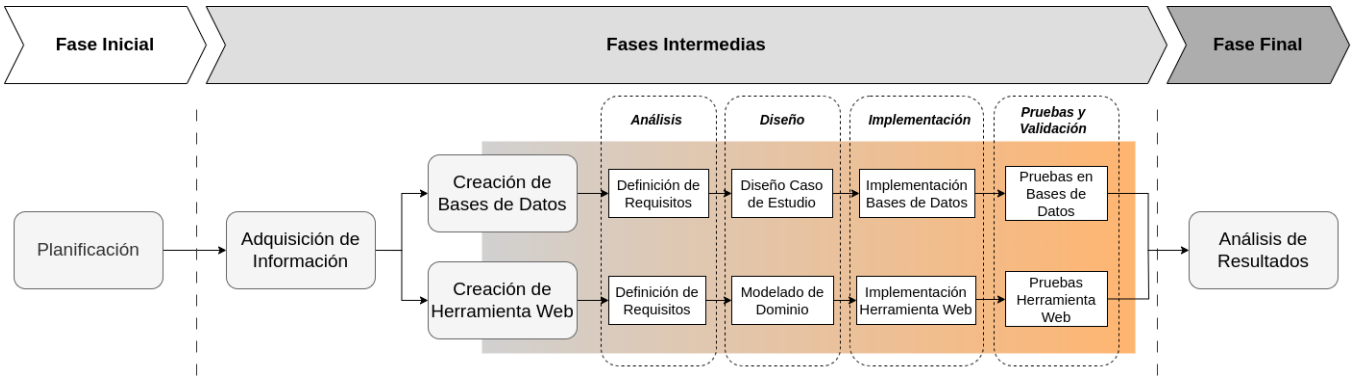


Figura 2.2: Ciclo de vida en cascada del proyecto (Fuente: elaboración propia)

2.2. Gestión del alcance del proyecto

El enunciado del alcance nos permite definir las fronteras del proyecto definiendo el trabajo que se realizará para entregar este TFG (Trabajo de Fin de Grado) con las características especificadas[21]. De esta manera, el propósito fundamental de este proyecto es llevar a cabo el estudio de diferentes bases de datos temporales que nos permitan almacenar de la mejor manera posible datos de series temporales de eficiencia energética. Diseñaremos pruebas para evaluar la capacidad y velocidad de consulta de determinadas bases de datos con el fin de analizar el rendimiento de estas. Además, se plantea la visualización de los resultados para apoyar el análisis acerca del rendimiento de las bases de datos por lo que de manera complementaria desarrollaremos una aplicación que nos permita analizar de forma gráfica los resultados sin ser el grueso del proyecto y haciendo hincapié en que el estudio de las distintas arquitecturas de bases de datos es la parte más valiosa del TFG.

Las fases que hemos seleccionado para nuestro proyecto en la sección anterior forman el primer nivel del EDT (Estructura de Desglose del Trabajo)que podemos observar en la Figura 2.3. En esta descomposición jerárquica se muestran una definición más detallada del trabajo que llevaremos a cabo.

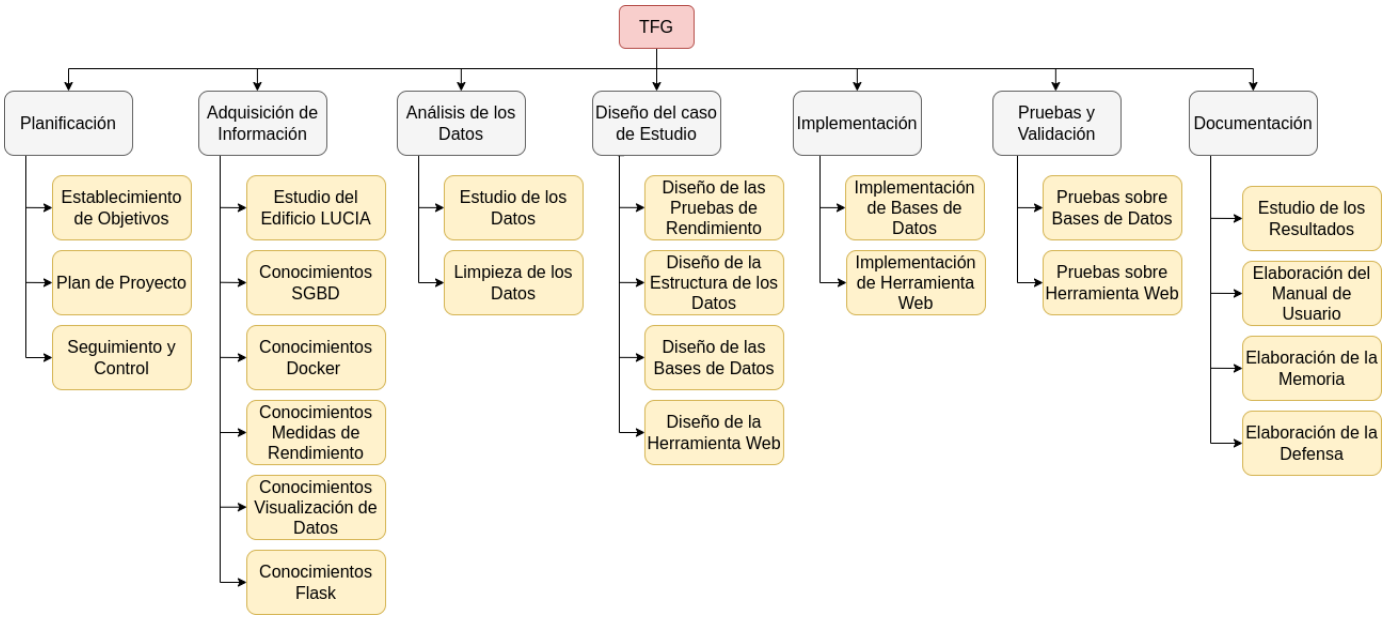


Figura 2.3: Estructura de Desglose del Trabajo

2.3. Plan de trabajo

En esta sección presentaremos el cronograma del proyecto con las tareas a desarrollar en cada una de las fases, su duración y calendarización.

En la Figura 2.4, se muestra el Diagrama de Gantt que nos sirve para representar visualmente el cronograma del proyecto a lo largo del tiempo. En la figura vemos que aparecen los entregables del proyecto en color negro y las tareas agrupadas por colores según su cometido:

- En color amarillo tenemos tareas de planificación, en un primer momento se establecen los objetivos del proyecto junto con el tutor del TFG y a continuación se realizan las tareas de definición de requisitos, alcance, actividades y riesgos.
- En color verde claro tenemos tareas relacionadas con la adquisición de conocimientos sobre el contexto del proyecto, el caso de estudio, los sistemas gestores de bases de datos para series temporales existentes y herramientas que utilizaremos para el desarrollo software.
- En color naranja están las tareas de diseño de las pruebas, las bases de datos que vamos a utilizar y de la estructura de los datos que utilizaremos.
- Una vez establecido el diseño tenemos en color azul oscuro tareas de implementación que consisten en montar el entorno de Docker y alojar las bases de datos.
- Finalizadas las tareas de implementación de bases de datos realizaremos la tarea de color morado enfocada a la ejecución de las pruebas de rendimiento y a la comprobación de la correcta ejecución de estás.
- Cuando ya contamos con los resultados de las pruebas pasamos a las tareas de color azul claro que se encargan del modelado de software de la aplicación web que desarrollaremos

para la visualización de resultados y la documentación de ésta a través del manual del instalador.

- En color verde oscuro tenemos la tarea de realizar pruebas de validación de la herramienta de visualización desarrollada.
- Por último, en color rosa tenemos la tarea de elaboración de la memoria del trabajo y en color fucsia las reuniones periódicas que tendremos con el tutor para realizar un seguimiento del TFG.

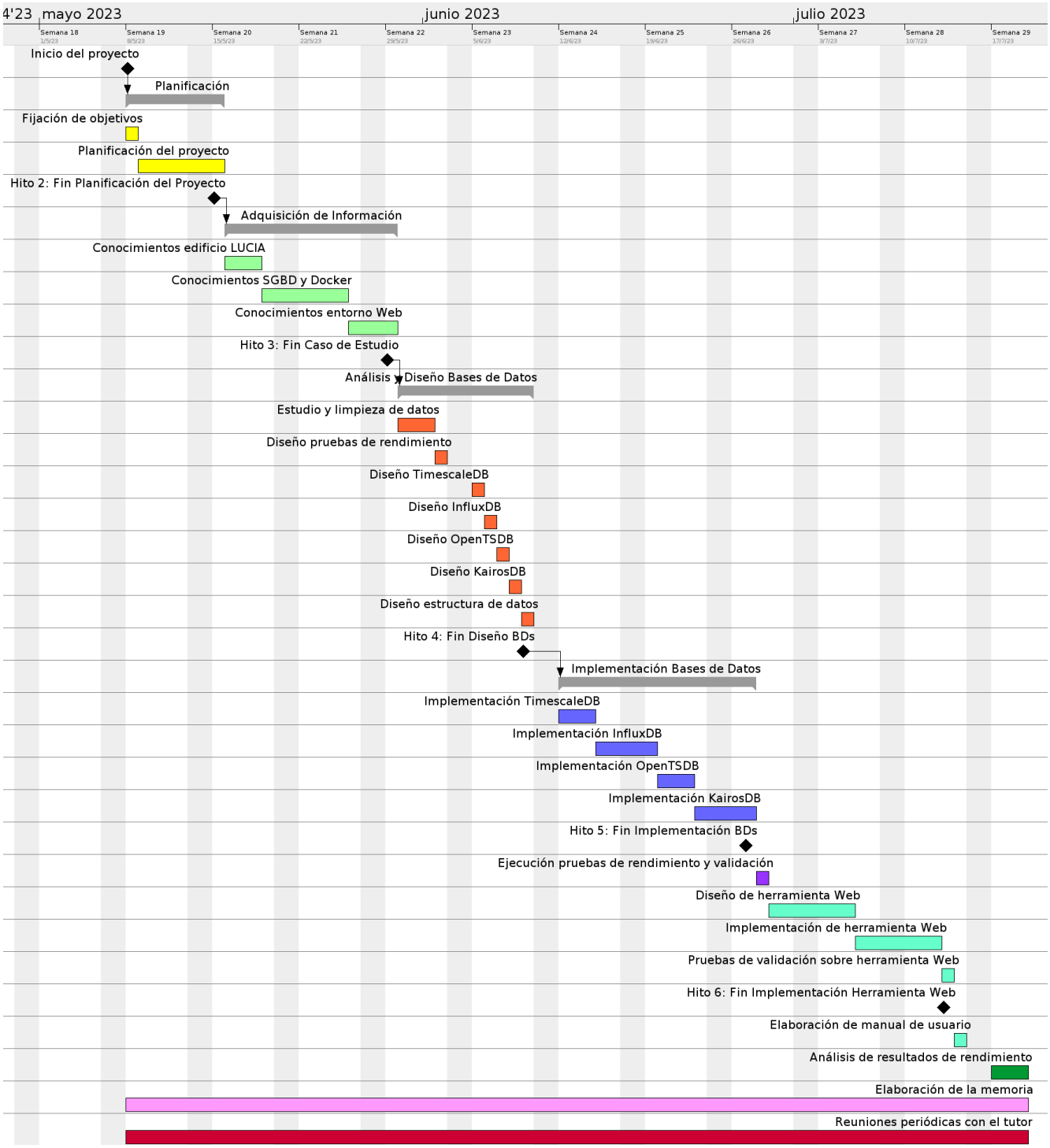


Figura 2.4: Diagrama de Gantt

2.3.1. Seguimiento del plan de trabajo

En la Tabla 2.1 podemos ver en azul la calendarización de las tareas que se desarrollarán para la realización del proyecto con su fecha de inicio y de fin. En color rojo tenemos las fechas reales en las que se realizaron las tareas y podemos observar un desfase en los días empleados para la realización del proyecto debido a la presencia de algunos de los riesgos que habíamos identificado en la fase de planificación y que se explican en la siguiente sección.

Nombre de la tarea	Duración prevista	Duración real	Comienzo previsto	Fin previsto	Comienzo real	Fin real
Planificación	**6 días**	**8 días**	**08/05/23**	**15/05/23**	**08/05/23**	**17/05/23**
Fijación de objetivos	1 día	1 día	08/05/23	08/05/23	08/05/23	08/05/23
Planificación del proyecto	5 días	7 días	09/05/23	15/05/23	09/05/23	17/05/23
Adquisición de Información	**10 días**	**14 días**	**16/05/23**	**29/05/23**	**18/05/23**	**06/06/23**
Conocimientos edificio LUCIA	3 días	3 días	16/05/23	18/05/23	18/05/23	22/05/23
Conocimientos SGBD y Docker	5 días	9 días	19/05/23	25/05/23	23/05/23	02/06/23
Conocimientos entorno web	2 días	2 días	26/05/23	29/05/23	05/06/23	06/06/23
Análisis y Diseño BDs	**9 días**	**22 días**	**30/05/23**	**09/06/23**	**07/06/23**	**14/07/23**
Estudio y limpieza de datos	3 días	7 días	30/05/23	01/06/23	07/06/23	23/06/23
Diseño pruebas rendimiento	1 día	2 días	02/06/23	02/06/23	26/06/23	27/06/23
Diseño TimescaleDB	1 día	5 días	05/06/23	05/06/23	28/06/23	04/07/23
Diseño InfluxDB	1 día	3 días	06/06/23	06/06/23	05/07/23	07/07/23
Diseño OpenTSDB	1 día	3 días	07/06/23	07/06/23	10/07/23	12/07/23
Diseño KairosDB	1 día	1 día	08/06/23	08/06/23	13/07/23	13/07/23
Diseño estructura de datos	1 día	1 día	09/06/23	09/06/23	14/07/23	14/07/23
Implementación BDs	**12 días**	**20 días**	**12/06/23**	**27/06/23**	**17/07/23**	**06/10/23**
Implementación TimescaleDB	3 días	7 días	12/06/23	14/06/23	17/07/23	07/08/23
Implementación InfluxDB	3 días	4 días	15/06/23	19/06/23	08/08/23	24/08/23
Implementación OpenTSDB	3 días	6 días	20/06/23	22/06/23	25/08/23	12/09/23
Implementación KairosDB	3 días	3 días	23/06/23	27/06/23	13/09/23	06/10/23

Ejecución pruebas rendimiento y validación	**1 día**	**2 días**	**28/06/23**	**28/06/23**	**09/10/23**	**10/10/23**
Diseño herramienta web	**5 días**	**9 días**	**29/06/23**	**05/07/23**	**11/10/23**	**23/10/23**
Implementación herramienta web	**5 días**	**17 días**	**06/07/23**	**12/07/23**	**24/10/23**	**06/12/23**
Pruebas validación herramienta web	**1 día**	**4 días**	**13/07/23**	**13/07/23**	**07/12/23**	**15/12/23**
Elaboración del manual del instalador	**1 día**	**2 días**	**14/07/23**	**14/07/23**	**18/12/23**	**22/12/23**
Análisis resultados de rendimiento	**3 días**	**7 días**	**17/07/23**	**19/07/23**	**26/12/23**	**23/01/24**
Elaboración de la memoria	**-**	**-**	**08/05/23**	**19/07/23**	**08/05/23**	**09/02/24**
Reuniones con el tutor	**-**	**-**	**08/05/23**	**19/07/23**	**08/05/23**	**09/02/24**
Total:	*53 días*	*105 días*			*08/05/23*	*09/02/24*

Tabla 2.1: Seguimiento del plan de trabajo

2.4. Gestión de riesgos

La gestión de los riesgos del proyecto tiene como objetivo identificar los problemas que se puedan presentar durante el desarrollo del proyecto y establecer un plan de mitigación para reducir el impacto negativo de estos y el aumento de los costes del proyecto.

Una vez identificados los posibles riesgos que pueden surgir elaboraremos una tabla para cada uno con la siguiente información: breve descripción del riesgo identificado; probabilidad de ocurrencia del riesgo; impacto o grado de importancia del riesgo en una escala relativa con los valores *'catastrófico'*, *'crítico'*, *'marginal'* y *'despreciable'*; plan de mitigación o medidas de prevención del riesgo; y plan de contingencia o medidas de acción para reducir el impacto en caso de ocurrencia del riesgo.

R-1	**Fallos en la planificación**
Descripción	Los tiempos establecidos en la planificación no es suficiente para el desarrollo de tareas y no se cumplen los plazos en la calendarización
Probabilidad	60 %
Impacto	Crítico
Plan de mitigación	Analizar las tareas de forma adecuada para no planificar tiempos que sean insuficientes. Desarrollar el trabajo en los tiempos establecidos
Plan de contingencia	Calcular la prioridad de las tareas a desarrollar, establecer consecuencias, reevaluar los riesgos y replanificar la calendarización de tareas

Tabla 2.2: R-1: Fallos en la planificación

R-2	Problemas con herramientas de desarrollo
Descripción	Las herramientas de software y hardware que utilizaremos para el desarrollo del proyecto no se encuentran disponibles o presentan fallos
Probabilidad	20 %
Impacto	Crítico
Plan de mitigación	Realizar un control de la disponibilidad de los recursos que se utilizan, tener herramientas alternativas que poder utilizar temporalmente o de sustitución
Plan de contingencia	Adelantar la realización de otras tareas paralelas que no dependan de las herramientas y en caso de que no se pueda replanificar la calendarización

Tabla 2.3: R-2: Problemas con herramientas de desarrollo

R-3	Indisponibilidad del desarrollador
Descripción	Por motivos personales o médicos del desarrollador no se pueden realizar las tareas planificadas
Probabilidad	40 %
Impacto	Crítico
Plan de mitigación	Realizar un control de la disponibilidad de los recursos que se utilizan, tener herramientas alternativas que poder utilizar temporalmente o de sustitución
Plan de contingencia	Adelantar la realización de otras tareas paralelas que no dependan de las herramientas y en caso de que no se pueda replanificar la calendarización

Tabla 2.4: R-3: Indisponibilidad del desarrollador

R-4	Conocimiento insuficiente sobre las tecnologías
Descripción	Al planificar las tareas de adquisición de conocimientos y aprendizaje de herramientas tecnológicas se ha establecido un tiempo inferior al necesario
Probabilidad	60 %
Impacto	Crítico
Plan de mitigación	Realizar un estudio exhaustivo de las tecnologías que se van a utilizar y analizar los conocimientos que tenemos acerca de ellas para ser más precisos con el tiempo asignado en la planificación
Plan de contingencia	Calcular la prioridad de las tareas a desarrollar, reevaluar los riesgos y replanificar la calendarización dedicando más tiempo a estas tareas

Tabla 2.5: R-4: Conocimiento insuficiente sobre las tecnologías

2.4.1. Seguimiento de riesgos

Durante el desarrollo del proyecto se han producido los siguientes riesgos:

- R-1 Fallos en la planificación: los tiempos estimados para las tareas en ocasiones no han sido suficientes y se han tenido que ampliar.
- R-2 Problemas con herramientas de desarrollo: los recursos del equipo personal en el que se desarrollaba el trabajo no era suficiente para la ejecución de los programas realizados a medida que estos crecían por lo que se ha necesitado solicitar recursos más potentes a la universidad.

- R-3 Indisponibilidad del desarrollador: se han presentado periodos de tiempo en los que el desarrollador no ha estado disponible por problemas personales y de salud.

Estos riesgos como puede verse de color rojo en la Tabla 2.1 han supuesto cambios en la planificación de las tareas provocando un desfase entre el tiempo estimado para la elaboración del proyecto y el tiempo realmente empleado. Se han llevado a cabo los planes de contingencia establecidos pero las consecuencias de estos riesgos han supuesto un retraso en la entrega del proyecto. En particular, el análisis e implementación de las bases de datos al tratarse de tecnologías que no se habían visto durante el grado y la implementación de la herramienta web por la misma razón.

2.5. Costes del proyecto

La estimación de los costes supone desarrollar una estimación de los costes de los recursos precisos para llevar a cabo las actividades planificadas. Realizaremos una estimación reducida sin tener en cuenta los costes de esfuerzo o aquellos que son indirectos aplicados al personal del proyecto y que habría que tener en cuenta en la gestión de los costes de proyectos más grandes. En la Tabla 2.6 recogemos simplemente los costes del hardware y software que utilizaremos para la realización del trabajo propuesto.

Herramienta	**Coste estimado**	**Coste real**
Ordenador personal	699 €	699 €
Máquina virtual de la universidad	1569.36 €	Suponemos un coste de 0 €ya que habrá sido amortizado por la universidad para otros proyectos
GanttProject	0 €	0 €
Sublime Text	91.90 €	Se puede utilizar la versión de prueba si no te registras: 0 €
Overleaf	0 €	En 2024 necesitamos la versión de pago para que compile el proyecto: *2 meses* x 9.68 €= 19.36 €
Jupyter Notebook	0 €	0 €
Docker	0 €	0 €
Bases de datos	0 €	0 €
Flask	0 €	0 €
Draw.io	0 €	0 €
GitHub	0 €	0 €
Total:	***718.36 €***	

Tabla 2.6: Estimación de costes del proyecto

Todas las herramientas que nos planteamos utilizar son de software libre o como *'Sublime-Text'* que se puede utilizar de manera gratuita. El único gasto que surge en la etapa final del proyecto y que no se había contemplado es la cuota de pago de *'Overleaf'* que es la plataforma en línea en la que se encuentra la memoria del proyecto.

Capítulo 3

Caso de estudio

3.1. Edificio LUCIA

En el año 2013 se construyó el edificio LUCIA en el Campus Miguel Delibes de la Universidad de Valladolid para acoger laboratorios y centros de investigación científica. Proyecto diseñado por el arquitecto Francisco Valbuena García con el objetivo de conseguir una infraestructura sostenible que garantizase y promoviese la eficiencia energética.

En la Figura 3.1 podemos observar algunas de las características más significativas del edificio como son[57]:

- Fachada e iluminación: el edificio presenta un diseño de sus fachadas en forma de 'zigzag' con el fin de reducir las cargas de refrigeración en verano aprovechando el autosombreamiento y produciendo ganancias térmicas en invierno. En cuanto a la iluminación de la estructura se combinan la utilización de pozos de luz y lucernarios, con esto se consigue reducir la demanda eléctrica de iluminación artificial.
- Aparcamiento descubierto: se opta por este diseño de cara a la obtención de ventilación e iluminación naturales reduciendo así el consumo de aparatos de refrigeración. Se emplea el uso de un pavimento filtrante como parte del sistema de geotermia.
- Energía nula: la energía utilizada proviene del propio edificio mediante la utilización de energías exclusivamente renovables como son la fotovoltaica, geotérmica (construcción de un pozo canadiense que brinda la posibilidad de ajustar la temperatura de las instalaciones sin consumir energía eléctrica) y biomasa para cubrir todas las necesidades. Además, gracias a esto es posible el autoabastecimiento y la cesión de energía a otros edificios del campus.
- Vegetación: se incluye el uso de plantas y especies arbóreas de hoja caduca autóctonas, además de una cubierta de techo ajardinada con plantas del género "Sedum" que ocupan el 73,5 % de la superficie, favoreciendo así el efecto isla de calor en la edificación y la generación de microclimas.
- Recuperación de agua y gestión de residuos: la utilización de una cubierta vegetal posibilita recuperar el 100 % del agua de lluvia y a su vez, se realiza un proceso de reciclado

de aguas grises y un tratamiento del agua de los laboratorios ubicados en el edificio. En cuanto a la gestión de residuos, se ha realizado un estudio en la fase de construcción, utilización y demolición del edificio de cara a la recogida y reutilización de estos, por ejemplo mediante el reciclaje o la creación de compost.

(a) Fachada en forma dentada (b) Aparcamiento abierto

Figura 3.1: Edificio LUCIA (Fuente: blog LUCIA)

Un sistema de control centralizado se ocupa de gestionar de la forma más eficiente posible las instalaciones del edificio; para la monitorización de los dispositivos se utiliza un software de Siemens[43] denominado *Desigo Insight*[1]. Esta herramienta permite disponer de un sistema de control centralizado capaz de examinar la integración desplegada por el edificio, así como controlar todas las fuentes de energía con un único sistema y hacer posible la comunicación de datos a través de diversas tecnologías. Este software consiste en un sistema SCADA (Supervisión, Control y Adquisición de Datos), utilizado tradicionalmente para la gestión de instalaciones complejas permitiendo un control automático y retroalimentación en tiempo real de los datos recogidos por sensores y actuadores. En la Figura 3.2 podemos ver el aspecto de su interfaz.

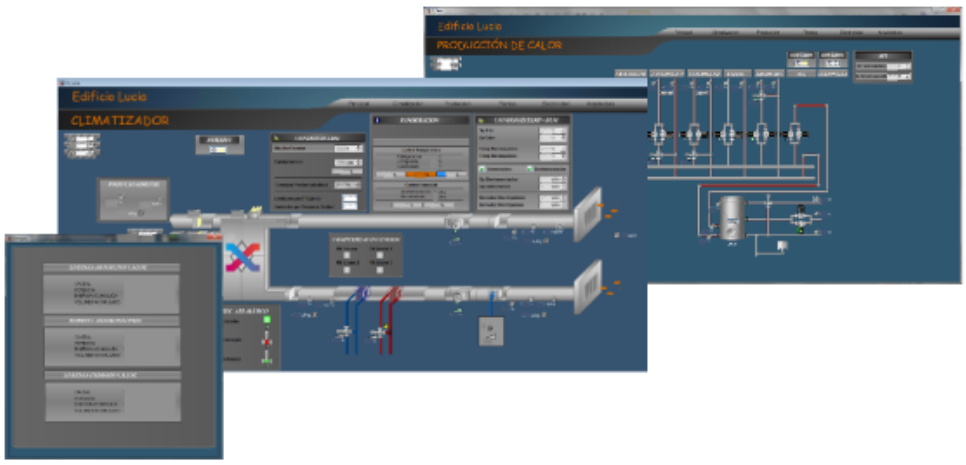


Figura 3.2: Capturas de pantalla del sistema de monitorización (Fuente: SCADA Siemens)

[1]Desigo® es una marca registrada de Siemens Building Technologies Ltd

Las instalaciones cuentan con contadores de energía integrados mediante protocolo M-bus (Meter Bus), contadores de agua mediante pulsos, analizadores de red integrados mediante protocolo Modbus, ventiloconvectores y un sistema para el alumbrado integrado con el protocolo KNX. Además, cuenta con medidores de estado y alarma para el sistema de electricidad, así como controladores mediante impulsos para compuertas y lucernarios[57].

Dentro del sistema de control del edificio encontramos una serie de controladores repartidos por las cuatro plantas que se comunican con un controlador general, situado en el sótano, desde el que se gestiona la instalación. Los equipos que se encuentran conectados a estos controladores se comunican mediante diferentes protocolos. Otra parte importante de la instalación son los más de noventa y cinco analizadores de redes eléctricas, cuyo objetivo es el de medir la potencia eléctrica y otros parámetros como pueden ser tensiones, corrientes de cada fase, etc.

Por último, mencionar que hay un contador del consumo energético térmico para todo el sistema. Las razones por las que únicamente se cuenta con un dispositivo para esta medición en lugar de tener varios distribuidos por las distintas plantas que componen el edificio son tres: instrumento de medida con un coste económico elevado, posibilidad de obtener datos inválidos porque se den ciertas circunstancias y el bajo consumo que supone frente al consumo energético debido al gran aislamiento y al destacable ratio térmico del edificio.

A continuación se describen de forma breve los principales sistemas que conforman el edificio y que pueden ser controlados desde el SCADA por personal cualificado[57].

- Climatizador de aire primario
  El climatizador se encarga de la limpieza del aire en retorno y del control, en función de sondas, de la temperatura y la humedad de éste. El edificio cuenta con fan coils de cuatro tubos repartidos por los distintos despachos y laboratorios, estos son equipos agua-aire formados por un intercambiador de frío o calor y un ventilador. EL climatizador está compuesto por un control de presión, un control de temperatura, un control de humedad, sistema de recuperación, alarmas y seguridad ante incendios.

- Distribución de agua
  El edificio dispone de cuatro circuitos de distribución de agua fría y seis circuitos de distribución de agua caliente que arrancan en función de la demanda de energía de los mismos y de la que se encuentre disponible en producción.

- Producción de agua
  A la hora de producir frío, una máquina de absorción y una enfriadora se encargan de ello. La máquina de absorción será la primera opción a la hora de producir frío y en caso de que no sea suficiente o haya algún inconveniente se recurrirá a poner en funcionamiento la enfriadora.

- Contadores
  El edificio cuenta con cinco contadores de energía que se integran en el sistema mediante protocolo M-bus, y se encargan de gestionar el calor de absorción, el frío de absorción, el calor primario, el ACS (agua caliente sanitaria) y la enfriadora. En el SCADA podremos visualizar información acerca del caudal, potencia, energía acumulada y volumen acumulado. Además, dispone de contadores de agua por pulsos del ACS, climatización, general, incendios, reciclada y torre de refrigeración. Además, los analizadores de red integrados

mediante protocolo M-bus (Meter Bus) nos permiten observar en el sistema señales significativas como son la energía activa, frecuencia, factor de potencia e intensidades, tensiones y potencia de fase.

- Unidades terminales
  Encontramos diferentes unidades terminales, ubicadas en cada dependencia del edificio, integradas en el sistema mediante protocolo KNX. Los fan coils de las instalaciones tienen una unidad ambiente desde la que se puede cambiar el valor de la temperatura, la velocidad del ventilador o el modo de funcionamiento. En el SCADA podemos visualizar el estado de las variables de cada dispositivo, pero al poder alterar la unidad ambiente puede que la orden representada en el sistema no sea la que se está dando en el dispositivo, ya que la última orden sobre el dispositivo es la que se impone. También podemos encontrar unidades que nos permitan modificar parámetros relacionados con la iluminación.

- Alumbrado
  En cuanto a la iluminación artificial del edificio, se recurre también a un sistema KNX, donde la última orden es la que se ejecuta y ésta se puede modificar desde el puesto central o desde la unidad ambiente dispuesta en cada habitación. El encendido del alumbrado en zonas comunes se realiza mediante sensores de presencia y en cuanto al apagado un programa de horario semanal se establece en el sistema para realizar los barridos de apagado.

- Producción de calor en el edificio anexo
  El edificio opta por la utilización de energías únicamente renovables. La climatización y la producción de energía eléctrica se llevan a cabo a través de la explotación de la biomasa, lo que supone promover la utilización de este recurso frente al uso de combustibles fósiles y una reducción de dióxido de carbono. Un edificio anexo presenta un cogenerador y una caldera de biomasa destinados a la producción de calor que se trasladará al edificio LUCIA.

Capítulo 4

Estado del arte

4.1. Introducción a las bases de datos

Las bases de datos han sido siempre uno de los métodos más utilizados a la hora de almacenar datos y con el paso del tiempo han evolucionado drásticamente desde un modelo de bases de datos jerárquicas e información estructurada en tablas a, en la actualidad, modelos de bases de datos no relacionales e información estructurada en formato de columnas individuales o directamente en forma de documentos.

Las bases de datos convencionales modelan el estado de los datos en un momento determinado, es decir, cuando estos dejan de ser ciertos se modifican y los datos nuevos sustituyen a los que había anteriormente que dejan de ser recuperables.

4.1.1. Historia de las bases de datos

Comenzaremos definiendo el concepto de base de datos como un conjunto de datos estructurados y almacenados en una unidad lógica. En un principio, antes de que llegasen las bases de datos a nuestras vidas se trabajaba con sistemas de ficheros los cuales surgieron al traspasar la información almacenada físicamente a dispositivos electrónicos, con el fin de ganar eficiencia a la hora de acceder a ella.

Con el paso del tiempo, la existencia de un conjunto de ficheros de datos, junto con sus respectivos programas de aplicación, presentaban diversos inconvenientes como son la inconsistencia de los datos al tener diversas copias repartidas por los dispositivos o la dependencia de los datos frente al soporte físico.

En la década de los años sesenta, el desarrollo de IDS (Integrated Data Store) por Charles Bachmann, reconocido como uno de los pioneros en los sistemas de bases de datos, supuso la creación de un nuevo tipo de sistemas de bases de datos conocido como sistema de red. Posteriormente, se crean diferentes grupos de expertos para poder establecer unos estándares acerca de las bases de datos. De esta forma se conforma la primera generación de SGBD (Data Base Management Systems) compuesta por los sistemas jerárquicos y de red.

Más tarde, en la década de los años setenta, gracias a Edgar Frank Codd aparece la segunda generación de los sistemas gestores de bases de datos, con su presentación del modelo relacional y solución de los inconvenientes de los sistemas existentes hasta el momento. Esto, daría paso al posterior desarrollo de un lenguaje estructurado de consultas denominado SQL (Structured Query Language) y la producción de varios SGBD relacionales, SQL/DS (Structured Query Language/Data System) y Oracle, durante la década de los años ochenta[27].

Ahora bien, todo avance realizado conlleva la aparición de inconvenientes, por lo que nos encontramos con la tercera generación de los SGBD al tener que lidiar con la capacidad limitada a la hora de modelar los datos que presenta el modelo relacional. Así, a partir de la década de los años noventa comienzan a aparecer nuevos tipos de bases de datos como las orientadas a objeto que se encargan de almacenar objetos completos formados por un estado y un comportamiento incorporando todos los conceptos del paradigma de objetos. Como resultado de la incapacidad de las bases de datos relacionales de responder a consultas recursivas o de deducir relaciones indirectas entre los datos surgen las bases de datos deductivas basadas en lógica matemática[36].

4.1.2. Evolución de las bases de datos

Poco a poco, con el desarrollo de la tecnología, las bases de datos comenzaron a tener una posición relevante en el mercado, siendo cada vez más necesarias herramientas para gestionar los nuevos enfoques de bases de datos que iban surgiendo y nuevas formas de incorporar datos de maneras diferentes.

Con la llegada de las redes sociales a nuestras vidas cuya evolución se puede ver en la Figura 4.1, el aumento de datos se hizo notable y apareció la necesidad de disponer de bases de datos escalables y de alto rendimiento para poder cumplir con las expectativas de las nuevas tecnologías. En la actualidad, 4760 millones de personas en todo el mundo utilizan redes sociales, es decir, más de la mitad de la población mundial. Dicho de otra manera, tenemos un 59,4 % de la población mundial como usuarios activos en las redes sociales[25].

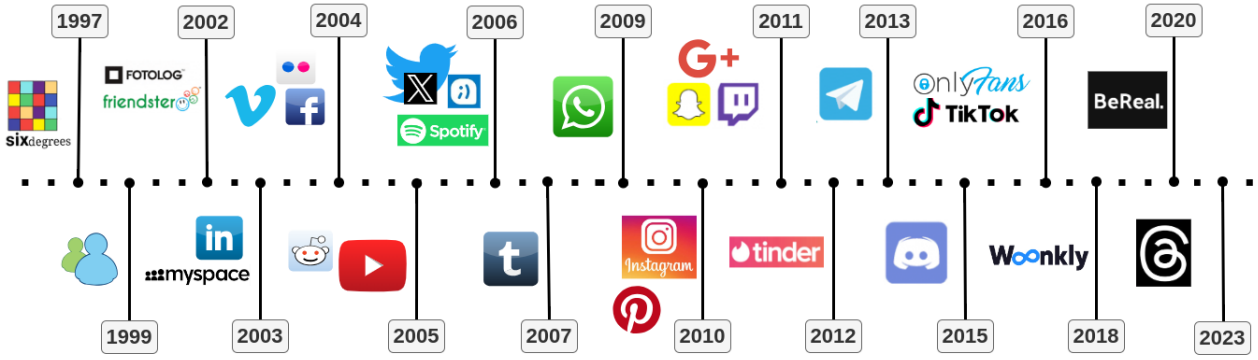


Figura 4.1: Gráfico de aparición de las redes sociales (Figura de elaboración propia)

En el siglo XXI, además de presenciar el surgimiento de las aplicaciones que nos conectan con el resto de personas a través de internet, también se lleva a cabo la conexión de las personas

con los objetos que les rodean, surgiendo así lo que conocemos como IoT (Internet of Things). Al mismo tiempo, la eficiencia energética de los edificios va ganando peso en la sociedad, ya que supone un ahorro económico y un impacto medioambiental considerable que se traduce en la transformación de edificios convencionales en edificios inteligentes mediante el uso de dispositivos electrónicos.

Como podemos observar en la Figura 4.2, 307,8 millones de hogares en Enero de 2023, a nivel mundial, contaban con dispositivos inteligentes, teniendo en cuenta para las estadísticas dispositivos domésticos conectados y controlados digitalmente, a distancia o no, como son sensores, actuadores y servicios en la nube.

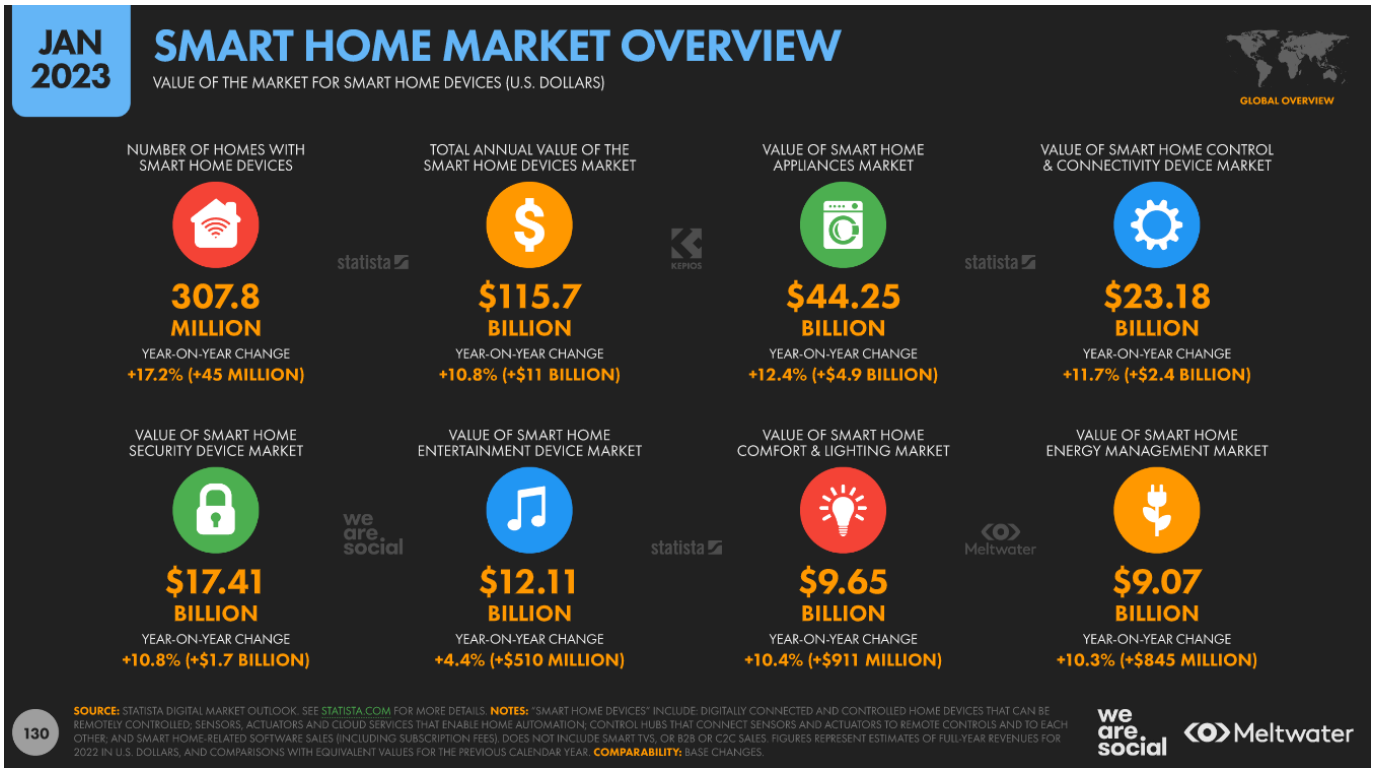


Figura 4.2: Mercado de dispositivos domésticos inteligentes (Fuente: Digital 2023)

Las cifras que se pueden ver en la imagen hacen referencia al gasto destinado, en dólares, a dispositivos de automatización en el año 2022. Podemos ver que, en comparación con el año natural anterior, hay un aumento superior al 10 % de inversión en cualquier tipo de uso de los dispositivos inteligentes[25].

Tal es la importancia de los datos en nuestro día a día que, en cifras, la cantidad total de datos que se prevé crear, obtener, copiar y consumir es de 181 zettabytes en 2025, a nivel mundial. Dicho de otra manera, estamos en un período en el que a medida que avanza nuestra tecnología el "Big Data"gana cada vez más poder. Estos grandes conjuntos de datos; los macrodatos a los que nos referíamos en la Sección 1, son fundamentales a la hora de obtener conocimiento y crear nuevas oportunidades, pero esto también supone un gran problema para la ciencia de datos a hora de gestionarlos[49].

Figura 4.3: Big Data características (Figura de elaboración propia)

Las características principales de los macrodatos se pueden resumir en las conocidas "5V's"[1]:

- Volumen: cantidad de datos generados y almacenados. Suele ser superior a terabytes y petabytes.
- Variedad: tipo y naturaleza de los datos.
- Velocidad: velocidad a la que se generan y procesan los datos. Diferenciando entre frecuencia de generación y frecuencia de manipulación, registro y publicación. En nuestro estudio esta será la V más importante.
- Veracidad: calidad de los datos. Los datos deben ser fiables para conseguir determinado valor al analizarlos.
- Valor: valor en información que puede obtenerse al procesar y analizar los datos.

Los macrodatos se generan a partir de múltiples fuentes como transacciones móviles, comercio electrónico, contenidos generados por usuarios en Internet, sensores inalámbricos, series temporales financieras, datos biométricos, etc. Como ya se ha comentado, la recolección, almacenamiento, consulta, compartición o análisis de los datos se vuelve más compleja a medida que crece el volumen de estos. Los retos que el Big Data nos presenta necesitan de nuevas ideas estadísticas, métodos computacionales, infraestructuras informáticas y métodos de almacenamiento de datos específicos.

Con la finalidad de cubrir las necesidades de los macrodatos surgen los grandes repositorios

de datos, también llamados "data warehouses", plataformas de procesamiento en paralelo, modelos de programación para computación paralela y distribuida sobre grandes conjuntos de datos, como el conocido "MapReduce", estructuras de código abierto para almacenar grandes volúmenes de datos y ejecutar aplicaciones distribuidas, como es el caso de Hadoop, o computación basada en la nube[10].

Por otro lado, centrándonos en la gestión y almacenamiento de los datos, comienzan a ganar peso las bases de datos NoSQL y se comienzan a realizar investigaciones sobre las bases de datos temporales y las bases de datos multimedia.

4.1.3. Clasificación de bases de datos

Podemos clasificar brevemente los diferentes tipos de bases de datos de la siguiente forma[35]:

- En función de su variabilidad:
  - **Estáticas**: únicamente de lectura. Registro de datos históricos para futuros estudios.
  - **Dinámicas**: la información se modifica con el tiempo. Funciones constantes de actualización, edición y eliminación de datos.
- En función de su contenido:
  - **Bibliográficas**: clasificación de diversos campos de datos. Consultar campos de forma separada o conjunta pero no se consigue todos los datos al completo.
  - **De texto completo**: todas las funciones de las bibliográficas y la posibilidad de buscar términos concretos. Se puede consultar toda la información de forma conjunta.
  - **Directorios**: formadas por elementos básicos con el fin de ordenar y organizar la información almacenada.
- En función de su modelo[18]:
  - **Jerárquicas**: se almacena la información siguiendo un orden de importancia. Árbol invertido formado por nodos y ramas.
  - **De red**: siguen el mismo modelo que las jerárquicas pero cambiando el concepto de nodo. Un nodo puede tener varios padres.
  - **Orientadas a objetos**: almacena objetos como tal, formados por unas características concretas que las diferencian del resto.
  - **Relacionales**: información estructurada en tablas formadas por registros (filas) y campos (columnas). SQL es el lenguaje predominante de este tipo de bases de datos.
  - **Multidimensionales**: diferentes a nivel conceptual con las relacionales. Los campos de una tabla pueden representar dimensiones de tabla o métricas que se desean aprender.
  - **Deductivas**: permite aplicar inferencia sobre los datos y obtener deducciones. Datalog es el lenguaje utilizado para resolver deducciones sobre las respuestas de las consultas.

    - **Orientada a grafos**: información almacenada en nodos de un grafo (entidades) y aristas (relaciones). Debe estar normalizada.

- Otros:

    - **En la nube**: colección de datos, estructurados o no, que se localizan en la nube privada, pública o una combinación. Dos tipos de modelos: tradicional y base de datos como servicios (DBaaS).
    - **Multimodelo**: combinan diferentes modelos de bases de datos en un único servidor integrado. Puede almacenar diferentes tipos de datos.
    - **No relacional (llamado NoSQL)**: base de datos no relacional que permite almacenar y manipular datos no estructurados y semiestructurados. En función de su modelo pueden ser orientadas a: clave-valor, columnas, documentos o a grafos.

Ahora bien, teniendo en cuenta que hay muchos tipos de bases de datos surge un dilema a la hora de elegir uno u otro. En el año 2000, Eric A.Brewer presentó una conjetura sobre sistemas distribuidos que dos años más tarde sería probado como el "Teorema de Brewer" o teorema CAP (Consistency, Availability, Partition Tolerance)[14]. Este teorema enuncia la imposibilidad de garantizar simultáneamente en un sistema distribuido, red que almacena datos en más de un nodo al mismo tiempo, las siguientes tres características:

- Consistencia: cualquier petición de lectura de datos en un sistema distribuido con nodos replicados recibe como respuesta el estado actual, independientemente del nodo desde el que se acceda a la base de datos.
- Disponibilidad: cualquier usuario que realice una solicitud de datos obtendrá respuesta, incluso si uno o mas nodos no se encuentran activos.
- Particionamiento: se define como partición una conexión perdida o con retardo entre nodos. La tolerancia al particionamiento significa que el sistema seguirá trabajando a pesar de que se produzca una interrupción en la comunicación.

El teorema CAP es relevante a la hora de diseñar aplicaciones distribuidas y elegir un almacén de datos relacional o no relacional. Los sistemas de bases de datos relacionales (RDBMS) eligen la consistencia sobre la disponibilidad ya que siguen el enfoque ACID (Atomicity, Consistency, Isolation, Durability), mientras que las bases de datos no relacionales, que siguen el enfoque BASE (Basically Available, Soft state, Eventually consistent), se definen por las características CAP que engloban lo siguiente[1]:

- Bases de datos CA: bases de datos que presentan consistencia y disponibilidad pero no son tolerantes al particionamiento. Es la opción menos escalable ya que no tiene la capacidad de acabar con los errores y esto supondría dejar inoperativo el sistema si ocurriese un error.
- Bases de datos CP: bases de datos que presentan consistencia y son tolerantes al particionamiento pero carecen de disponibilidad. Cuando se produce un error el sistema cierra el nodo, arrebatándole su disponibilidad.

- Bases de datos AP: bases de datos que presentan disponibilidad y son tolerantes al particionamiento pero carecen de consistencia. Todos los nodos continúan estando disponibles para la comunicación pero la información puede carecer de fiabilidad o ser antigua.

Este teorema nos permite tener una base al conocer los tres atributos fundamentales a la hora de elegir una infraestructura y una base de datos que se ajuste a las necesidades que tengamos.

4.2. SQL vs. NoSQL

Las bases de datos relacionales, también llamadas bases de datos SQL, requieren la especificación explícita de tablas, atributos, dominios, claves y otras restricciones referentes a la integridad que se definen mediante el esquema de la base de datos. Por el contrario, la mayoría de los sistemas NoSQL no tienen un esquema de base de datos explícito ya que los cambios sobre los datos semiestructurados o no estructurados pueden darse en cualquier momento.

Como podemos ver en la Figura 4.4 las bases de datos no relacionales utilizan una arquitectura de almacenamiento distribuido, almacenándose la información en pares clave-valor, columnas o familias de estas, almacenes de documentos o grafos[1].

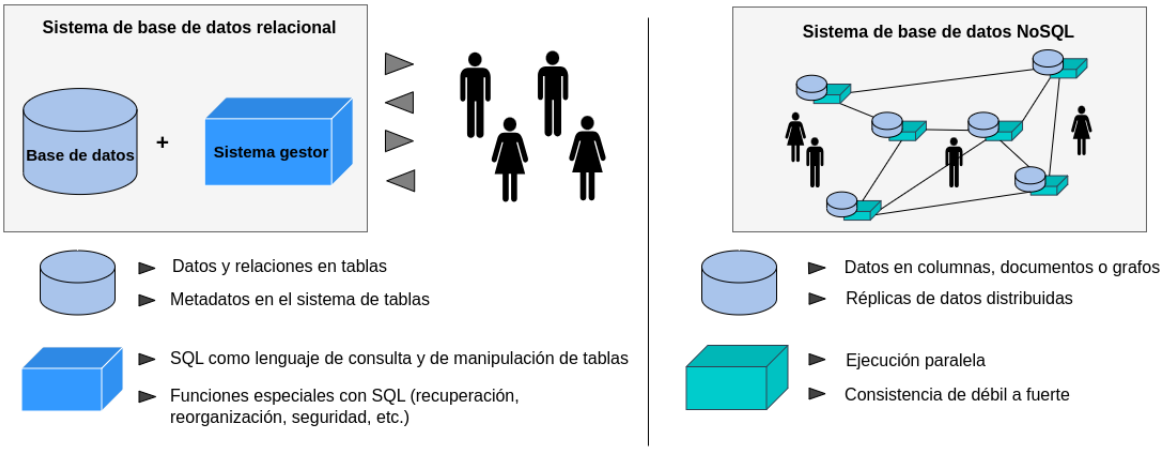


Figura 4.4: Estructura básica de los SGBD relacionales y NoSQL (Fuente: capítulo 1. Data management[1])

Podemos ver en la Tabla 4.1 resumidas las diferencias principales entre las bases de datos relacionales y las no relacionales. Las bases de datos que siguen el modelo relacional son las más conocidas y las que más han evolucionado desde sus inicios, al garantizar las propiedades de atomicidad, consistencia, aislamiento y durabilidad del modelo ACID se posicionan como la forma más robusta para almacenar información. El problema es que en la actualidad, la cualidad que las diferencia positivamente lo hace también al contrario, haciendo complejo el proceso de modificación a la hora de ampliar el almacenamiento o el número de usuarios de la base de datos. Por otro lado, tenemos las bases de datos no relacionales que rompen con la robustez del modelo relacional y se presentan como opciones de almacenamiento mucho más flexibles y adaptables a cambios.

	SQL	NoSQL
Almacenamiento	Datos almacenados en tablas y que necesitan un esquema predefinido	Datos almacenados en bruto
Escalabilidad	Escalabilidad vertical, invirtiendo en hardware más potente	Alta escalabilidad horizontal ya que se orientan al manejo de grandes volúmenes de datos
Adaptación a cambios	Puede resultar compleja debido a la estructura de los datos	Los cambios en la arquitectura no generan problemas ya que los datos con poca o ninguna estructura son más flexibles
Estandarización	Base de datos homogénea, los datos son del mismo tipo	Datos de diferentes fuentes sin estandarizar
Integridad de datos	Se caracterizan por la consistencia de los datos, presentan restricciones de integridad que los datos almacenados deben satisfacer	La integridad puede verse comprometida, buscan agilidad y rapidez en el acceso a los datos
Modelo	ACID	BASE

Tabla 4.1: Diferencias entre SQL y NoSQL

Dadas las fortalezas y debilidades de los dos modelos, así como la importancia de los datos en la actualidad, en este trabajo se realizará una comparación de las prestaciones que nos ofrecen distintas bases de datos, relacionales y no relacionales, a la hora de almacenar datos ligados a la variable tiempo.

4.3. Bases de datos temporales

Durante este proceso de evolución enfocado hacia el almacenamiento más eficiente de la información, surgió el problema de añadir una marca temporal a todos los datos que se obtenían en determinado momento, por lo que nacieron las bases de datos temporales. Hoy en día podemos disponer de una cantidad enorme de dispositivos conectados; sin ir más lejos; en un edificio monitorizado tenemos sensores capaces de tomar medidas en intervalos de tiempo programados.

Una base de datos temporal almacena datos relacionados con instancias de tiempo, permitiendo tener constancia de la información que se ha recogido desde el pasado hasta el presente. Un sistema de gestión de base de datos temporal (TDBMS) admite el eje de tiempo y contiene elementos de lenguaje temporal para poder realizar consultas. En el campo de este tipo de bases de datos, existen varios modelos de lenguaje que facilitan el tratamiento de información ligada al tiempo, aunque no muchos sistemas soportan estos conceptos temporales en la actualidad[1].

Richard Thomas Snodgrass es un informático estadounidense y uno de los principales contribuyentes al desarrollo del modelo temporal de bases de datos. Sus trabajos más reconocidos están relacionados con la optimización y evaluación de consultas, el diseño de bases de datos y el diseño de lenguajes de consulta. Entre los numerosos proyectos en los que está involucrado Snodgrass encontramos TAU (Temporal Access for Users); un conjunto de trabajos cuyo objetivo es proporcionar a los usuarios facilidades a la hora de gestionar datos orientados a tiempo mediante APIs (Application Programming Interfaces) y lenguajes de usuario[30]. Además, merece la pena destacar que es el codirector de TimeCenter, un centro internacional enfocado a dar soporte a aplicaciones de bases de datos temporales en tecnologías DBMS (DataBase Management System) tradicionales y emergentes[29].

Merece la pena mencionar algunas investigaciones y proyectos llevados a cabo por Snodgrass y otras personas relevantes de su entorno en relación con el aspecto temporal de los datos para los cuales no hemos encontrado cabida en este proyecto pero que son realmente interesantes. Por ejemplo, el desarrollo de un marco de referencia llamado τ Bench[51] para evaluar esquemas temporales y lenguajes de consulta sobre datos XML (Extensible Markup Language) y datos relacionales; la realización de trabajos acerca del diseño conceptual, lógico y físico de bases de datos temporales y la propuesta de soluciones avanzadas para la gestión de datos temporales[23]; y el desarrollo de dos nuevos lenguajes de consulta para datos temporales denominados TQuel (Temporal QUEry Language)[44], que surge a partir de un lenguaje de consulta para un sistema gestor de base de datos Ingres, y ATSQL2, basado en SQL con soporte para datos temporales.

Además, a partir de este último lenguaje de consulta mencionado Andreas Steiner, doctorado en Ciencias Tecnológicas, en su tesis doctoral propone un enfoque general para definir y aplicar modelos de datos que trabajen con datos dimensionales (tiempo, datos espaciales, etc.) e implementa un sistema de base de datos relacional bitemporal escrito en SICStus Prolog (en su primera versión, seguido de otra basada en Java con nuevas funcionalidades añadidas) y que utiliza ATSQL2 como lenguaje de consulta[47].

4.3.1. Estudio de Bases de datos de series temporales

Las bases de datos de series temporales contienen datos que varían con el tiempo y ofrecen un soporte que nos permite modelar la dimensión temporal de los datos. Definimos una serie temporal como una colección de observaciones de un suceso a lo largo del tiempo. El creciente uso de series temporales es la consecuencia de que las nuevas tecnologías cada vez produzcan más datos de este tipo. El motivo: los datos recogidos en un intervalo de tiempo son más útiles que una medición aislada en un momento específico ya que nos permiten saber el orden en el que ocurrieron determinados acontecimientos o los cambios que se sucedieron a lo largo de un periodo de tiempo[7].

Tal es la importancia de este tema, que hoy en día las tareas que despiertan más interés en el estudio de datos de series temporales son los siguientes[9]:

- **Consultas por contenido**
  El área de investigación más activa en la actualidad y aquella en la que se enfoca este trabajo es la recuperación de un conjunto de datos en respuesta a una consulta presentada por el usuario en una base de datos de series temporales. Otras de las cuestiones en las que se centran estas consultas son el manejo de la escala, el ruido o la falta de los datos.

- **Agrupamiento**
  Es el proceso destinado a encontrar grupos naturales, llamados clusters, en un conjunto de datos. El objetivo es encontrar un conjunto de series temporales que presenten el mayor número de similitudes entre ellas pero a su vez se diferencien del resto de series temporales que no pertenecen al cluster. Puede estar enfocado a agrupar series temporales completas o subsecuencias dentro de estas.

- **Clasificación**
  Esta tarea tiene como objetivo encontrar los rasgos que distinguen unas clases de otras y basándose en ello asignar etiquetas a cada serie temporal dentro de un conjunto.

- **Segmentación**
  Esta tarea está dirigida a reducir la dimensionalidad de las series temporales manteniendo sus características fundamentales, creando una aproximación precisa entre ellas. El objetivo está basado en reducir el error de reconstrucción entre la representación reducida y la serie temporal original.

- **Predicción**
  Otra de las áreas más importantes en relación con las series temporales es el pronóstico de valores futuros de la serie en base a la modelización de explícita de las dependencias entre los datos pasados y los presentes.

- **Detección de anomalías**
  El propósito de la detección de anomalías consiste en hallar subsecuencias extrañas dentro de una serie, utilizado por ejemplo a la hora de determinar si el flujos de datos procedente de una fuente incluye alguna medición anómala.

- **Descubrimiento de subsecuencias**
  Estas subsecuencias se caracterizan por denominarse "motivos" y hacer referencia a una subsecuencia típica no solapada que aparecen de manera recurrente en una serie temporal larga. Esta idea surge del análisis de genes en bioinformática.

Con lo que hemos expuesto hasta ahora, podemos ver que las series temporales tienen gran relevancia y uso en la actualidad por lo que los investigadores han dirigido sus esfuerzos a obtener un almacenamiento eficiente para este tipo de datos.

Las bases de datos temporales almacenan una serie de datos que poseen ciertas características[48]:

- En el proceso de creación de datos, estos tienden a ser inmutables ya que, por lo general, las mediciones generan nuevos datos que son añadidos al conjunto de datos original sin modificar las mediciones anteriores.

- Un conjunto de datos ligado a marcas de tiempo aumenta de volumen de manera muy rápida y esto supone un desafío de cara al tratamiento y almacenamiento de estos.

- El concepto de datos formado por o que se consideran series temporales surge de la necesidad de tener un historial a lo largo del tiempo sobre el que poder realizar un seguimiento, pero no todos los datos tienen la misma importancia del conjunto. La información que podemos obtener de los datos recientes es más relevante que la que podemos conseguir de datos antiguos, de la misma manera, los datos de un periodo corto y reciente, son más fácilmente procesables que los datos acumulados desde hace diez años.

En la actualidad existen diversas bases de datos para almacenar y tratar datos ligados al tiempo y bases de datos temporales, especializadas ya en ello, de código abierto y de licencia privativa. Para la realización del trabajo de fin de grado se ha optado por la utilización de herramientas de código abierto siempre que sea posible, ya que existe la posibilidad de experimentar con diferentes bases de datos sin que eso repercuta en el coste económico del proyecto.

Capítulo 5

Datos

En este proyecto trabajaremos con conjuntos de datos reales del mismo dominio formados por mediciones de energía consumida y el momento en el que se realizaron. En lo que respecta al proceso de extracción de datos cabe destacar que el software DESIGO INSIGHT es muy cerrado a la hora de poder exportar los datos fuera del sistema. Se tiene constancia de que existen datos históricos en formato *'.art'* pero no se ha conseguido encontrar una forma de acceder a los datos en dicho formato, más allá del visualizador de tendencias de la aplicación, por lo que la forma en la que se han extraído los datos es mediante una aplicación que realiza copias instantáneas de los datos por un rango de fecha dado, obteniendo de esta forma los registros en el espacio temporal de un mes para las variables que se desean estudiar en archivos Excel[28].

Partiremos de un conjunto de archivos en formato *'xlsx'* que contienen la recolección de lecturas de los sensores del área de integraciones del LUCIA. Trataremos datos de un total de trece meses (11 meses del año 2019 y 2 meses del año 2020) que tendremos que estudiar, limpiar y dar formato para almacenar en las diferentes bases de datos.

5.1. Descripción de los datos

Los datos en crudo o *raw data* son aquellos que no han sido procesados y que deberemos analizar, limpiar y estructurar para su posterior almacenamiento. Estos datos en crudo estarán formados por un conjunto de archivos que almacenan series temporales de las variables monitorizadas por el software DESIGO en el período de tiempo de un mes. Los archivos se dividen en distintas hojas que agrupan datos referentes a un mismo entorno y presentan una estructura en común como se muestra a continuación.

Como podemos ver en la Tabla 5.1 tenemos un grupo de datos ordenados cronológicamente con una estructura en forma de registro formado por una marcha de tiempo, una medición y un estado de medición.

Además, en la Tabla 5.2 podemos observar la estructura común que en su gran mayoría presentan los archivos de datos con los que contamos y que podemos desglosar de la siguiente manera:

- En la primera fila y cada cuatro columnas encontramos la dirección del sensor y el nombre asignado por el software DESIGO.
- En la segunda fila y cada cuatro columnas encontramos información del sensor como el lugar en el que se encuentra y el sistema al que pertenece entre otros datos.
- En la tercera fila encontraremos datos referentes a la monitorización del sensor que son:
  - En la primera columna se registra la fecha y hora de la medición del sensor en formato 'DD/MM/AA HH:MM:SS'.
  - En la segunda columna, cuya cabecera es la unidad de medida del sensor, se registra el valor leído.
  - En la tercera columna se guarda el estado y validez de la medición. Cabe destacar que algunos ficheros no cuentan con la columna *'tag_de_tipo'* y que será algo que haya que tener en cuenta al procesar los datos.
  - Los datos de las medidas están registrados a partir de la cuarta fila siguiendo este patrón.

ditech://LUCIA:AInt'PB'AnlzRed16'AI65,PrVal;AInt'PB'AnlzRed16'Trnd'TrdUFnct1		
LUCIA:Área Integraciones'Planta Baja'PBA5'Energia Activa,Valor actualValor actual; Función universal de tendencia		
Fecha/Hora	**kWh**	**Tag de tipo**
01/03/2019 03:07:39	-4	Bien, Hora cambiada
...	...	...
03/03/2019 15:44:41	507,2	Bien, Hora cambiada
03/03/2019 23:16:20	507,2	Bien, Hora cambiada
...	...	...

Tabla 5.1: Filas y columnas del fichero de datos en crudo

sensor_1			**sensor_2**			...		
información_1			**información_2**			...		
fecha y hora	**unidad_medida_1**	**tag_de_tipo_1**	**fecha y hora**	**unidad_medida_2**	**tag_de_tipo_2**	...		
fecha y hora	valor medido	estado medición	fecha y hora	valor medido	estado medición	...		
...	...	...	...	...	...	...		
fecha y hora	valor medido	estado medición	fecha y hora	valor medido	estado medición	...		
	valor resumen			valor resumen				valor resumen

Tabla 5.2: Estructura en común de los archivos de datos

A la hora de estudiar los datos y con el objetivo de tener una visión general de todos ellos realizamos un recuento de los sensores monitorizados, la unidad de medida en la que se toman los valores y su localización como podemos ver recogidos en la Tabla 5.3.

Identificador del sensor	Unidad	Información
AInt'PS'AnlzRed21'En1	MWh	'Planta Sótano'General Clima'
AInt'PS'AnlzRed22'AI65	kWh	'Planta Sótano'Enfriadora'
AInt'PS'AnlzRed23'AI65	kWh	'Planta Sótano'Absorción'

Sigue en la página siguiente.

Identificador del sensor	Unidad	Información
AInt'PS'AnlzRed24'AI65	kWh	'Planta Sótano'Torre'
AInt'PS'AnlzRed25'AI65	kWh	'Planta Sótano'Imp Cl Aire'
AInt'PS'AnlzRed26'AI65	kWh	'Planta Sótano'Ret Cl Aire'
AInt'PS'AnlzRed39'En1	MWh	'Planta Sótano'General Red'
AInt'PS'AnlzRed29'AI65	kWh	'Planta Sótano'P,Baja CPD CS-PB-CPD'
AInt'PS'AnlzRed34'AI65	kWh	'Planta Sótano'Fotovoltaica Lucernarios'
AInt'PS'AnlzRed41'En1	MWh	'Planta Sótano'General Fachada Sur'
AInt'PS'AnlzRed27'AI65	kWh	'Planta Sótano'P,Sót, Grupo CS-PS-S'
AInt'PS'AnlzRed36'AI65	kWh	'Planta Sótano'P,Sót, Red CS-PS-S'
AInt'PS'AnlzRed28'AI65	kWh	'Planta Sótano'P,Baja Grupo CS-PB'
AInt'PS'AnlzRed37'AI65	kWh	'Planta Sótano'P,Baja Red CS-PB'
AInt'PS'AnlzRed31'AI65	kWh	'Planta Sótano'P,1 Grupo CS-P1'
AInt'PS'AnlzRed38'AI65	kWh	'Planta Sótano'P,1 Red CS-P1'
AInt'PS'AnlzRed32'AI65	kWh	'Planta Sótano'P,2 Grupo CS-P2'
AInt'PS'AnlzRed35'AI65	kWh	'Planta Sótano'P,2 Red CS-P2'
AInt'PS'AnlzRed33'AI65	kWh	'Planta Sótano'Laboratorios P1'
AInt'PB'AnlzRed01'AI65	kWh	'Planta Baja'PBA1G'
AInt'PB'AnlzRed02'AI65	kWh	'Planta Baja'PBA2G'
AInt'PB'AnlzRed03'AI65	kWh	'Planta Baja'PBA3G'
AInt'PB'AnlzRed04'AI65	kWh	'Planta Baja'PBA4G'
AInt'PB'AnlzRed05'AI65	kWh	'Planta Baja'PBA5G'
AInt'PB'AnlzRed06'AI65	kWh	'Planta Baja'PBRACK'
AInt'PB'AnlzRed07'AI65	kWh	'Planta Baja'PBB1G'
AInt'PB'AnlzRed08'AI65	kWh	'Planta Baja'PBB2G'
AInt'PB'AnlzRed09'AI65	kWh	'Planta Baja'PBB3G'
AInt'PB'AnlzRed10'AI65	kWh	'Planta Baja'PBB4G'
AInt'PB'AnlzRed11'AI65	kWh	'Planta Baja'CF4A'
AInt'PB'AnlzRed12'AI65	kWh	'Planta Baja'PBA1'
AInt'PB'AnlzRed13'AI65	kWh	'Planta Baja'PBA2'
AInt'PB'AnlzRed14'AI65	kWh	'Planta Baja'PBA3'
AInt'PB'AnlzRed15'AI65	kWh	'Planta Baja'PBA4'
AInt'PB'AnlzRed16'AI65	kWh	'Planta Baja'PBA5'
AInt'PB'AnlzRed17'AI65	kWh	'Planta Baja'PBB1'
AInt'PB'AnlzRed18'AI65	kWh	'Planta Baja'PBB2'
AInt'PB'AnlzRed19'AI65	kWh	'Planta Baja'PBB3'
AInt'PB'AnlzRed20'AI65	kWh	'Planta Baja'PBB4'
AInt'PB'AnlzRed44'AI65	kWh	'Planta Baja'PBMVending'
AInt'P1'AnlzRed01'AI65	kWh	'Planta Primera'P1A1G'
AInt'P1'AnlzRed02'AI65	kWh	'Planta Primera'P1A2G'
AInt'P1'AnlzRed03'AI65	kWh	'Planta Primera'P1A3G'
AInt'P1'AnlzRed04'AI65	kWh	'Planta Primera'P1A4G'
AInt'P1'AnlzRed05'AI65	kWh	'Planta Primera'P1A5G'
AInt'P1'AnlzRed06'AI65	kWh	'Planta Primera'P1A6G'
AInt'P1'AnlzRed07'AI65	kWh	'Planta Primera'P1A7G'
AInt'P1'AnlzRed08'AI65	kWh	'Planta Primera'P1B1G'
AInt'P1'AnlzRed09'AI65	kWh	'Planta Primera'P1B2G'
AInt'P1'AnlzRed10'AI65	kWh	'Planta Primera'P1B3G'
AInt'P1'AnlzRed11'AI65	kWh	'Planta Primera'P1B4G'

Sigue en la página siguiente.

Identificador del sensor	Unidad	Información
AInt'P1'AnlzRed12'AI65	kWh	'Planta Primera'P1B5G'
AInt'P1'AnlzRed13'AI65	kWh	'Planta Primera'P1B6G'
AInt'P1'AnlzRed14'AI65	kWh	'Planta Primera'P1A1'
AInt'P1'AnlzRed15'AI65	kWh	'Planta Primera'P1A2'
AInt'P1'AnlzRed16'AI65	kWh	'Planta Primera'P1A3'
AInt'P1'AnlzRed17'AI65	kWh	'Planta Primera'P1A4'
AInt'P1'AnlzRed18'AI65	kWh	'Planta Primera'P1A5'
AInt'P1'AnlzRed19'AI65	kWh	'Planta Primera'P1A6'
AInt'P1'AnlzRed20'AI65	kWh	'Planta Primera'P1A7'
AInt'P1'AnlzRed21'AI65	kWh	'Planta Primera'P1B1'
AInt'P1'AnlzRed22'AI65	kWh	'Planta Primera'P1B2'
AInt'P1'AnlzRed23'AI65	kWh	'Planta Primera'P1B3'
AInt'P1'AnlzRed24'AI65	kWh	'Planta Primera'P1B4'
AInt'P1'AnlzRed25'AI65	kWh	'Planta Primera'P1B5'
AInt'P1'AnlzRed26'AI65	kWh	'Planta Primera'P1B6'
AInt'P2'AnlzRed01'AI65	kWh	'Planta Segunda'P2A1'
AInt'P2'AnlzRed02'AI65	kWh	'Planta Segunda'P2A2'
AInt'P2'AnlzRed03'AI65	kWh	'Planta Segunda'P2A3'
AInt'P2'AnlzRed04'AI65	kWh	'Planta Segunda'P2A4'
AInt'P2'AnlzRed05'AI65	kWh	'Planta Segunda'P2A6'
AInt'P2'AnlzRed06'AI65	kWh	'Planta Segunda'P2A7'
AInt'P2'AnlzRed07'AI65	kWh	'Planta Segunda'P2A8'
AInt'P2'AnlzRed08'AI65	kWh	'Planta Segunda'P2B1'
AInt'P2'AnlzRed09'AI65	kWh	'Planta Segunda'P2B2'
AInt'P2'AnlzRed10'AI65	kWh	'Planta Segunda'P2B3'
AInt'P2'AnlzRed11'AI65	kWh	'Planta Segunda'P2B4'
AInt'P2'AnlzRed12'AI65	kWh	'Planta Segunda'P2B5'
AInt'P2'AnlzRed13'AI65	kWh	'Planta Segunda'P2A5'
AInt'P2'AnlzRed14'AI65	kWh	'Planta Segunda'P2A1G'
AInt'P2'AnlzRed15'AI65	kWh	'Planta Segunda'P2A2G'
AInt'P2'AnlzRed16'AI65	kWh	'Planta Segunda'P2A3G'
AInt'P2'AnlzRed17'AI65	kWh	'Planta Segunda'P2A4G'
AInt'P2'AnlzRed18'AI65	kWh	'Planta Segunda'P2A5G'
AInt'P2'AnlzRed19'AI65	kWh	'Planta Segunda'P2A6G'
AInt'P2'AnlzRed20'AI65	kWh	'Planta Segunda'P2A8G'
AInt'P2'AnlzRed21'AI65	kWh	'Planta Segunda'P2B1G'
AInt'P2'AnlzRed22'AI65	kWh	'Planta Segunda'P2B2G'
AInt'P2'AnlzRed23'AI65	kWh	'Planta Segunda'P2B3G'
AInt'P2'AnlzRed24'AI65	kWh	'Planta Segunda'P2B4G'
AInt'P2'AnlzRed25'AI65	kWh	'Planta Segunda'P2B5G'
AInt'P2'AnlzRed26'AI65	kWh	'Planta Segunda'P2A7G'
AInt'P2'AnlzRed27'AI65	kWh	'Planta Segunda'P2Lab'
AInt'PS'AnlzRed23'PotT	Valor	'Planta Sótano'Absorción'

Tabla 5.3: Sensores monitorizados

Contamos con mediciones de 93 analizadores de red pertenecientes al área de integraciones del edificio (AInt). Estos se encargan de registrar datos de energía activa del edificio medidas en kWh (kilovatio hora) y MWh (megavatio hora). También tenemos un analizador de red, el

último que aparece en la Tabla 5.3, que mide el valor de potencia activa total y que no tiene una unidad de medida definida.

Para tener una visión general acerca de cómo son los datos decidimos estudiar dos meses, Julio y Octubre de 2019. Como vemos en la Tabla 5.4 se muestran estadísticas de los valores medidos por tres analizadores de red situados en la planta del sótano y dos analizadores de las otras plantas. Con esta información observamos que la frecuencia en la toma de medidas por los analizadores de red varía en el tiempo, pasamos de 1793 registros mensuales a 3099 para el analizador de red 21 del sótano, y varía de un sensor a otro, viendo que algunos realizan en el mismo mes 68 medidas y otros aumentan notablemente esta cifra. También podemos destacar que los analizadores de red de consumos generales, como son los que se muestran en las tres primeras filas de la tabla ('general clima', 'general red' y 'general fachada sur' respectivamente) son los que más mediciones toman, en comparación con sensores destinados a monitorizar variables en departamentos o laboratorios.

Variable	Mes	Nº de registros	Unidad de medida	Mínimo	Máximo	Media	Desviación estándar	Valores nulos
AInt'PS'AnlzRed21'En1	Julio 2019	1793	MWh	774.45	795.0	785.206	5.662	0
	Octubre 2019	3099	MWh	812.05	819.9	816.290	2.270	0
AInt'PS'AnlzRed39'En1	Julio 2019	1860	MWh	-2.0	2515.41	2486.954	60.153	0
	Octubre 2019	3099	MWh	2596.43	2641.22	2619.129	12.968	0
AInt'PS'AnlzRed41'En1	Julio 2019	1860	MWh	-2.0	52.83	52.354	1.287	0
	Octubre 2019	3099	MWh	58.43	55.66	55.324	0.237	0
AInt'PB'AnlzRed01'AI65	Julio 2019	79	kWh	-2.0	3418.9	3283.783	384.334	0
	Octubre 2019	153	kWh	3.0	3432.2	3407.548	277.093	0
AInt'PB'AnlzRed15'AI65	Julio 2019	79	kWh	-2.0	964.9	943.520	107.933	0
	Octubre 2019	153	kWh	3.0	1008.5	994.926	80.823	0
AInt'P1'AnlzRed10'AI65	Julio 2019	61	kWh	-2.0	0.0	-0.032	0.256	0
	Octubre 2019	153	kWh	3.0	105.6	84.016	14.127	0
AInt'P1'AnlzRed23'AI65	Julio 2019	73	kWh	-2.0	1851.3	1776.849	212.724	0
	Octubre 2019	153	kWh	3.0	1875.0	1856.928	150.891	0
AInt'P2'AnlzRed10'AI65	Julio 2019	61	kWh	-2.0	1522.6	1483.029	193.484	0
	Octubre 2019	153	kWh	3.0	1564.8	1544.516	125.592	0
AInt'P2'AnlzRed26'AI65	Julio 2019	68	kWh	-2.0	0.2	0.167	0.266	0
	Octubre 2019	153	kWh	0.2	3.0	0.2183	0.226	0

Tabla 5.4: Estadísticas de diferentes analizadores de red

Cabe destacar que las mediciones son acumuladas y que en lo que respecta a los datos de estos dos meses y nueve variables no existen nulos, además la alta desviación estándar de las diferentes variables y la presencia de valores negativos será algo a tener en cuenta a la hora de limpiar los datos.

Utilizamos gráficos de líneas para poder ver la tendencia temporal de los datos. En la Figura 5.1 podemos ver los datos de consumo energético en MWh del analizador de red 21 situado en la planta sótano del edificio. Como se puede ver es un consumo que aumenta a lo largo del tiempo ya que se trata de valores acumulados como ya habíamos mencionado anteriormente.

Representamos gráficamente el consumo energético medido en kWh para el analizador 27 de la segunda planta en la Figura 5.2 y vemos que los valores medidos a lo largo del tiempo tienen una fluctuación muy reducida salvo por las mediciones negativas que consideramos *outliers* o valores atípicos.

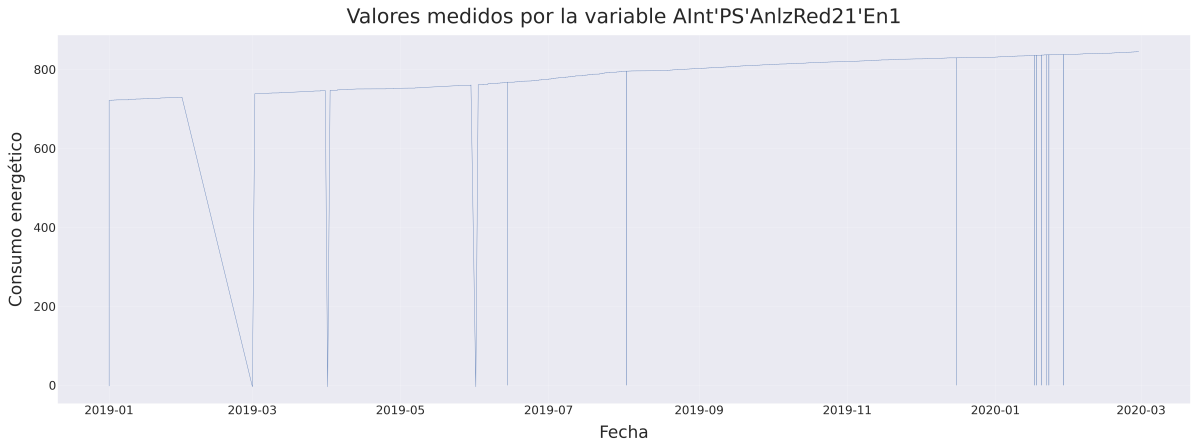


Figura 5.1: Medidas tomadas por el analizador de red general 21 (planta sótano)

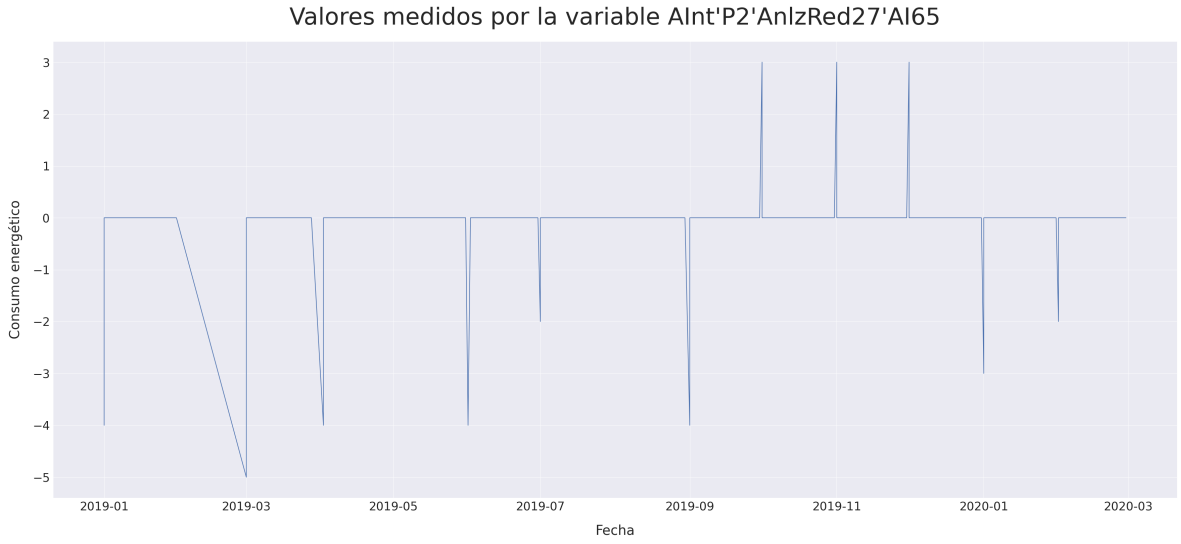
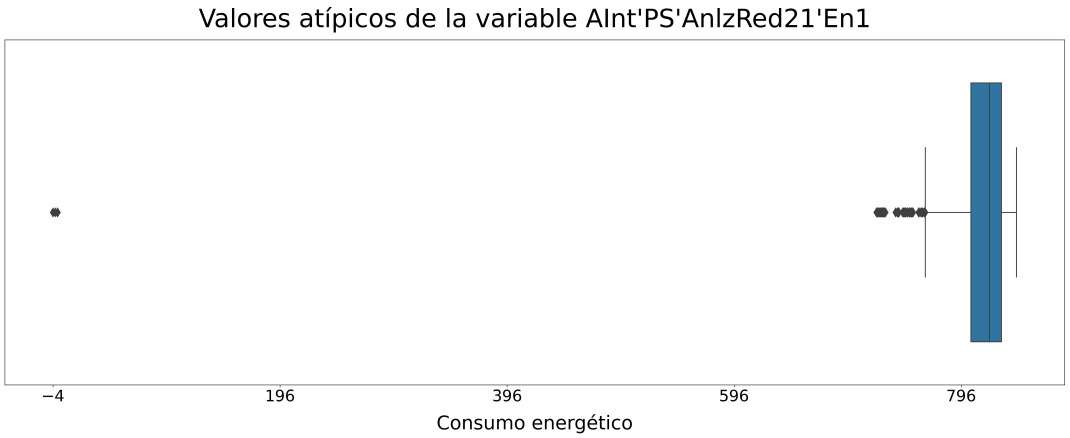


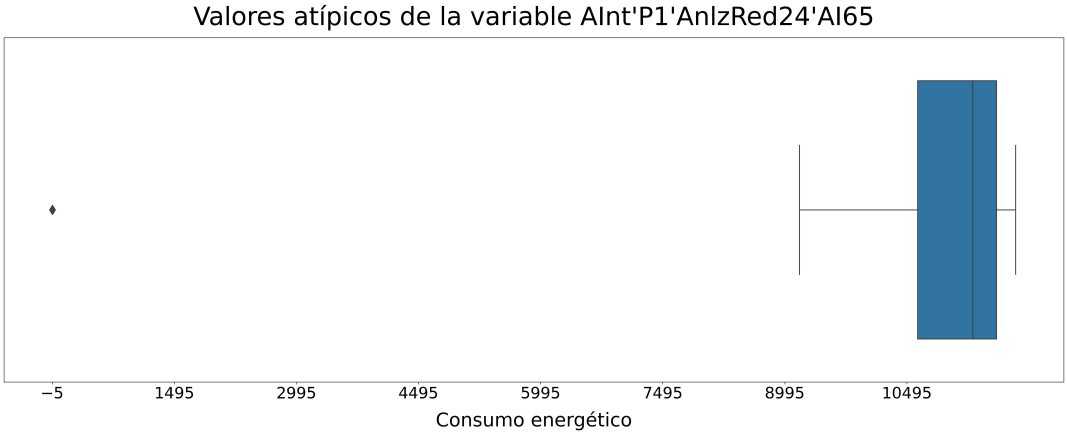
Figura 5.2: Medidas tomadas por el analizador de red 27 (segunda planta)

Como ya habíamos podido observar mediante la Tabla 5.4 y las representaciones anteriores contamos con valores atípicos en el conjunto de datos por lo que decidimos representar algunas mediciones de determinados analizadores de red mediante diagramas de caja y bigotes. En la Figura 5.3 podemos hacernos una idea de como es la distribución de los datos para un analizador de red general y otro de una zona más concreta.

Al mismo tiempo y contando con las seis variables que hemos identificado tras el primer análisis del conjunto de datos (identificador del analizador de red, marca de tiempo de la medición, valor de la medición, unidad de medida del sensor, validez de la medición e información sobre la ubicación) nos interesamos por la variable cualitativa que se ocupa de estudiar el tipo de validez que tiene la medición tomada por el sensor para saber que tipo de valores toma y la frecuencia de cada uno de ellos, información que se puede ver en la Figura 5.4. Cabe destacar que en la lectura de datos identificamos que el primer mes de datos con el que contamos carece de la columna *'tag_de_tipo'* por lo que con el fin de no reducir el conjunto de datos de los que disponemos eliminando estos registros y viendo que podrían ser mediciones válidas decidimos introducir una categoría denominada *Desconocido*.



(a) Outliers analizador de red general 21 (planta sótano)



(b) Outliers analizador de red 24 (primera planta)

Figura 5.3: Outliers de diferentes variables

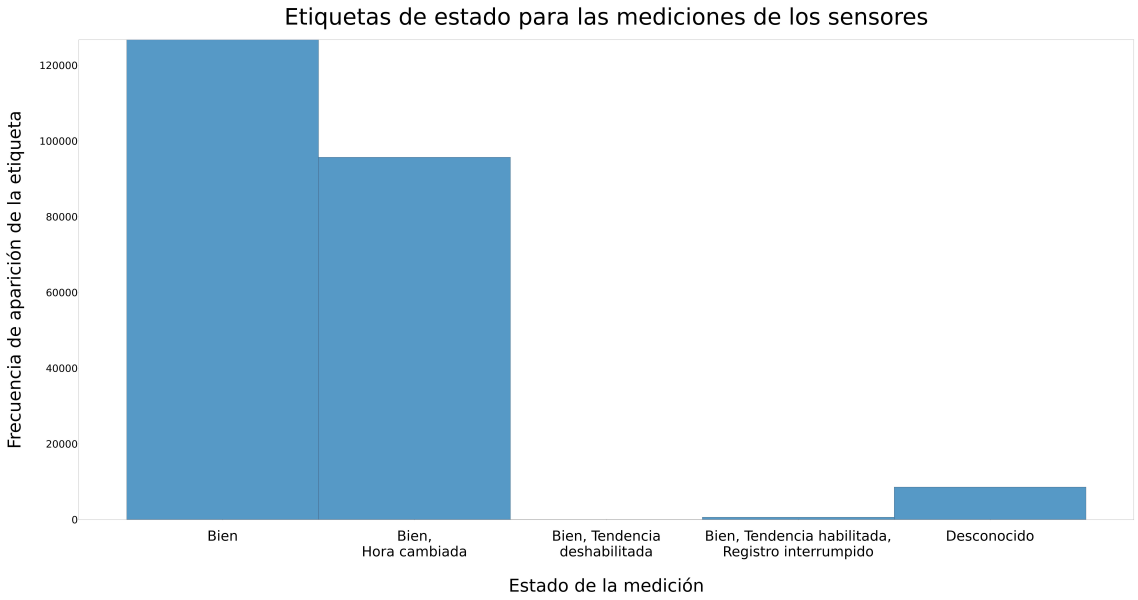


Figura 5.4: Frecuencia de valores presentes en la columna *'tag_de_tipo'*

5.2. Limpieza de los datos

Tras explorar y estudiar los datos de los que disponemos a través de estadísticos, gráficos de líneas y diagramas de caja y bigotes procedemos con la limpieza de datos que realizaremos en cuatro pasos:

- **Eliminación de valores NaN (Not a Number)**
  Partimos de un conjunto con 543982 registros de mediciones y encontramos un valor NaN para la variable *'device_measurement'* o valor medido.

- **Eliminación de registros con un tipo de datos inválido**
  Anteriormente, en la fase de representación gráfica de los datos nos damos cuenta de que para algunos identificadores de sensor obteníamos fallos a la hora de realizar alguna gráfica, al igual que a la hora de querer contabilizar los valores negativos para la variable que almacena la medición. Teniendo en cuenta que tenemos 543981 registros de datos a simple vista no logramos ver cuál puede ser la causa por lo que decidimos analizar el tipo de datos de todos los valores para todas las variables que tenemos.

  Utilizando la estructura *DataFrame* de *Pandas*[34] generamos columnas con el tipo de datos para cada variable del conjunto y utilizando la función *'value_counts()'* que nos devuelve el recuento de valores únicos, podemos ver en la Figura 5.5c que la variable *'timestamp'* cuyos valores deberían ser todos del tipo de datos *'datetime'* (fecha y hora) presenta valores de tipo *'float'* (número con decimales) y que la variable *'device_measurement'* cuyos valores deberían ser todos del tipo de datos *'float'* o *'int'* (números enteros) presenta valores de tipo *'str'* (cadena de texto) y *'datetime'*. A partir de esta información analizamos los datos de las figuras 5.5a y 5.5b y al tratarse de un número reducido de valores erróneos decidimos eliminar estos registros del conjunto.

- **Eliminación de mediciones negativas**
  En este paso descartamos los valores negativos presentes en las mediciones de los analizadores de red del conjunto de datos ya que estamos trabajando con cantidades de energía consumida y esta tiene que ser positiva.

- **Eliminación de registros duplicados**
  Para evitar la inserción de registros duplicados en la base de datos realizamos una búsqueda de estos basada en un subconjunto de variables formado por el identificador del sensor, la marca temporal, la medición realizada y la validez de la medición. Una vez eliminados estos registros contamos con un total de 517284 registros de mediciones para los trece meses de datos que tenemos.

	device_id	timestamp	device_measurement
303762	AInt'PB'AnlzRed11'AI65	2019-10-20 15:04:52	473.8
371814	AInt'PB'AnlzRed11'AI65	2019-11-10 12:49:37	1901-04-17 19:12:00
534192	AInt'PB'AnlzRed11'AI65	2020-02-11 11:58:26	473.8
171858	AInt'PB'AnlzRed11'AI65	2019-05-26 09:19:13	473.8
106093	AInt'PB'AnlzRed11'AI65	2019-12-04 22:26:27	473.8
42503	AInt'PB'AnlzRed11'AI65	2019-03-10 13:29:19	473.8
303721	AInt'PB'AnlzRed11'AI65	2019-10-12 03:36:04	1901-04-17 19:12:00
171834	AInt'PB'AnlzRed11'AI65	2019-05-03 11:20:08	473.8

(a) Atributo *'device_measurement'*

	device_id	timestamp	device_mea
1793	AInt'PS'AnlzRed22'AI65	2019-07-01 00:28:52	
1794	AInt'PS'AnlzRed22'AI65	2019-07-01 00:43:52	
1795	AInt'PS'AnlzRed22'AI65	2019-07-01 00:58:52	
1796	AInt'PS'AnlzRed22'AI65	2019-07-01 01:13:52	
1797	AInt'PS'AnlzRed22'AI65	2019-07-01 01:28:52	
...	...	...	
485595	AInt'PS'AnlzRed22'AI65	2020-02-29 23:13:15	
485596	AInt'PS'AnlzRed22'AI65	2020-02-29 23:28:15	
485597	AInt'PS'AnlzRed22'AI65	2020-02-29 23:43:15	
485598	AInt'PS'AnlzRed22'AI65	2020-02-29 23:58:15	
485599	AInt'PS'AnlzRed22'AI65	-43862.082373	

(b) Atributo *'timestamp'*

```
t_ts = dff['t_timestamp'].value_counts()
t_ts
```

```
<class 'datetime.datetime'>    543973
<class 'float'>                     9
Name: t_timestamp, dtype: int64
```

```
t_dev_m = dff['t_device_measurement'].value_counts()
t_dev_m
```

```
<class 'float'>                471091
<class 'int'>                   71520
<class 'str'>                     757
<class 'datetime.datetime'>       614
Name: t_device_measurement, dtype: int64
```

(c) Recuento del tipo de datos de las variables

Figura 5.5: Proceso de limpieza de datos con un tipo erróneo

Una vez que tenemos todos los datos preparados para proceder con las pruebas de rendimiento lo que haremos será dividir el conjunto de datos completo en conjuntos de datos del tamaño de 3, 6, 10 y 13 meses de datos para analizar el rendimiento de distintas bases de datos sobre distintos tamaños de muestras. En en la Tabla 5.5 podemos ver el número de registros de mediciones que contendrá cada tamaño del conjunto de datos.

Tamaño del conjunto de datos	3 meses	6 meses	10 meses	13 meses
Número de registros	21348	98292	316488	517285

Tabla 5.5: Número de registros para cada tamaño de datos

Capítulo 6

Diseño e implementación

6.1. Bases de datos utilizadas

Para desarrollar el presente trabajo y cumplir con los objetivos expuestos en la Sección 1.2 utilizaremos las bases de datos recogidas en la Tabla 6.1.

	InfluxDB	**TimescaleDB**	**OpenTSDB**	**KairosDB**
Modelo principal	SGBD de series temporales			
Modelo secundario	SGBD espacial	SGBD relacional	x	x
Website	Influxdata	Timescale	OpenTSDB	KairosDB
Año de lanzamiento	2013	2017	2010	2013
Lenguaje de implementación	Go	C	Java	Java
Lenguajes soportados	.Net, Haskell, Java, JS, Python, R y Ruby entre otros	.Net, C, C++, Java, JS, Python, R y Ruby entre otros	Erlang, Go, Java, Python, R y Ruby	Java, JS, PHP, y Python
SQL	Soporta un lenguaje similar a SQL	SQL basado en PostgreSQL	No	No
APIs y métodos de acceso	HTTP API y JSON over UDP	ADO.NET, JDBC y biblioteca nativa de C entre otros	HTTP API y Telnet API	Graphite protocol, HTTP REST y Telnet API
Particionamiento	Sharding	Partición hash (a través de espacio y tiempo)	Sharding	Sharding
Posición ranking DB-Engines (clasificación SGBD de series temporales)	1	5	10	17
Logo				

Tabla 6.1: Diferencias entre las bases de datos que vamos a utilizar

Cabe resaltar que la clasificación que se menciona en la Tabla 6.1 para las distintas bases de datos que vamos a probar procede del portal DB-Engines[22]. Solid IT, empresa austriaca

de consultoría informática, lanza en 2012 un proyecto con el propósito de recopilar información sobre sistemas gestores de bases de datos tanto relacionales como NoSQL. El ranking que nos proponen se actualiza mensualmente y está elaborada a partir de los siguientes criterios:

1. Número de menciones en páginas web: cantidad de resultados en motores de búsqueda como Google o Bing.
2. Interés general: frecuencia de búsqueda y popularidad ofrecida por Google Trends.
3. Frecuencia de discusiones técnicas: número de dudas o discusiones y usuarios interesados que utilizan páginas como Stack Overflow o Stack Exchange.
4. Número de ofertas de empleo: anuncios de trabajo en sitios web como Indeed.
5. Relevancia en redes profesionales: cantidad de perfiles en LinkedIn o Upwork.
6. Relevancia en redes sociales: número de menciones en Twitter.

6.1.1. InfluxDB

InfluxDB[20] es un almacén de datos de series temporales escrito en lenguaje de programación Go y compilado en un único binario sin dependencias externas. Éste forma parte de una plataforma de código abierto, desarrollada en 2013 por la empresa InfluxData, denominada pila TICK (Telegraf, InfluxDB, Chronograf y Kapacitor), diseñada para recopilar, almacenar, procesar y visualizar datos procedentes de series temporales como pueden ser métricas de rendimiento de un servidor, medidas de precipitaciones, datos de sensores industriales, etc.

Los datos de series temporales tienen algunas propiedades que los hacen diferentes en cuanto a la carga de trabajo que suponen con respecto a otros datos. InfluxDB se diseña pensando en la exploración de grandes volúmenes de datos y en la gestión del ciclo de vida de estos, dando como resultado tiempos de consulta de milisegundos en meses de datos o soluciones integradas de resumen de datos. De este modo, existen consultas continuas y políticas de retención, que son funciones que ayudan a automatizar los procesos de reducción de datos y la caducidad de estos para no generar problemas de almacenamiento.

Esta base de datos organiza las series temporales en un formato estructurado en el que podemos encontrar varios niveles, donde una serie temporal está formada por varios puntos de datos y un punto de datos tiene los siguientes elementos[6]:

- **Bucket**: cubo, contenedor lógico que almacena datos de series temporales cuyo identificador debe ser un nombre único dentro de la base de datos. Un bucket puede contener varias measurements y puede tener políticas de retención.
  En nuestro proyecto un bucket podría referirse al área de integraciones del edificio, que engloba los datos de energía de los sensores que vamos a almacenar.
- **Measurement**: medición, agrupación lógica de datos pertenecientes a una serie temporal. Esta medición contiene etiquetas, campos y un valor de tiempo, y tiene como objetivo recopilar distintas métricas dentro de un conjunto de elementos relacionados. Un

measurement es similar a una tabla en SQL y su nombre es la descripción de los datos almacenados.
En nuestro caso serán datos de energía, ya que es lo que miden los analizadores de red que forman nuestro conjunto de datos.

- **Tags**: etiquetas, pares clave/valor con valores que difieren pero que no cambian a menudo. Estas etiquetas deberán ser cadenas y servirán para guardar metadatos de cada serie temporal.
  En nuestro proyecto la etiqueta será el identificador del analizador de red. Los tags y el measurement son valores indexados con el fin de agilizar las búsquedas en series de tiempo específicas.
- **Fields**: campos, pares clave/valor que cambian con el tiempo, pueden tomar como valores números enteros, números de punto flotante, booleanos o cadenas.
  En nuestro proyecto contaremos con campos como unidad de medida, medición de energía, estado de la medición o información del sensor.
- **Timestamp**: marca temporal, cada punto de datos tiene un valor de tiempo que representa el momento en el que se registró la medición. En InfluxDB la marca de tiempo puede almacenarse en segundos o con mayor precisión si fuera necesario.

En cuanto al lenguaje de consulta utilizado por esta base de datos nos encontramos con la existencia de dos posibilidades:

- InfluxQL (Influx Query Language) es un lenguaje de consulta similar a SQL que se utiliza para interactuar con InfluxDB. Se diseñó con la intención de que resultase familiar a los ojos de usuarios de entornos SQL, comparte parte de la sintaxis pero no es SQL y carece de soporte para algunas operaciones avanzadas como pueden ser hacer uniones, pivotear o trabajar con datos geotemporales.
- Flux un lenguaje de script de datos diseñado para consultar, analizar y procesar datos de series temporales. InfluxData desarrolla en 2017 Flux con el objetivo de manejar específicamente datos de series temporales de forma más eficaz y flexible. Este surge de la combinación de la potencia de InfluxQL y la funcionalidad de TICKscript (lenguaje específico de dominio utilizado en Kapacitor para manejar alertas o cambios arbitrarios en los datos). Utilizando este lenguaje de consulta la estructura de una consulta en InfluxDB se compone de operaciones de entrada (source), transformación (pipe) y salida (destination).

Por otro lado, tenemos que mencionar el funcionamiento del motor de almacenamiento[56] de los datos ya que utiliza WAL (Write Ahead Log) o registro de escritura anticipada para garantizar la durabilidad de los datos en caso de fallo, además de escribir los datos en una caché en memoria, la cual se escribe en archivos TSM (Time-Structured Merge Tree) periódicamente en el disco. En los archivos TSM, los datos se organizan en formato de columnas y en bloques contiguos ordenados en el tiempo para cada valor de sus campos, optimizando de esta manera la compresión de los datos y la velocidad de consulta. Además, cuenta también con un índice de series temporales o TSI, que se encarga de controlar la rapidez de las consultas a medida que aumenta la cardinalidad de series almacenando claves de series agrupadas por medición, etiqueta y campo.

A continuación, en la Figura 6.1 podemos ver los cuatro componentes fundamentales de la arquitectura de esta base de datos. Podemos localizar tres almacenes de datos, *'WAL'* para recuperación, *'Catalog'* para almacenar metadados y *'Object Storage'* dedicado a almacenar los datos reales. En azul tenemos el proceso de introducción de datos, donde el *ingester* identificará las tablas y el esquema de columnas para los datos, validará los datos de forma síncrona con la solicitud de escritura, particionará los datos en días, eliminará datos duplicados y una vez estén procesados los guardará como un archivo Parquet (formato de archivo de datos en columnas de código abierto que surgió de Cloudera) cuyo tamaño será entre diez y cien veces menor que el de los datos en bruto y por último, se actualizará el almacenamiento 'Catalog'. En color rojo tenemos el proceso de compactación de datos que se encarga de obtener un número menor de archivos más grandes y sin solapamientos. En rosa, el recolector de basura se encargará de la retención de datos a partir de las políticas definidas por el usuario y la eliminación de archivos marcados por el compactador para recuperar almacenamiento. Finalmente, en verde tenemos el enrutador de consultas que se encarga de reenviar a un consultor, *querier*, las consultas enviadas en SQL o InfluxQL por los usuarios. Este formulará un plan óptimo de consulta utilizando DataFusion y Arrow (marcos de software para planificación, optimización y ejecución de consultas y procesado de datos en columnas) que ejecutará sobre los datos de la caché y del *ingester*.

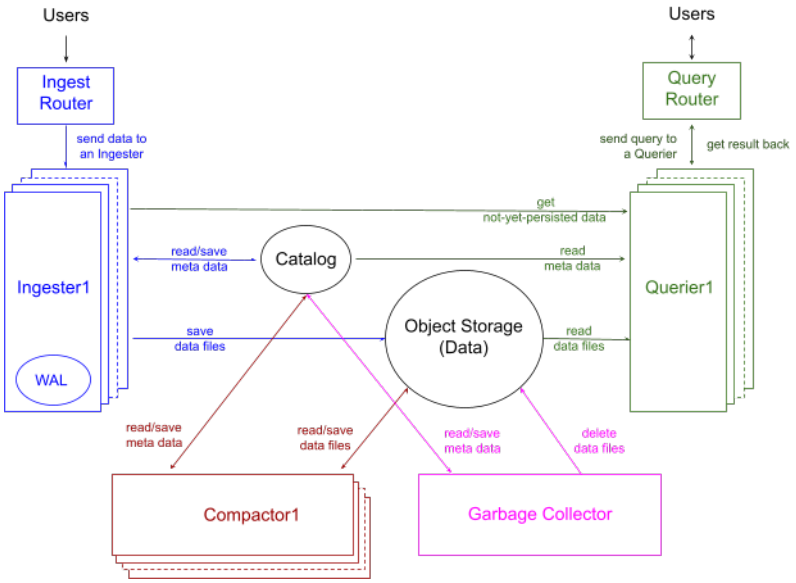


Figura 6.1: Arquitectura interna de InfluxDB 3.0 (Fuente: Influxdata)

A la hora de cargar datos en InfluxDB podemos utilizar bibliotecas cliente escritas en Python, C++, Java y muchos más lenguajes, o bien, utilizar ya sea la línea de comandos CLI, herramientas de terceros o una API HTTP(interfaz de programación de aplicaciones basada en el protocolo HTTP).

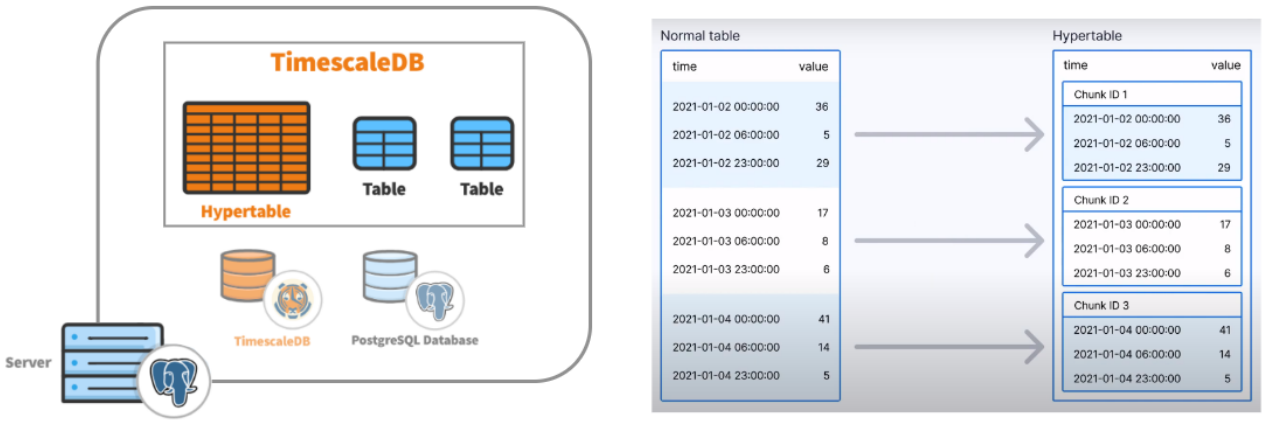
Grandes compañías como Siemens o SolarCity utilizan InfluxDB para almacenar datos de sensores o para monitorizar dispositivos IoT en infraestructuras en tiempo real, el CERN para almacenar métricas recogidas por Flume y Spark en un experimento llamado ALICE (A Large Ion Collider Experiment) o IBM para almacenamiento de estadísitcas, recopilación de métricas operativas con Telegraf y Grafana para visualizar datos de rendimiento de cara a mejorar sus productos.

6.1.2. TimescaleDB

TimescaleDB[54] es una base de datos relacional diseñada sobre PostgreSQL para series temporales, escrita en lenguaje de programación C y de código abierto. Timescale Inc lanza al mercado esta base de datos temporal en el año 2017 con el objetivo de abordar los desafíos que presenta el manejo de datos temporales a gran escala sirviéndose de la potencia de SQL y de las funcionalidades avanzadas de PostgreSQL. Presenta una alta escalabilidad horizontal siendo capaz de distribuir los datos en múltiples nodos o servidores, reduciendo la carga de trabajo además de optimizar las consultas utilizando técnicas de *'data skipping'* para acceder a datos relevantes con mayor rapidez.

A la hora de ofrecer sus servicios, TimescaleDB se centra en los siguientes aspectos para optimizar el almacenamiento y el rendimiento de los datos temporales:

- Hipertablas: concepto introducido por Timescale para definir tablas PostgreSQL que particionan los datos por tiempo, son una vista lógica que se crea sobre segmentos de los datos en bruto como podemos ver en la Figura 6.2b dando al usuario la sensación de estar trabajando con una tabla normal pero con la posibilidad de hacer uso de funciones adicionales que mejoran el rendimiento en la inserción y consulta de datos. El usuario podrá utilizar hipertablas para almacenar datos de series temporales y a su vez tablas PostgreSQL normales para datos relacionales, conviviendo ambas en el entorno de la base de datos como refleja la Figura 6.2a.



(a) Arquitectura interna

(b) Representación de hipertabla

Figura 6.2: Base de datos temporal TimescaleDB (Fuente: Timescale)

- Compresión: los datos de series temporales pueden comprimirse mediante el programador de tareas integrado de la base de datos para convertir filas de datos en columnas comprimidas. Utilizando los segmentos de las hipertablas, también llamados 'chunk' de datos definidos por un rango de tiempo específico y almacenados en tablas separadas, se agrupan varios registros en una sola fila reduciendo el espacio en disco que se necesitará y agilizando la velocidad de consulta de datos. TimescaleDB también ofrece la posibilidad de comprimir datos en función de su diferencia en lugar de almacenar el valor de cada punto de datos, útil en casos donde los datos presentan patrones repetitivos, o compresión por índices, que reduce la cantidad de datos a leer en las consultas.

- Agregados continuos: teniendo en cuenta el volumen y la velocidad de crecimiento de las series temporales, esta base de datos presenta el concepto de vista agregada continua que permite realizar cálculos agregados en los datos de manera eficiente. Las agregaciones se mantienen automáticamente y se actualizan a medida que llegan nuevos datos para evitar realizar cálculos sobre toda la tabla en tiempo real reduciendo los tiempos de respuesta.

En cuanto a la organización de los datos TimescaleDB organiza las series temporales utilizando los siguientes recursos[55]:

- **Tablas**: son la base del esquema de datos, se pueden crear las tablas que sean necesarias y transformarlas en las hipertablas mencionadas anteriormente. Cada tabla representará una entidad específica como pueden ser registro de eventos o de mediciones.
  En nuestro caso, contaremos con una tabla normal para almacenar datos básicos de los analizadores de red así como la unidad de medida en la que realizan las lecturas o información básica y una hipertabla para almacenar los registros de los sensores.

- **Columnas**: cada tabla tendrá un número de columnas en función de los atributos que queramos almacenar acerca de los datos. Estas podrán ser de diferentes tipos de datos, desde cadenas hasta datos en formato tiempo o datos numéricos.
  En nuestro proyecto necesitaremos contar con atributos como el identificador del sensor, la unidad de medida del sensor, información general, la medición, la validez de la medición y la marca de tiempo.

- **Índices**: la base de datos indexa por defecto las series temporales en función del atributo tiempo pero TimescaleDB nos da la posibilidad de crear nuevos índices aparte de éste para mejorar el rendimiento de las consultas en la búsqueda y recuperación de los datos que deseemos. Podemos crear índices compuestos por cualquier número de columnas, siempre que se incluya la columna con la marca de tiempo.

- **Claves primarias y restricciones**: al igual que en SQL podemos definir una o más claves primarias para identificar inequívocamente una fila dentro de una tabla y poder acceder a los datos de forma eficiente. Además, podemos fijar condiciones para los datos que vamos a almacenar asegurando la integridad y coherencia de nuestros datos.

- **Vistas**: cuando se habla de vistas, la base de datos hace referencia a consultas almacenadas que pueden tratarse como tablas virtuales creadas sobre subconjuntos de los datos. Esto está relacionado con las agregaciones continuas que utilizan y actualizan estas vistas en segundo plano, actuando únicamente sobre los nuevos datos y no sobre todo el conjunto.

En lo relacionado con las consultas de datos aparte de utilizar PostreSQL, TimescaleDB proporciona características adicionales para ayudar con el análisis de datos[53]:

- Optimizador SkipScan para consultas que utilizan DISTINCT en TimescaleDB 2.2.1 y posteriores, optimizando las consultas que pretenden encontrar valores únicos en los datos saltando incrementalmente de un valor ordenado al siguiente sin necesidad de leer todos los índices como hace PostreSQL nativo.

- Hiperfunciones que permiten analizar datos de series temporales y obtener información significativa. Estas, algunas incluidas por defecto en la base de datos y otras accesibles a través de la extensión Timescale Toolkit, están diseñadas para trabajar con el concepto de hipertabla y buscan sacar la máxima utilidad de SQL permitiendo realizar consultas críticas sobre los datos y funciones como rellenar datos que faltan, interpolaciones o realizar cálculos avanzados.
- Canalizaciones de funciones que transforman la programación funcional en consultas SQL. El elemento más importante dentro de una canalización es un tipo de datos personalizado denominado 'vector temporal' sobre el que los demás elementos (transformadores y finalizadores) actúan para crear la consulta deseada.

En ocasiones se quiere mantener un histórico de los datos, pero otras veces en las aplicaciones de series temporales los datos pierden utilidad a medida que pasa el tiempo por lo que se ofrece la posibilidad de establecer políticas de retención para eliminar determinados segmentos o datos completos, manteniendo resúmenes de ellos pero reduciendo el tamaño de datos antiguos.

A la hora de insertar datos en TimescaleDB podremos hacerlo a través de herramientas de terceros, a través de consultas de inserción típicas en SQL o bien mediante ficheros en formato CSV o JSON.

TimescaleDB es utilizado por empresas como Zondax, para proyectos de blockchain y servicios financieros, Conserv, para almacenar datos de sensores y monitorización de series temporales, o Edeva, encargada de crear soluciones para ciudades inteligentes que utiliza esta base de datos para agilizar las consultas sobre su gran volumen de datos.

6.1.3. OpenTSDB

OpenTSDB[31] (Open Time Series Database) es una base de datos temporal distribuida y escalable de código abierto, escrita en Java y desarrollada en el año 2010 por Benoit Sigoure para monitorizar las métricas del motor de búsqueda StumbleUpon.

El modelo de datos implementado por OpenTSDB se basa en una estructura de la serie de tiempo como dos o más puntos de datos que registran una medida en un momento determinado a lo largo del tiempo y que está formado por los siguientes elementos:

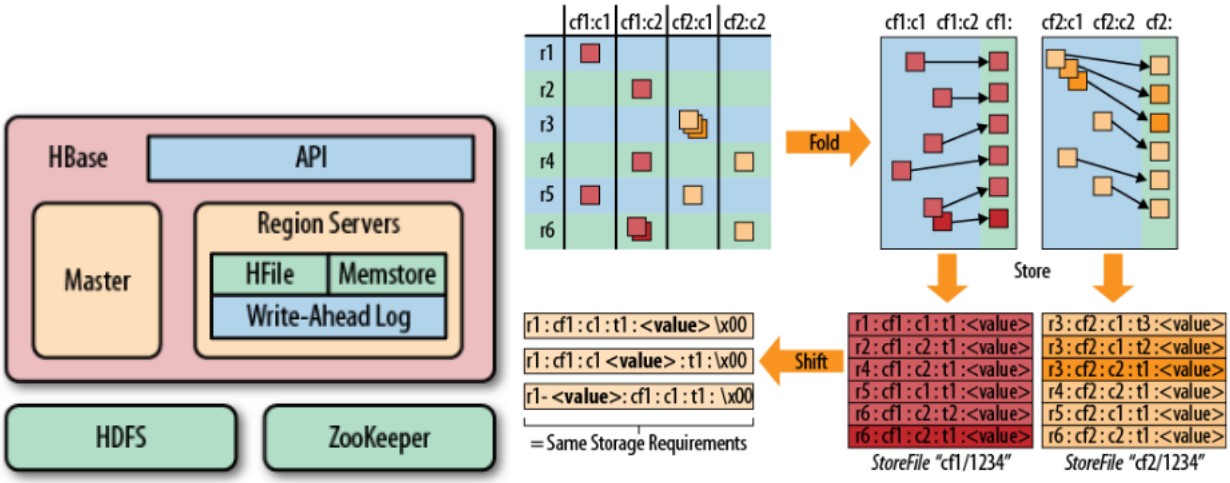
- **Metric**: la métrica hace referencia al nombre de una medida cuantitativa como puede ser las descargas de un archivo en un servidor o la temperatura en una ciudad. El nombre elegido deberá ser una cadena que no contenga espacios.
  En nuestro proyecto la métrica sería la energía medida por los analizadores de red.
- **Value**: un valor que represente la medición numérica de la métrica dada.
  En nuestro caso el valor serán las mediciones realizadas a lo largo del tiempo por los sensores.
- **Timestamp**: será la marca de tiempo en la que se registró el valor de la métrica. Este deberá estar en formato de tiempo UNIX, que representa la cantidad de segundos o milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC.

- **Tags**: las etiquetas son pares clave-valor que proporcionan los metadatos para identificar y filtrar los puntos de datos de una serie. Las series de tiempo se identificarán entre sí por la combinación de la métrica y sus etiquetas.
  En nuestro caso, tendremos etiquetas para el identificador de cada analizador de red, la unidad de medida del sensor, la validez de la medición e información relevante.

OpenTSDB admite en la actualidad Apache HBase[12] como principal backend o arquitectura interna de almacenamiento, la cual es una base de datos no relacional que sigue el modelo de datos clave-valor. En la Figura 6.3a vemos los componentes principales de la arquitectura empleada por HBase:

- RegionServers es el componente principal y administra una o varias regiones que son rangos contiguos de filas en una tabla.
- Master es el responsable de administrar el clúster de HBase, controlando la asignación de regiones a los RegionServers, el balanceo de carga y la gestión de las tablas y los esquemas.
- HFile formato de archivo utilizado para almacenar datos en el disco. Los HFile están formados por bloques de datos comprimidos, organizados en función del tamaño y del espacio temporal.
- Memstore como estructura de almacenamiento temporal en memoria utilizada por cada región antes de que se escriban en el disco.
- WAL como almacenamiento basado en archivos que registra todos los cambios que se realizan sobre los datos en HBase para cada RegionServer.
- HDFS (Hadoop Distributed File System) encargado de almacenar conjuntos de datos masivos.
- ZooKeeper que es un servicio de coordinación distribuida para realizar la asignación y bloqueo de regiones y un seguimiento del estado de éstas, del Master y de otros componentes del sistema.

HBase almacena los datos en uno o más HFiles y organiza estos en tablas (Figura 6.3b) donde encontramos filas que se identifican unívocamente por su clave de fila (row key) y familias de columnas que afectan a la distribución física de los datos en el almacenamiento. Todas las filas de una tabla tienen las mismas familias de columnas, identificadas por un nombre de columna (qualifier), aunque una fila no necesita tener datos en todas sus familias. Las celdas de datos se identifican de forma única por una combinación de una clave de fila, una familia de columnas y un nombre de columna y el valor que almacenan están versionados por una marca de tiempo. Hbase permite almacenar múltiples versiones de una celda, donde cada versión tiene una marca de tiempo asociada. OpenTSDB crea una tabla HBase que incluye columnas para la métrica, la marca de tiempo, las etiquetas y el valor, y donde la clave de fila se construye a partir de la combinación de la métrica y las etiquetas asociadas a los datos de la serie.



(a) Arquitectura interna

(b) Representación de tabla

Figura 6.3: Características de HBase (Fuente: capítulo 8. Architecture y capítulo 9. Advanced Usage, respectivamente[12])

Para realizar las consultas[32] en OpenTSDB se utiliza un lenguaje de consulta flexible para extraer, manipular y analizar datos. Para consultar los datos se pueden utilizar herramientas de terceros como Grafana o Bosun, herramientas CLI o a través de la API HTTP. En la Tabla 6.2 se recogen los elementos que puede tener una consulta en OpenTSDB.

Parámetro	Tipo de dato	Campo	Descripción
Hora de inicio	Cadena o número entero	Obligatorio	Instante inicial al que hace referencia la consulta, puede ser absoluta o relativa.
Hora de finalización	Cadena o número entero	Opcional	Instante final al que hacer referencia la consulta, si no se da se toma la hora actual
Métrica	Cadena	Obligatorio	Nombre completo de la métrica dentro del sistema
Función de agregación	Cadena	Obligatorio	Funciones matemáticas para combinar series temporales
Filtro	Cadena	Opcional	Filtro de etiquetas para reducir el número de series temporales recogidas en la consulta
Subconjunto	Cadena	Opcional	Intervalo o función para reducir el número de puntos de datos devueltos
Tasa	Cadena	Opcional	Tasa de cambio, por segundo, para el resultado de la consulta
Funciones	Cadena	Opcional	Funciones de manipulación de datos
Expresiones	Cadena	Opcional	Funciones de manipulación de datos entre series temporales

Tabla 6.2: Componentes de las consultas en OpenTSDB

Además de HBase, OpenTSDB también funciona con otras tecnologías como Bigtable de Google en la nube o Cassandra. Los datos de series temporales se dividen en porciones que se distribuyen en clústeres de nodos de HBase, donde cada porción tiene un conjunto de series permitiendo un almacenamiento escalable y eficiente. Por otro lado, OpenTSDB se beneficia de las capacidades de replicación y tolerancia a fallos de HBase para garantizar la disponibilidad de los datos.

En esta base de datos, en lo que se refiere a su funcionamiento[2] y que está reflejado en la Figura 6.4, el componente principal es el TSD (Time Series Daemon). Los "demonios TSD" son procesos independientes del servidor que se ejecutan en los nodos del clúster y se encargan de manejar la escritura y lectura de datos en OpenTSDB. Los usuarios de la TSD no acceden directamente al almacén de datos, sino que se comunican con ella a través de protocolos de tipo telnet, API HTTP o una interfaz gráfica integrada.

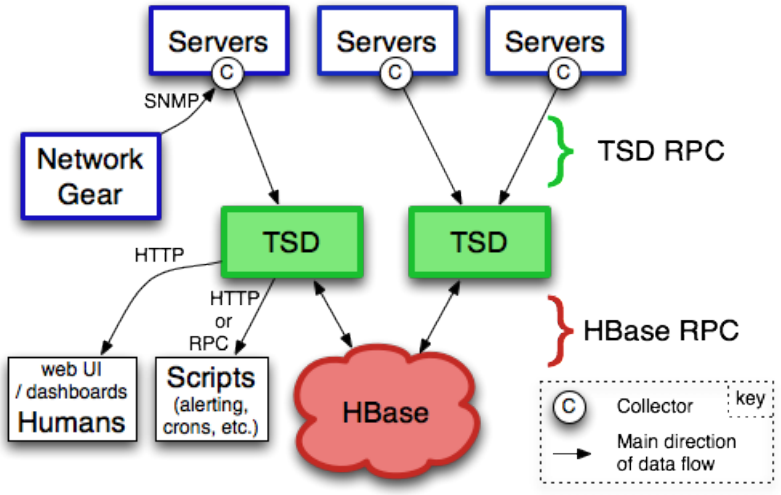


Figura 6.4: Arquitectura de OpenTSDB (Fuente: OpenTSDB)

OpenTSDB admite la ejecución de varios TSD para gestionar una única consulta, o lo que también se conoce como balanceo de carga, que permite la lectura paralela desde HBase para acelerar la obtención de datos. Por otro lado, también permite escrituras concurrentes sin utilizar bloqueos, haciendo que las escrituras sean idempotentes estableciendo un límite temporal fijo para cada fila de datos. En definitiva, los demonios TSD son los responsables de almacenar, recuperar, procesar y mantener los datos en OpenTSDB asegurando un almacenamiento distribuido, escalable y eficiente.

Por último, mencionar un aspecto importante dentro de esta base de datos referente a la cardinalidad. En OpenTSDB, la cardinalidad se entiende como el número de valores únicos que puede tener un 'tag', o etiqueta, en los datos almacenados y tiene mucha importancia en la fase del diseño de datos y etiquetas ya que si es alta puede tener un impacto negativo en el rendimiento y en la escalabilidad. Los índices utilizados para acelerar las consultas se basan en las combinaciones de etiquetas y métricas, por lo que tener muchas combinaciones resulta en tener muchos índices y con ello aumenta la carga de procesamiento en las consultas. En cuanto a la escalabilidad, si la cardinalidad es alta puede llevar a un mayor consumo de memoria así como incrementar la complejidad en la administración de los datos.

Empresas como Kentik, Server Density y Cloud Wiz utilizan los servicios de OpenTSDB.

6.1.4. KairosDB

KairosDB[24] es una base de datos de series de tiempo basada en Cassandra, escrita en Java, de código abierto y desarrollada en el año 2013 por KairosDB Inc.

A la hora de ofrecer una base de datos de series de tiempo eficiente KairosDB prioriza el rendimiento, utilizando tecnologías como Apache Cassandra y Apache Lucene para facilitar un acceso rápido a los datos y consultas eficientes; la escalabilidad, permitiendo la expansión horizontal mediante la agregación de nodos con el fin de manejar grandes cargas de trabajo y volúmenes de datos sin perjudicar el rendimiento; la eficiencia en el almacenamiento, empleando un modelo de datos basado en columnas, técnicas de compresión e indexación y la configuración de la granularidad de los datos almacenados; la flexibilidad en las consultas, proporcionando numerosas funciones y operaciones de consulta como agregaciones o filtros por etiquetas; y la facilidad de integración, pudiéndose complementar con otras herramientas o plugins para ampliar las características de la base de datos.

El modelo de datos que sigue KairosDB estructura las series de tiempo de forma similar a la base de datos descrita anteriormente, OpenTSDB. El formato de los datos emplea los siguientes componentes:

- **Metric**: la métrica hace referencia a un conjunto de series temporales relacionadas, agrupan y organizan los datos en el nivel más alto de la base de datos.
  En nuestro caso será la energía medida por los analizadores de red del área de integraciones del edificio.

- **Datapoints**: los puntos de datos representan los valores y marcas de tiempo asociados a una métrica en particular.
  Cada punto de datos hace referencia a una medición y se almacena en orden cronológico, en nuestro caso serán el valor de energía medida acompañado de una marca de tiempo con formato de tiempo Unix en segundos o milisegundos.

- **Tags**: las etiquetas son pares clave-valor que proporcionan información adicional de la serie temporal y que sirven para filtrar los datos a la hora de realizar consultas.
  En nuestro estudio emplearemos estas etiquetas para agregar información que describe los datos almacenados como el identificador del analizador de red, la unidad de medida de energía, la validez de la medición e información extra.

En cuanto al backend utilizado por esta base de datos nos encontramos con dos posibilidades que son H2 o Cassandra. KairosDB está configurado por defecto para utilizar H2 que es una base de datos relacional en memoria y que puede ser utilizada para casos de prueba u ocasiones en las que la escalabilidad no sea un requisito principal. Por otro lado, tenemos Apache Cassandra[3] que nos ofrece escalabilidad, tolerancia a fallos y un mayor rendimiento a la hora de trabajar con macrodatos que será la que utilicemos en este proyecto.
KairosDB a través de Cassandra utiliza un modelo de datos basado en filas anchas y columnas para almacenar los datos ligados a la variable tiempo en tablas. Este sistema de almacenamiento subyacente agrupa datos de un período de tiempo determinado en cada fila en la fase de escritura y cuanto mayor sea la cantidad de puntos de datos que quepan en una de estas filas mayor será el rendimiento a la hora de consultar los datos. Las familias de columnas que nos podemos encontrar siguiendo este esquema son las siguientes:

- *'data_points'*: almacén de datos.

- *'row_key_index'*: índice heredado que se utiliza para buscar las filas necesarias en las consultas.
- *'row_key_time_index'*: índice para saber en que marco temporal se almacena una métrica. Facilita la recuperación de datos en consultas que se orientadas a un rango de tiempo específico.
- *'row_keys'*: clave de filas para tablas donde se almacenan las métricas o clave de partición, la cual suele ser la combinación de la métrica, la marca de tiempo de la fila y las etiquetas. Esta clave ayuda a distribuir puntos de datos relacionados en diferentes nodos de un clúster, es decir, organizan los datos en las tablas de Cassandra.
- *'tag_indexed_row_keys'*: índice de etiquetas en la clave de filas para facilitar el filtrado de los datos en función de las etiquetas asociadas a la métrica.
- *'string_index'*: es utilizado para acelerar la búsqueda de datos basada en valores de tipo cadena, pudiéndose crear índices específicos para el valor de una etiqueta.
- *'service_index'*: almacén de metadatos.
- *'spec'*: tabla interna que almacena ajustes internos de la base de datos como el ancho de fila, que por defecto son tres semanas pero se puede redefinir, y la granularidad temporal.

En lo referente a las consultas en KairosDB cabe destacar los siguientes aspectos[17]:

- Almacenamiento de etiquetas
  Cada combinación de etiqueta y valor, para todas las etiquetas asociadas a una métrica, se escribirá en una partición separada utilizando la nomenclatura CQL (Cassandra Query Language) y a su vez la clave de partición se escribirá en una tabla de índices de partición. Por ejemplo, en nuestro caso para todo el conjunto de datos tendremos cuatro etiquetas con un número determinado de valores que pueden tomar; identificador del analizador con 94 valores, unidad de medida del analizador con 3 valores, validez de la medición con 5 valores e información general con 95 valores; de esta manera tendríamos un total de 132540 particiones.
- Proceso de las consultas
  - **Fase 1**: KairosDB lee el índice de claves de la fila para el periodo de tiempo especificado en la consulta, de esta manera procesará todos los índices que haya en las particiones (tres semanas por defecto) que se quieren consultar. En esta fase, la cardinalidad de las combinaciones de etiqueta y valor repercutirá en la eficiencia de las consultas al tener que leer un mayor o menor número de claves de índices de partición.
  - **Fase 2**: KairosDB una vez haya obtenido los datos, los filtrará si se han incluido etiquetas en la consulta. En esta fase, la cardinalidad mencionada sólo afectará a la eficiencia de la consulta si no se filtran los datos.
- Métricas sobre consultas e índices
  KairosDB define métricas internas que pueden utilizarse para saber que parte de las consultas toma más tiempo para poder tomar decisiones sobre cómo organizar los datos para

almacenarlos. Estas son mostradas cada vez que se realiza una consulta al backend y se puede realizar operaciones con ellas para conseguir información más específica. Por ejemplo, tendríamos 'kairosdb.datastore.query_time' que mide el tiempo que se tarda en procesar la consulta sin incluir agregaciones de datos o 'kairosdb.datastore.cassandra.key_query_time' que mide el tiempo que se tarda en consultar la información de la clave de fila, con respecto a estas dos métricas podríamos obtener el tiempo de lectura de datos de Cassandra restándolas.

A la hora de escribir consultas en esta base de datos lo haremos mediante una consulta con formato JSON enviada a la base de datos utilizando una REST API[50]. La consulta más básica que podremos realizar tendrá que contar obligatoriamente con los siguientes parámetros:

- *time_expr*: el rango de tiempo que se quiere consultar puede expresarse de tres maneras: relativa (5d ago, 1h ago, etc), absoluta (milisegundos en formato de tiempo UNIX) o especial para hacer referencia a la actualidad como instante final de consulta (now).

- *metric_expr*: contiene el nombre de la métrica que se quiere consultar y una condición de etiqueta que es opcional. Se definiría de la siguiente manera:

Luego se podrán añadir otros campos a mayores para modelar el conjunto de datos que deseamos obtener, como por ejemplo:

- *condition_expr*: filtra los datos en función de los valores de etiquetas que se incluyan en la consulta y se escribirá como *'tagX = valX'*.

- *aggregation_expr*: para realizar funciones de agregación sobre los datos consultados, por ejemplo 'sum(1d)' para obtener el sumatorio de los datos en un día o 'avg(1d)' para conseguir el promedio diario. Es importante destacar que esta base de datos necesita que le proporcionen el tamaño en el que muestrear los datos, es decir, si se quiere obtener la media de los datos almacenados habrá que especificar el rango en el que tiene que dividir los datos, aceptando por rango más alto el de un año.

- *group_expr*: para agrupar los datos en función de los valores de alguna etiqueta que contenga la métrica, definido como 'tags : *tagX*'.

Una de las funcionalidades destacable de KairosDB que tiene como fin mejorar el rendimiento de las consultas es la posibilidad de crear "rollups" que se utilizan para crear resúmenes de datos variando la granularidad temporal y aplicando funciones de agregación. Los "rollups" realizan consultas sobre los datos, aplican funciones y escriben los resultados en otra métrica manteniendo los datos originales. Estos se programan creando una tarea que tiene uno o varios "rollups" y una frecuencia de ejecución.

Empresas como Proofpoint, dedicada a la seguridad empresarial ofreciendo software para la prevención de la pérdida de datos, Zalando, dedicada a la venta online de diversos productos, o Abiquo, plataforma de software de computación en la nube, utilizan los servicios que ofrece KairosDB.

6.2. Diseño del caso de estudio

En el diseño de nuestro caso de estudio tenemos por un lado que ocuparnos de los datos presentados en la Sección 5 y por consiguiente, determinar la arquitectura de almacenamiento que se emplea en cada base de datos para poder modelar los datos y adaptar el diseño del caso de estudio a cada una de ellas. Por otro lado, tenemos que hacer frente al almacenamiento de los resultados de los experimentos que se explican en la Sección 7. El SGBD que utilizaremos será PostgreSQL que es un sistema de gestión de bases de datos relacional robusto, de código abierto y ampliamente utilizado. Siguiendo el modelo lógico de la base de datos mostrado en la Figura 6.5 definimos seis tablas para almacenar información relevante de las pruebas de rendimiento que llevaremos a cabo.

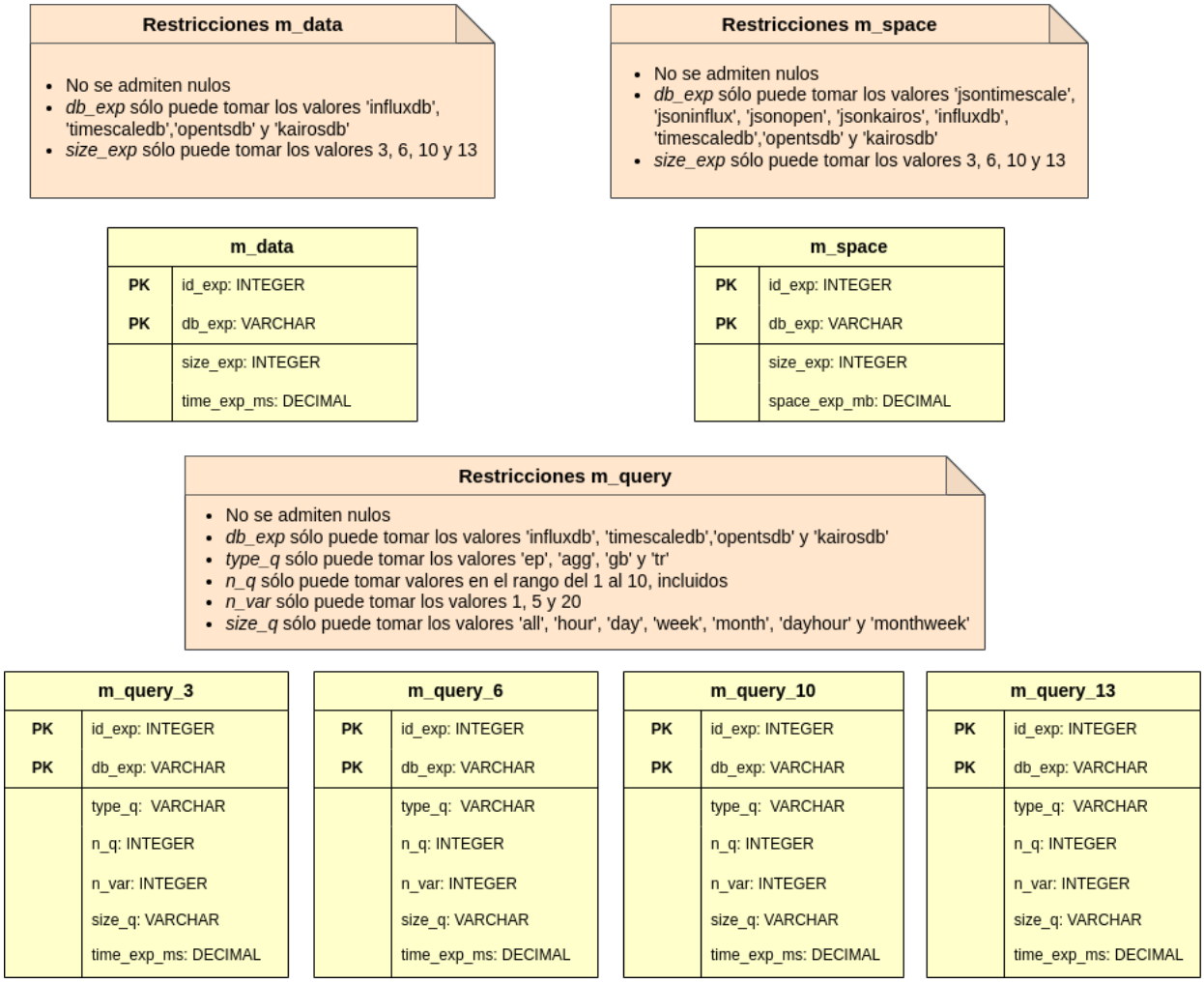


Figura 6.5: Modelo lógico de la base de datos PostgreSQL (Figura de elaboración propia)

Para cada experimento, la tabla *'m_data'* guardará el tiempo de carga en milisegundos de diferentes tamaños de conjuntos de datos de series temporales y la tabla *'m_space'* registrará en megabytes el tamaño del archivo de datos así como el tamaño de almacenamiento necesario para almacenar en cada base de datos dichos conjuntos.

A su vez, decidimos tener una tabla *'m_query'* por cada tamaño del conjunto de datos ya

que de esta forma resulta más cómodo el acceso a los datos de cada experimento ya que contamos con muchas columnas para definir otros parámetros importantes de las pruebas aunque se podrían fusionar en una única tabla. Estas tablas reunirán información perteneciente al tiempo que toma ejecutar diferentes consultas en cada base de datos con el fin de analizar cómo varía el rendimiento con el tamaño de los conjuntos de datos y con el número de variables (identificadores de analizadores de red) solicitadas en dichas consultas. El atributo *'type_q'* identifica el tipo de consulta realizada como 'ep' (exact point) para la consulta de una marca de tiempo determinada, 'agg' (aggregation) para las consultas analíticas básicas, 'gb' (group by) para las consultas de agrupación y 'tr' (time range) para las consultas que aplican rangos de tiempo. Además, el atributo *'size_q'* nos permite diferenciar dentro de los dos últimos tipos de consultas el rango temporal de agrupación o de consulta que estamos abarcando.

La implementación del caso de estudio en InfluxDB se hace aplicando un modelo basado en pares clave-valor.

Para la implementación del caso de estudio en TimescaleDB, seguimos un modelo relacional y como puede verse en la Figura 6.6, definimos una tabla para los identificadores con alguna información adicional y una hipertabla para guardar los datos temporales. Como ya se explicó anteriormente el concepto de hipertabla en esta base de datos nos permite segmentar los datos en función del tiempo y aprovechar dicha arquitectura para optimizar el almacenamiento. Además, definimos un índice sobre la columna del identificador en la hipertabla con el fin de aprovechar las oportunidades que nos da esta base de datos para mejorar el rendimiento de las consultas.

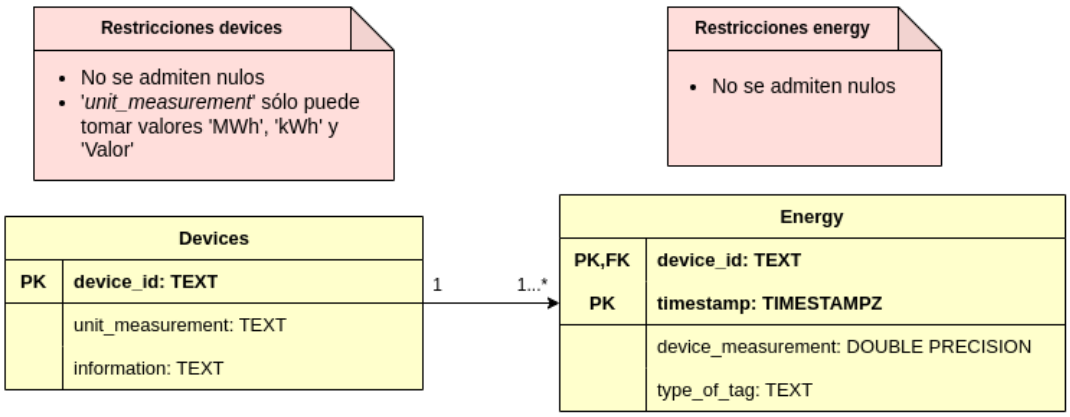


Figura 6.6: Modelo entidad-relación para TimescaleDB (Figura de elaboración propia)

La implementación del caso de estudio en OpenTSDB se hace utilizando una arquitectura basada en HBase que aplica un modelo basado en pares clave-valor.

La implementación del caso de estudio en KairosDB se consigue utilizando un modelo basado en colecciones de columnas y pares clave-valor que surge de la arquitectura manejada por Cassandra en su backend.

6.3. Implementación de las bases de datos

Puesto que para las pruebas de rendimiento trabajaremos con cuatro bases de datos optamos por utilizar Docker. Esta es una plataforma de código abierto basada en la virtualización de entornos aislados, llamados contenedores, para facilitar el despliegue de aplicaciones en la propia máquina. En otras palabras, esta herramienta nos permite crear un contenedor donde correrá el servicio deseado a partir de una imagen que consiste en un paquete ejecutable que contiene código, bibliotecas y una configuración inicial para poder funcionar. De esta manera sólo tendremos que modificar dicha configuración en base a nuestras necesidades sin preocuparnos de otras muchas cosas que implicaría instalar en nuestra máquina cada base de datos.

Docker nos ofrece portabilidad de servicios y consistencia de estos, así como un rápido montaje y eliminación de entornos que en nuestro caso agiliza de manera significativa el proceso de experimentación. Como vemos en la imagen 6.7 utilizaremos una herramienta de Docker denominada Compose, la cual consiste en un archivo YAML (lenguaje de serialización de datos) que nos permite definir servicios y lanzarlos en contenedores de Docker. En primer lugar, este archivo incluido en la Sección A.1 descargará las imágenes para las bases de datos PostgreSQL[16], InfluxDB[19], TimescaleDB[52], OpenTSDB[15] y KairosDB[11].

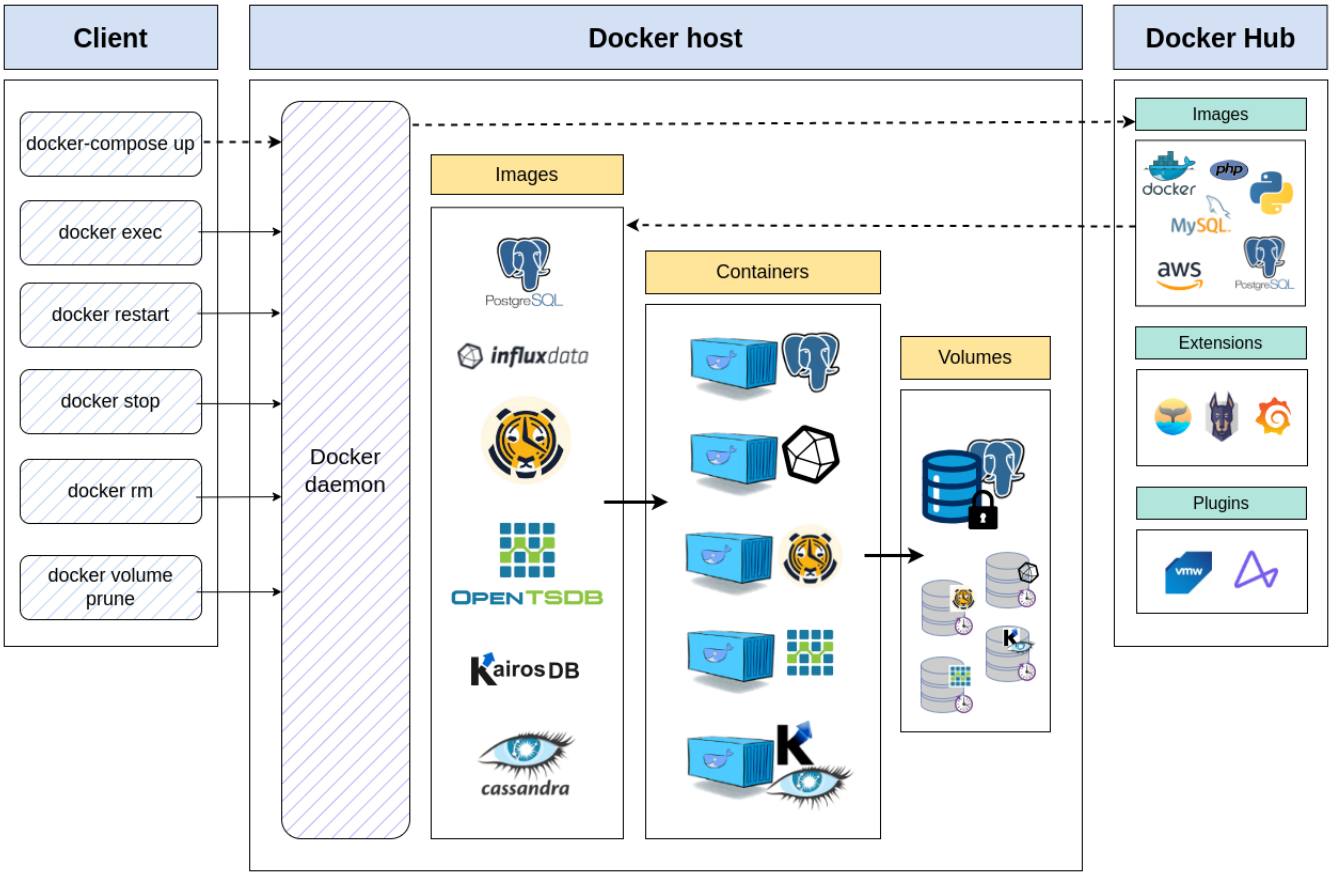


Figura 6.7: Arquitectura de contenedores en Docker para nuestro proyecto (Figura de elaboración propia)

Por otro lado, para realizar las pruebas de las bases de datos bajo las mismas condiciones nos encargaremos de crear y eliminar tanto contenedores como volúmenes para cada experimento

con cada tamaño del conjunto de datos, número de variables consultadas y de cada base de datos con la excepción de que la base de datos PostgreSQL se mantendrá activa y nos encargaremos de respaldar los datos mapeando su volumen a la máquina local para no perder los resultados cuando paremos el contenedor.

Como mencionamos anteriormente, los contenedores contienen una configuración inicial que podemos y en la mayoría de casos, necesitamos modificar para conseguir nuestros objetivos. Para las bases de datos que utilizaremos necesitaremos modificar los ficheros de configuración de TimescaleDB para que incorpore dos librerías y el de OpenTSDB para que cuando se consulte una etiqueta que no exista en la base de datos la respuesta sea vacía para dicha etiqueta en lugar de cancelar la ejecución de la consulta y devolvernos un error.

6.4. Estructuras de datos para la ingesta de los Raw Data en cada base de datos

Para llevar a cabo las pruebas que se especifican en la Sección 7 utilizando las bases de datos que se presentan en la Sección 6.1 decidimos generar una estructura común para nuestros datos por lo que optamos por el formato JSON (JavaScript Object Notation). Se trata de un formato ligero de datos ampliamente utilizado cuyo proceso de generación e interpretación resulta muy sencillo y cuya organización se basa en pares clave-valor.

Para evaluar el rendimiento de carga de datos entre las diferentes bases de datos, así como la escalabilidad de cada una de éstas con respecto al aumento del volumen de datos generaremos cuatro archivos en formato JSON de tres, seis, diez y trece meses con una sintaxis adecuada para cada base de datos:

- Formato JSON para InfluxDB (marca de tiempo UTC (Coordinated Universal Time)):

```
{
"measurement": "energy",
"fields": {
  "dev_m": 721.44
  },
"time": "2019-01-01T17:29:08Z",
"tags": {
  "device_id": "AInt'PS'AnlzRed21'En1",
  "unit_measurement": "MWh",
  "type_of_tag": "Desconocido",
  "information": "LUCIA:Area Integraciones'Planta..."
  }
}
```

- Formato JSON para TimescaleDB (marca de tiempo sin zona horaria):

```
{
"device_id": "AInt'PS'AnlzRed21'En1",
"timestamp": "2019-01-01T17:29:08",
```

```
    "device_measurement": 721.44,
    "unit_measurement": "MWh",
    "type_of_tag": "Desconocido",
    "information": "LUCIA:Area Integraciones'Planta..."
    }
```

- Formato JSON para OpenTSDB (marca de tiempo Unix en milisegundos). Se modifican las cadenas de texto que toman las etiquetas porque no se admiten comillas simples, espacios ni signos de puntuación:

```
    {
    "metric": "energy",
    "value": 721.44,
    "timestamp": 1546360148000,
    "tags": {
      "device_id": "AInt_PS_AnlzRed21_En1",
      "unit_measurement": "MWh",
      "type_of_tag": "Desconocido",
      "information": "LUCIA_Area_Integraciones_Planta..."
      }
    }
```

- Formato JSON para KairosDB (marca de tiempo Unix en milisegundos):

```
    {
    "name": "energy",
    "timestamp": 1546360148000,
    "type": "double",
    "value": 721.44,
    "tags": {
      "device_id": "AInt'PS'AnlzRed21'En1",
      "unit_measurement": "MWh",
      "type_of_tag": "Desconocido",
      "information": "LUCIA:Area Integraciones'Planta..."
      }
    }
```

Capítulo 7

Estudio de rendimiento de bases de datos

A la hora de realizar pruebas de rendimiento para diferentes bases de datos debemos asegurar unas condiciones similares para cada una de ellas y fundamentalmente realizar los experimentos en el mismo entorno para que los resultados sean comparables entre sí. El equipo experimental en el que probaremos las bases de datos que hemos seleccionados consta de un procesador Intel® Core™ Xeon® Gold 6128 CPU @ 3.4GHz x 4, memoria RAM de 32GB, y un sistema operativo de Ubuntu 22.04.3 LTS 64-bits. Realizaremos la comparación de InfluxDB v2.7.3, TimescaleDB v2.12.2, OpenTSDB v2.4.0 con HBase v1.4.4 y KairosDB v1.2.1 con Cassandra v3.11.16.

Cabe destacar que para la instalación y configuración de las bases de datos, como se menciona en la Sección 6.3, utilizaremos Docker Engine v24.0.7 y PostgreSQL v16 para almacenar los resultados de los experimentos.

7.1. Métricas

A la hora de evaluar el rendimiento de una base de datos hay que elegir los aspectos clave que queremos analizar, en nuestro caso nos centraremos en los siguientes experimentos[42]:

- Latencia de carga: hemos separado los datos con los que contamos en conjuntos de cuatro tamaños diferentes (3, 6, 10 y 13 meses) para medir el tiempo que necesita cada base de datos para cargar dichos conjuntos.
- Tamaño de almacenamiento: contaremos con cuatro conjuntos de datos de tamaños diferentes, por lo que resulta imprescindible medir el tamaño de almacenamiento que necesita cada base de datos para almacenar cada uno de ellos. Podremos ver como varía el tamaño a medida que agregamos más datos y lo más importante, conocer si existen grandes diferencias entre el modelo de datos y las técnicas de compresión que utilizan las bases de datos seleccionadas.
- Latencia en las consultas: nuestro trabajo consistirá en medir el tiempo que tarda en ejecutarse cada consulta desde su envío a la base de datos hasta la obtención de una

respuesta por parte del usuario. Construiremos las siguientes consultas basándonos en cuatro tipos básicos:

1. Consultas basadas en un instante de tiempo específico:
   A. Obtener para un determinado sensor una medición de energía en un día y una hora determinados.
2. Consultas para análisis básico:
   A. Obtener para un determinado sensor la mínima medición de energía.
   B. Obtener para un determinado sensor la máxima medición de energía.
   C. Obtener para un determinado sensor la media de sus mediciones de energía.
   D. Obtener para un determinado sensor la desviación estándar de sus mediciones de energía.
   E. Obtener para un determinado sensor el conteo de sus mediciones de energía.
   F. Obtener para un determinado sensor los valores sin duplicados que toman sus mediciones de energía.
3. Consultas de agrupación:
   A. Obtener para un determinado sensor la media de sus mediciones de energía por horas, semanas y meses.
   B. Obtener para un determinado sensor un intervalo en el que se encuentren los valores atípicos de sus mediciones de energía utilizando el rango intercuartílico.
4. Consultas basadas en rangos de tiempo:
   A. Obtener para un sensor determinado las mediciones de energía en un rango de tiempo determinado.

7.2. Resultados

En relación con la última métrica presentada en la sección anterior en relación con la latencia de las consultas, mostramos en la Tabla 7.1 la posibilidad o imposibilidad de realizar las consultas en las bases de datos elegidas. En la siguiente tabla podemos encontrarnos con el símbolo "✓" para las consultas que se completan correctamente, el símbolo "✓*" para las consultas que se completan pero presentan alguna diferencia con el resultado que consideramos correcto y el símbolo "✗" para las consultas que no se consigue realizar o para las que obtenemos una mínima parte.

También vemos en la tabla que la consulta 3.A y 4.A quedan a su vez subdivididas por el periodo de tiempo de la agrupación (días, semanas y meses) y por el rango de tiempo (un día, más de un día, un mes y más de un mes) en la consulta. Para cada base de datos buscamos encontrar la mejor manera de optimizar las pruebas con las opciones que nos ofrece cada una de ellas.

Consultas		**InfluxDB** A.2.1	**TimescaleDB** A.2.2	**OpenTSDB** A.2.3	**KairosDB** A.2.4
Punto exacto	1.A	✓	✓	✓	✓
Analíticas	2.A	✓	✓	✓	✓*
	2.B	✓	✓	✓	✓*
	2.C	✓	✓	✓*	✓*
	2.D	✓	✓	✓*	✓*
	2.E	✓	✓	✓	✓*
	2.F	✓	✓	✗	✗
Agrupación	3.A.1	✓	✓	✓	✓
	3.A.2	✓	✓	✓*	✓
	3.A.3	✓	✓	✓*	✓
	3.B	✗ (sólo un cuartil)	✓	✗ (sólo un cuartil)	✗ (sólo un cuartil)
Rangos de tiempo	4.A.1	✓	✓	✓	✓
	4.A.2	✓	✓	✓	✓
	4.A.3	✓	✓	✓	✓
	4.A.4	✓	✓	✓	✓

Tabla 7.1: Éxito de las consultas en las diferentes bases de datos

7.2.1. Carga de datos

La primera métrica que analizaremos será la de la ingesta de los datos presentados en la Sección 6.4 en las diferentes bases de datos. Ya que para cada tamaño de datos realizamos tres experimentos al preguntar por 1, 5 y 20 identificadores para la métrica de latencia de consultas, en la Tabla 7.2 podemos ver la media del tiempo en segundos que le lleva a cada base de datos almacenar cada uno de los tamaños.

En la Figura 7.1, centrándonos en las muestras que duplican más o menos su tamaño, podemos ver gráficamente los resultados y darnos cuenta de que TimescaleDB es la que mejor rendimiento de carga nos ofrece con mucha diferencia del resto de bases de datos. Podemos observar que todas las bases de datos presentan un factor de crecimiento geométrico y que OpenTSDB es la que peores resultados nos arroja al nivel de KairosDB a pesar de estar ambas específicamente diseñadas para manejar series temporales.

	Tamaño del conjunto de datos			
	3 meses	6 meses	10 meses	13 meses
InfluxDB	2.424	10.396	33.947	54.137
TimescaleDB	0.627	2.103	6.542	11.232
OpenTSDB	4.745	12.813	40.244	72.639
KairosDB	3.545	11.688	38.075	71.202

Tabla 7.2: Tiempos de carga de datos (en segundos)

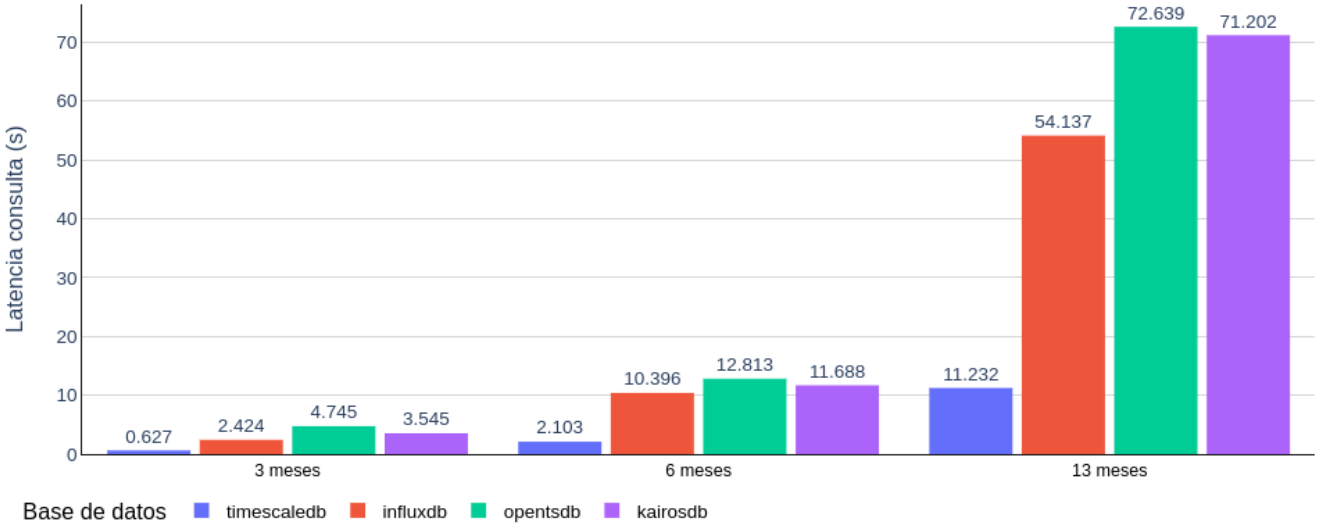


Figura 7.1: Gráfico de barras para los tiempos de carga de datos

En base a estos resultados y teniendo en cuenta que estamos comparando tres bases de datos NoSQL y una relacional (TimescaleDB), podemos ver que hay grandes diferencias entre ambos modelos de bases de datos y en términos de rendimiento de carga nos decantaríamos por una base de datos relacional ya que es medio orden de magnitud mejor que las demás. Dentro de las NoSQL InfluxDB obtiene tiempos competitivos, sobre todo para tamaños de conjunto de 3 y 13 meses, con respecto a OpenTSDB y KairosDB pero no frente a TimescaleDB.

Podríamos decir que la mejor opción es TimescaleDB, seguido de InfluxDB, KairosDB y OpenTSDB.

7.2.2. Tamaño de almacenamiento

La segunda métrica que examinaremos será la del espacio que ocupan los datos tras ser insertados en las bases de datos. Los resultados obtenidos en las pruebas de rendimiento los desglosaremos en tres tablas.

Por un lado, la Tabla 7.3 nos muestra el tamaño consumido por los datos medido en megabytes en su formato original (XLSX) y en los archivos adaptados a cada base de datos (JSON). De esta forma podemos analizar el coste que supone la transformación de los datos a un formato JSON debido a los campos específicos necesarios para la correcta inserción en cada base de datos. Vemos que todas las bases de datos consumen más espacio a nivel de JSON pero en el caso de TimescaleDB obtenemos el menor coste de transformación con gran diferencia y el peor con InfluxDB.

	Tamaño del conjunto de datos			
	3 meses	6 meses	10 meses	13 meses
XLSX	0.608	2.757	8.821	14.391
JSON InfluxDB	8.073	37.116	119.438	195.261
JSON TimescaleDB	6.770	31.116	100.121	163.688
JSON OpenTSDB	7.401	34.022	109.478	178.981
JSON KairosDB	7.809	35.897	115.514	188.847

Tabla 7.3: Espacio consumido por los archivos (MB archivo original vs. MB archivo adaptado)

Por otro lado, podemos también comparar el tamaño del archivo original con el tamaño de cada base de datos utilizando la Tabla 7.4 y ver que, KairosDB nos ofrece unos consumos de espacio un poco mayores pero muy similares al espacio que ocupan los datos del archivo original para casi todos los tamaños del conjunto de datos. También resalta que TimescaleDB es la que peor se comporta ofreciendo unos consumos de espacio muy altos en comparación con el tamaño de los datos en bruto y con el resto de bases de datos.

	Tamaño del conjunto de datos			
	3 meses	6 meses	10 meses	13 meses
XLSX	0.608	2.757	8.821	14.391
InfluxDB	1.593	4.696	12.054	18.671
TimescaleDB	13.788	26.930	64.205	98.403
OpenTSDB	2.084	10.751	32.105	43.816
KairosDB	1.816	4.129	9.786	14.957

Tabla 7.4: Eficiencia de almacenamiento (MB archivo original vs. MB base de datos)

Por último, podemos definir el factor de compresión de datos[40] como el cociente del tamaño de entrada dividido por el tamaño de salida, cuyo resultado si es menor que 1 implica que se ha producido una expansión y cuánto mayor sea de 1 será que la compresión es mejor:

$$\text{Factor de Compresión} = \frac{\text{Tamaño JSON}}{\text{Tamaño Base de Datos}}$$

En términos de rendimiento del consumo de espacio analizamos el factor de compresión de cada base de datos y nos encontramos con que, a partir de los datos de la Tabla 7.5, obtenemos factores de compresión bastante buenos y parecidos entre InfluxDB y KairosDB, siendo este último el que mejor resultados nos ofrece. TimescaleDB presenta las peores tasas de comprensión de datos, ya que como vemos en la tabla para el conjunto de datos de 3 meses se produce una expansión (los datos duplican su tamaño al insertarse) y aunque parece que mejora y si se comprimen los datos a medida que el volumen de estos aumenta no consigue hacer competencia a las demás bases de datos.

	Tamaño del conjunto de datos			
	3 meses	6 meses	10 meses	13 meses
JSON InfluxDB	8.073	37.116	119.438	195.261
InfluxDB	1.593	4.696	12.054	18.671
Factor compresión InfluxDB	5.1	7.9	9.9	10.5
JSON TimescaleDB	6.770	31.116	100.121	163.688
TimescaleDB	13.788	26.930	64.205	98.403
Factor compresión TimescaleDB	0.5	1.2	1.6	1.7
JSON OpenTSDB	7.401	34.022	109.478	178.981
OpenTSDB	2.084	10.751	32.105	43.816
Factor compresión OpenTSDB	3.5	3.2	3.4	4.1
JSON KairosDB	7.809	35.897	115.514	188.847
KairosDB	1.816	4.129	9.786	14.957
Factor compresión KairosDB	4.3	8.7	11.8	12.6

Tabla 7.5: Factor de compresión (MB archivo adaptado vs. MB base de datos)

En base a estos resultados, podemos destacar que las bases de datos NoSQL se comportan mucho mejor que la base de datos relacional en cuanto a la compactación de datos se refiere.

En general, podríamos decir que la mejor opción es KairosDB, seguido de InfluxDB, OpenTSDB y TimescaleDB. Esta última nos ofrecía unos tamaños de archivo JSON muy reducidos en comparación con las demás pero no es un precio que se deba pagar si el resultado va a ser una base de datos que ocupe mucho y que no comprima de forma eficiente los cientos de miles de datos que va a almacenar.

7.2.3. Latencia en las consultas

La tercera métrica que estudiaremos será la de la velocidad de respuesta de diferentes tipos de consultas basadas en rangos de tiempo, funciones de agregación o agrupaciones en las distintas bases de datos que tenemos. Es importante señalar y tener en cuenta antes de analizar los resultados que, como reflejábamos en la Tabla 7.1, hay bases de datos que al realizar algunas de las consultas que desarrollaremos a continuación presentan las siguientes peculiaridades:

- KairosDB: para las consultas analíticas en las que cuenta con todos los datos tiene que muestrear el conjunto de datos antes de aplicar la función de agregación y el máximo rango posible es de un año, es decir en nuestro caso mientras que otras bases de datos nos ofrecen una media para el conjunto de datos, KairosDB nos dará dos medias, una por cada año, para las mediciones de cada año diferente presente en el conjunto de datos. Además, la consulta en la que se obtiene el conteo de los datos, KairosDB devuelve el número de veces que cada medición está presente en el conjunto pero sin devolvernos el sumatorio, por lo que hay que realizarlo posteriormente.

- OpenTSDB: para las consultas analíticas en las que calcula la media o la desviación estándar presenta pequeñas diferencias de cálculo por el método utilizado con respecto a las demás bases de datos.

Consulta de punto exacto

Al realizar la consulta para recuperar la medición asociada a una marca de tiempo concreta, como vemos en la Tabla 7.6 el claro ganador es TimescaleDB con unos tiempos de respuesta muy inferiores al resto.

Si hablamos de los peores resultados entonces nos encontramos con que para los cuatro experimentos realizados InfluxDB, OpenTSDB y KairosDB empatan en el recuento de peores casos. InfluxDB comienza con unos tiempos que podrían ser aceptables y que parece que escalan linealmente a medida que se aumenta el tamaño y el número de variables consultadas, pero para grandes conjuntos de datos a partir de 5 variables consultadas se produce un aumento brusco de los tiempos de respuesta que llegan a superar por mucho a las demás como podemos ver en color rojo para un tamaño de 13 meses en la Figura 7.2. OpenTSDB y KairosDB vemos que tienen un comportamiento similar con unos tiempos de consulta similares y relativamente uniformes.

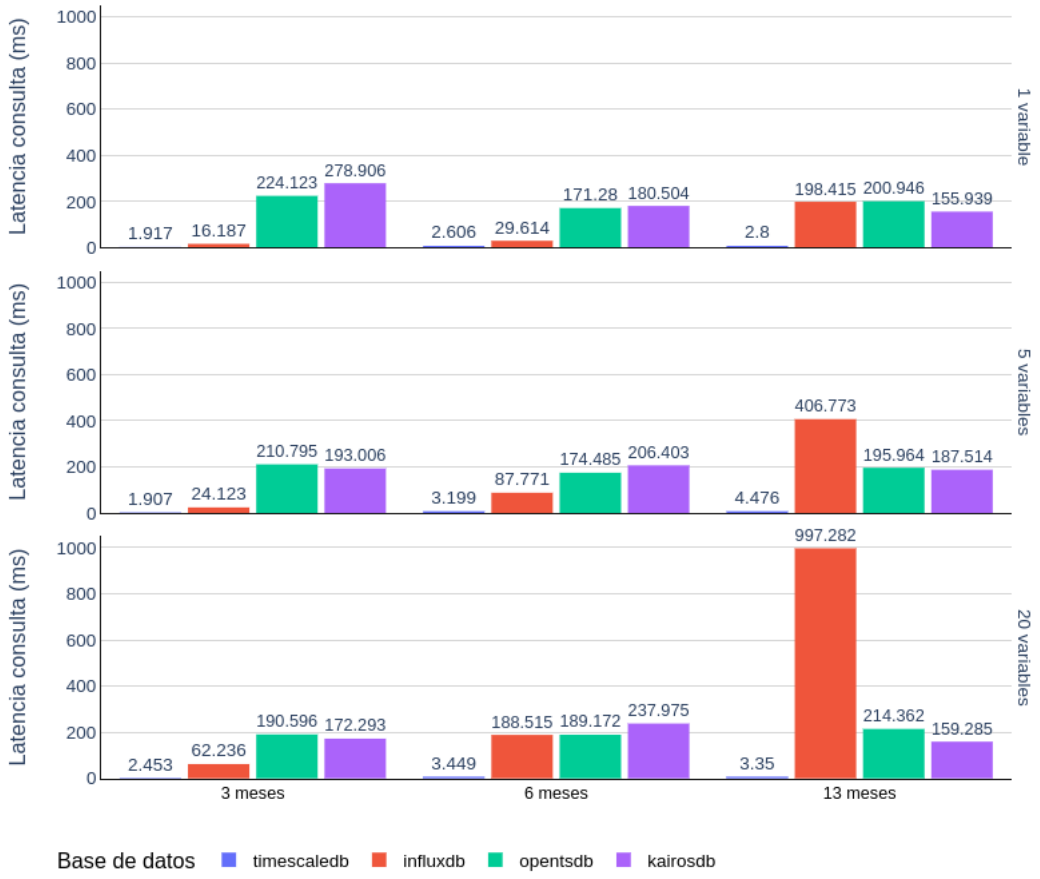


Figura 7.2: Gráfico de barras para la consulta 1 (punto exacto)

Si nos enfocamos en el número de variables consultadas no encontramos una relación clara entre tiempo de respuesta y número de variables ya que hay bases de datos como TimescaleDB e InfluxDB que con excepciones presentan un aumento esperable de la latencia de consulta frente al aumento del número de variables, pero hay otras como OpenTSDB que presentan una relación inversa entre estos dos criterios.

En base a estos resultados y para la consulta de obtener una medición de un punto exacto, nos decantaríamos por el modelo relacional ya que las diferencias en este caso con el modelo NoSQL es bastante grande.

Podríamos decir que la mejor opción es TimescaleDB, seguido de KairosDB, OpenTSDB e InfluxDB. Aclarar que le damos el tercer lugar a OpenTSDB ya que consideramos que es preferible tener unos tiempos de respuesta más o menos constantes antes que obtener unos resultados muy buenos solo bajo determinadas condiciones como es el caso de InfluxDB para esta consulta.

Consulta de agregación: Mínimo

Al realizar la consulta para obtener el mínimo de las mediciones de energía, como muestra la Tabla 7.7 TimescaleDB nos da los mejores tiempos.

En esta ocasión, nos encontramos con dos bases de datos que son candidatas a obtener el peor puesto y que podemos ver en la tabla, OpenTSDB y KairosDB. Tanto en la tabla como en la Figura 7.3 vemos que ambas presentan un factor de crecimiento geométrico y que ambas cuando se trata de conjuntos de datos reducidos se comportan de forma análoga. OpenTSDB se lleva el puesto de la peor en esta consulta ya que tiene 7/12 casos peores frente a los 5/12 de KairosDB y se aprecia que cuando más o menos duplica el tamaño del conjunto (pasando de 6 a 13 meses) casi se cuadruplica la latencia de esta consulta.

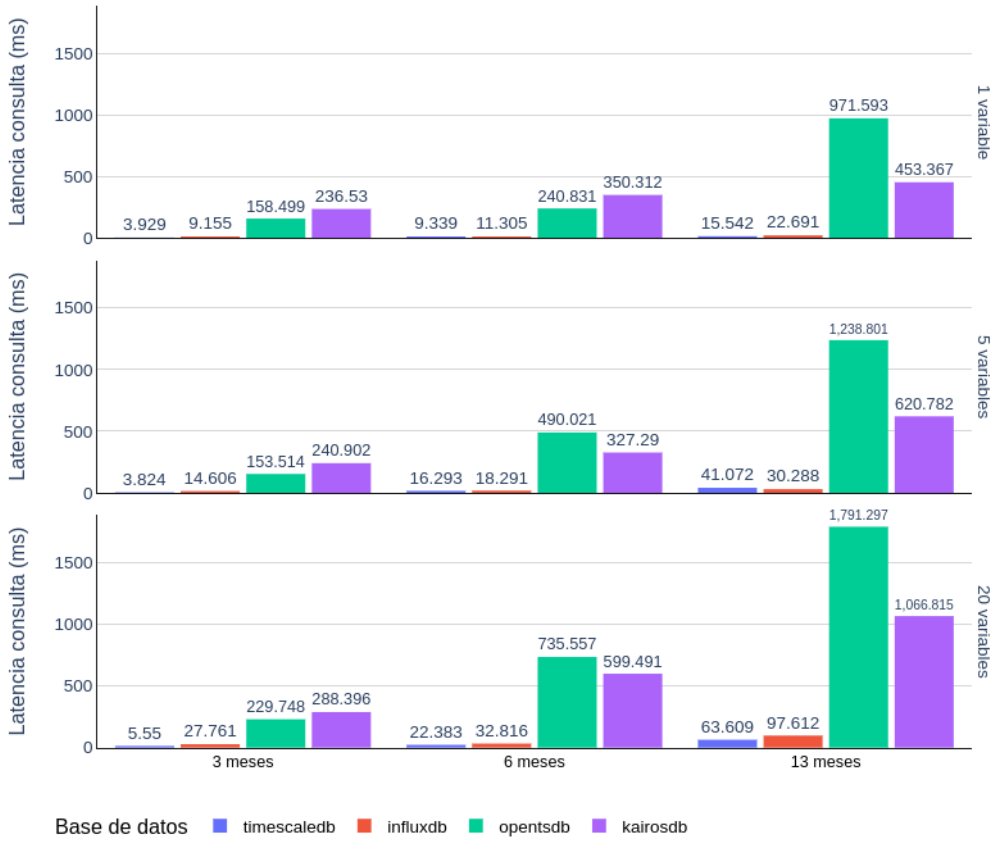


Figura 7.3: Gráfico de barras para la consulta 2 (mínimo)

Si nos enfocamos en el número de variables consultadas todas las bases de datos presentan

un aumento esperable de la latencia de consulta frente al aumento del número de variables consultadas.

En base a estos resultados, podemos decir que la base de datos de nuestras opciones que sigue un modelo relacional es la que mejor se comporta. Las opciones NoSQL son muchísimo peores aunque los resultados de InfluxDB serían aceptables siendo en ocasiones similares a la base de datos relacional. aunque si que es verdad que en este caso si lo comparamos con una de las opciones NoSQL no se ven grandes diferencias como en otras ocasiones.

Podríamos decir que la mejor opción es TimescaleDB, seguido de InfluxDB, KairosDB y OpenTSDB.

Consulta de agregación: Máximo

Al realizar la consulta para obtener el máximo de las mediciones de energía, como muestra la Tabla 7.8 TimescaleDB es la mejor opción y con bastante diferencia de cara a las demás bases de datos.

En la Figura 7.4 vemos que los peores tiempos de respuesta los consigue OpenTSDB con el mayor recuento de casos peores y con unos resultados que escalan por encima de la linealidad triplicándose los tiempos de respuesta al duplicarse el tamaño del conjunto. KairosDB también presenta malos resultados para el tamaño más pequeño del conjunto de datos en comparación con las demás, pero después se estabiliza y la latencia de consulta va aumentando según lo esperable aunque igualmente de forma muy superior a TimescaleDB e InfluxDB.

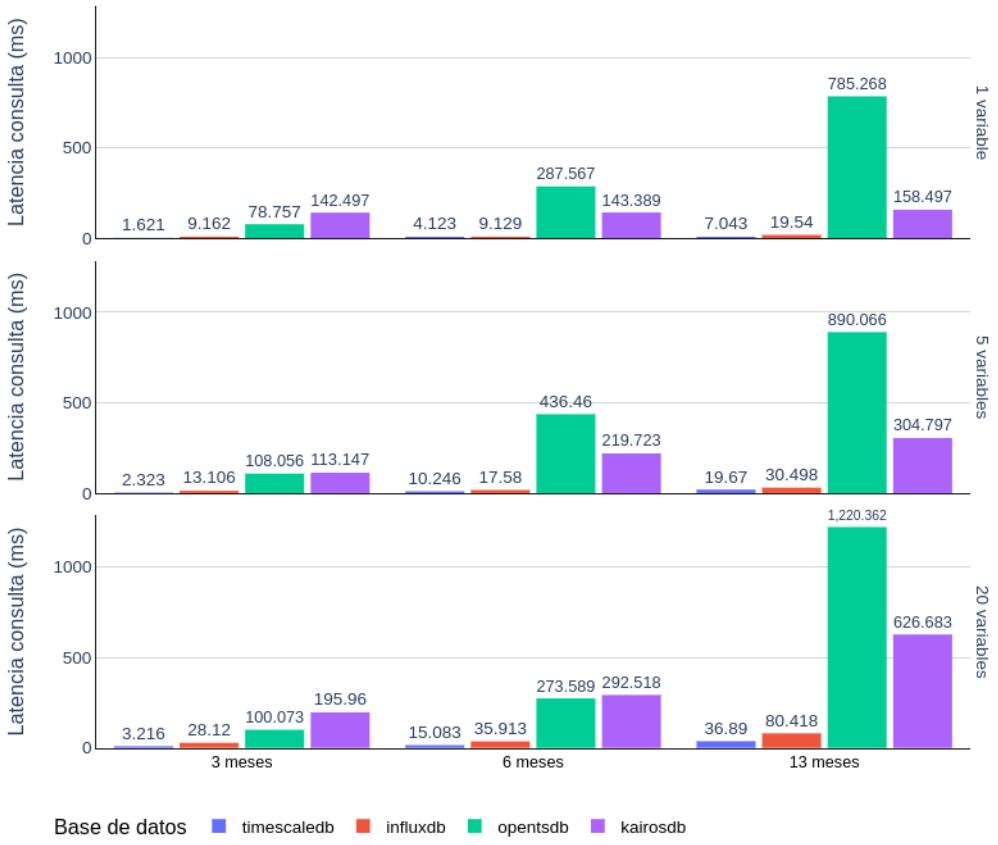


Figura 7.4: Gráfico de barras para la consulta 3 (máximo)

Si nos enfocamos en el número de variables consultadas todas las bases de datos presentan un aumento esperable de la latencia de consulta frente al aumento del número de variables consultadas.

Como era de esperar las consultas sobre máximo y mínimo obtienen similares rendimientos en todas las bases de datos y, por lo tanto, las conclusiones son las mismas que en el apartado anterior.

Podríamos decir que la mejor opción es TimescaleDB, seguido de InfluxDB, KairosDB y OpenTSDB.

Tam. del conjunto de datos	3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas	1	5	20	1	5	20	1	5	20	1	5	20
InfluxDB	16.187	24.123	62.236	29.614	87.771	188.515	96.904	310.859	580.922	198.415	406.773	997.282
TimescaleDB	1.917	1.907	2.453	2.606	3.199	3.449	2.663	3.325	4.084	2.800	4.476	3.35
OpenTSDB	224.123	210.795	190.596	171.280	174.485	189.172	197.927	218.274	208.694	200.946	195.964	214.362
KairosDB	278.906	193.006	172.293	180.504	206.403	237.975	190.404	171.495	172.516	155.939	187.514	159.285

Tabla 7.6: Tiempos para obtener la medición de un día y hora determinadas (en milisegundos)

T. del conjunto de datos	3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas	1	5	20	1	5	20	1	5	20	1	5	20
InfluxDB	9.155	14.606	27.761	11.305	18.291	32.816	16.251	28.714	52.277	22.691	30.288	97.612
TimescaleDB	3.929	3.824	5.550	9.339	16.293	22.383	11.803	23.499	42.863	15.542	41.072	63.609
OpenTSDB	158.499	153.514	229.748	240.831	490.021	735.557	759.105	792.984	1056.082	971.593	1238.801	1791.297
KairosDB	236.530	240.902	288.396	350.312	327.290	599.491	498.287	602.557	1276.288	453.367	620.782	1066.815

Tabla 7.7: Tiempos para obtener el mínimo de las mediciones (en milisegundos)

T. del conjunto de datos	3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas	1	5	20	1	5	20	1	5	20	1	5	20
InfluxDB	9.162	13.106	28.120	9.129	17.580	35.913	13.565	27.697	52.268	19.540	30.498	80.418
TimescaleDB	1.621	2.323	3.216	4.123	10.246	15.083	4.725	15.412	32.430	7.043	19.670	36.890
OpenTSDB	78.757	108.056	100.073	287.567	436.460	273.589	407.132	482.844	626.799	785.268	890.066	1220.362
KairosDB	142.497	113.147	195.960	143.389	219.723	292.518	316.793	321.322	457.044	158.497	304.797	626.683

Tabla 7.8: Tiempos para obtener el máximo de las mediciones (en milisegundos)

Consulta de agregación: Media aritmética

Al realizar la consulta para obtener la media aritmética de las mediciones de energía, como muestra la Tabla 7.9 TimescaleDB nos brinda los mejores resultados con diferencia.

Tanto en la tabla como en la Figura 7.5 queda claro que los peores resultados pertenecen a OpenTSDB, obtiene un recuento de peores casos de 9/12, ocurriendo un poco lo que pasaba en la consulta anterior ya que los tiempos se disparan al aumentar el tamaño del conjunto. En este caso, KairosDB también tiene malos resultados para el tamaño del conjunto de datos más pequeño aunque su crecimiento a partir de un tamaño de 6 meses es mucho más lento que el que experimenta OpenTSDB.

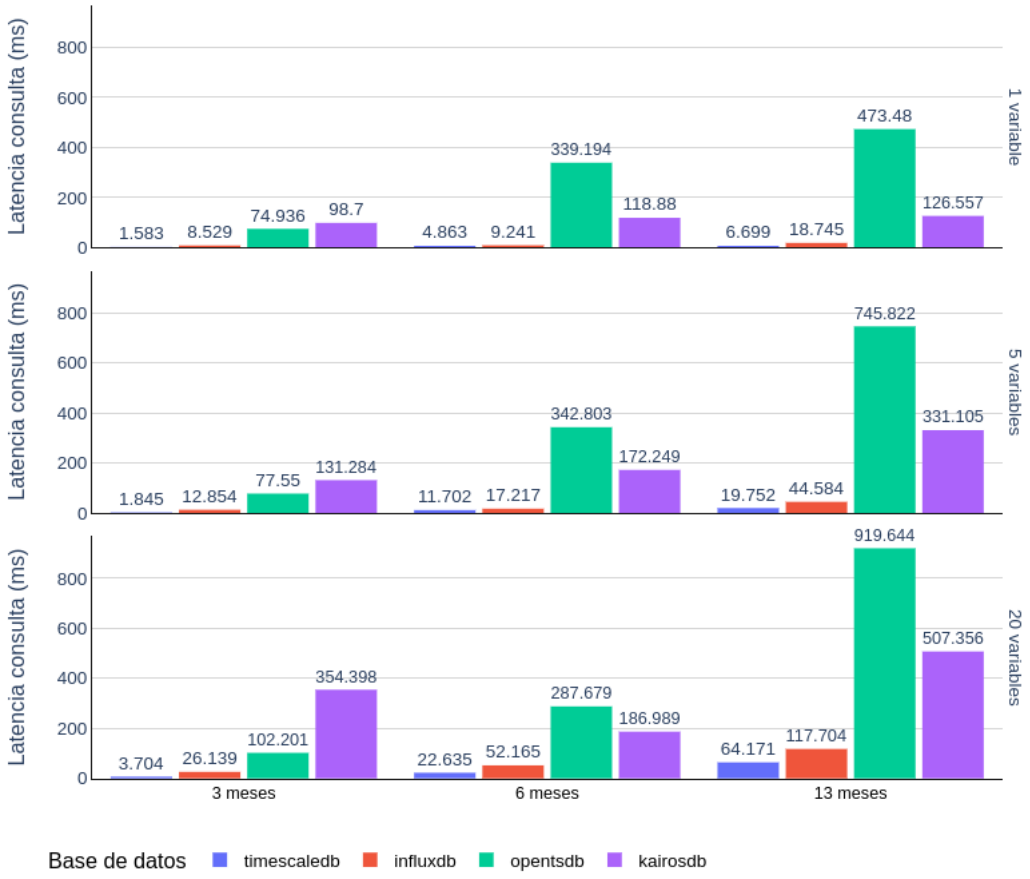


Figura 7.5: Gráfico de barras para la consulta 4 (media aritmética)

Si nos enfocamos en el número de variables consultadas todas las bases de datos presentan un aumento esperable de la latencia de consulta frente al aumento del número de variables consultadas.

En base a los resultados obtenidos, el modelo relacional resulta la mejor opción ya que al igual que en los dos casos anteriores cuándo se necesita recorrer todos los valores del conjunto de datos los rendimientos son similares.

Podríamos decir que la mejor opción es TimescaleDB, seguido de InfluxDB, KairosDB y OpenTSDB.

Consulta de agregación: Desviación estándar

Al realizar la consulta para obtener la desviación estándar de las mediciones de energía, como muestra la Tabla 7.10 TimescaleDB nuevamente nos brinda los mejores resultados con diferencia del resto de bases de datos.

OpenTSDB es la que peores resultados nos ofrece ya que son latencias de consulta mucho más lentas que el de las demás bases de datos. KairosDB, al igual que en otras consultas, repite el mismo comportamiento ya que a partir de un tamaño de conjunto de 6 meses los tiempos se estabilizan y crecen dentro de lo esperable en la línea que sigue esta base de datos.

Si nos enfocamos en el número de variables consultadas por lo general todas las bases de datos presentan un aumento esperable de la latencia de consulta frente al aumento del número de variables consultadas como podemos observar en la Figura 7.6.

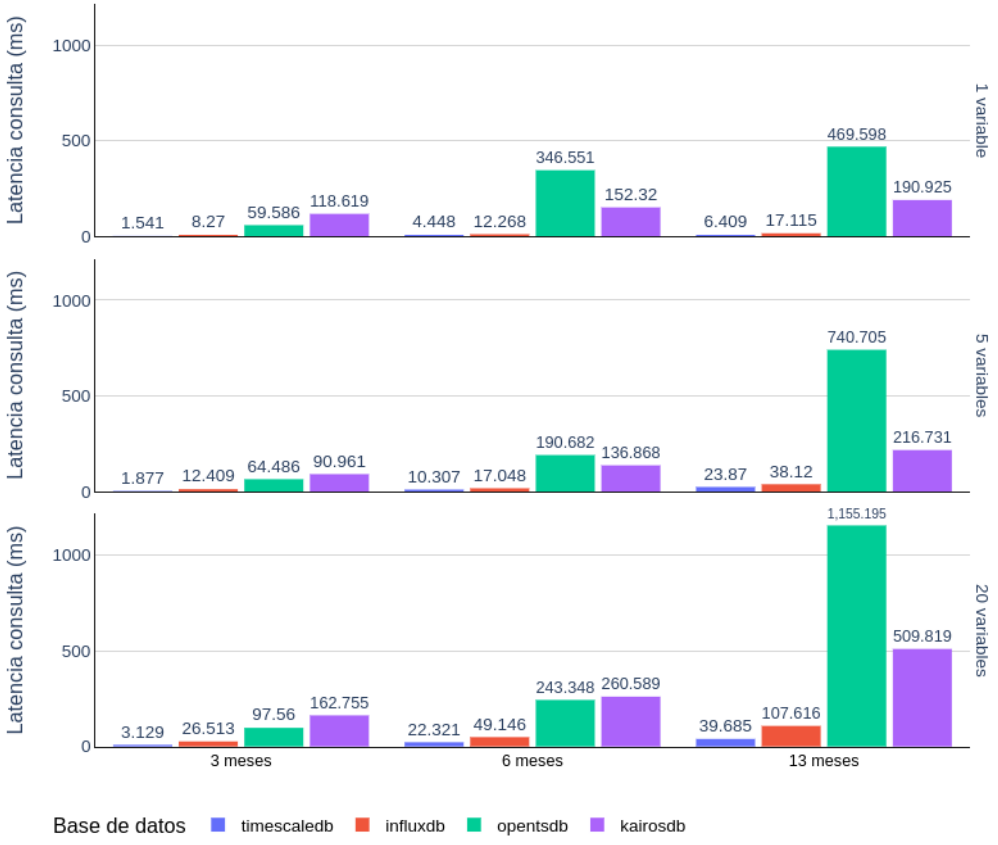


Figura 7.6: Gráfico de barras para la consulta 5 (desviación estándar)

En base a los resultados obtenidos y como era de esperar obtenemos resultados idénticos a las 3 consultas anteriores dónde una vez más el modelo relacional es el que ofrece mejores resultados y dentro de los modelos NoSQL con los que contamos vemos que InfluxDB se comporta bastante bien pero que las otras dos bases de datos que aplican este modelo sobrepasan muy por encima los resultados de sus competidores.

Podríamos decir que la mejor opción es TimescaleDB, seguido de InfluxDB, KairosDB y OpenTSDB.

Consulta de agregación: Conteo

Al realizar la consulta para obtener el conteo de las mediciones de energía, TimescaleDB es la base de datos que mejores tiempos de respuesta registra como se puede ver en la Tabla 7.11.

OpenTSDB es la que peores resultados arroja ya que son tiempos de respuesta muy altos en comparación con el resto, sobretodo cuando se trata de grandes conjuntos de datos y se consultan un gran número de variables. En la Figura 7.7 también podemos ver que KairosDB presenta unos tiempos bastante altos y un comportamiento muy irregular. Hay que tener en cuenta a la hora de compararla con las demás bases de datos que no es capaz de darnos el resultado final de manera inmediata, siendo necesario procesar el resultado de la consulta para obtener el conteo total (el tiempo de procesamiento no se incluye en la medición de la latencia de esta consulta).

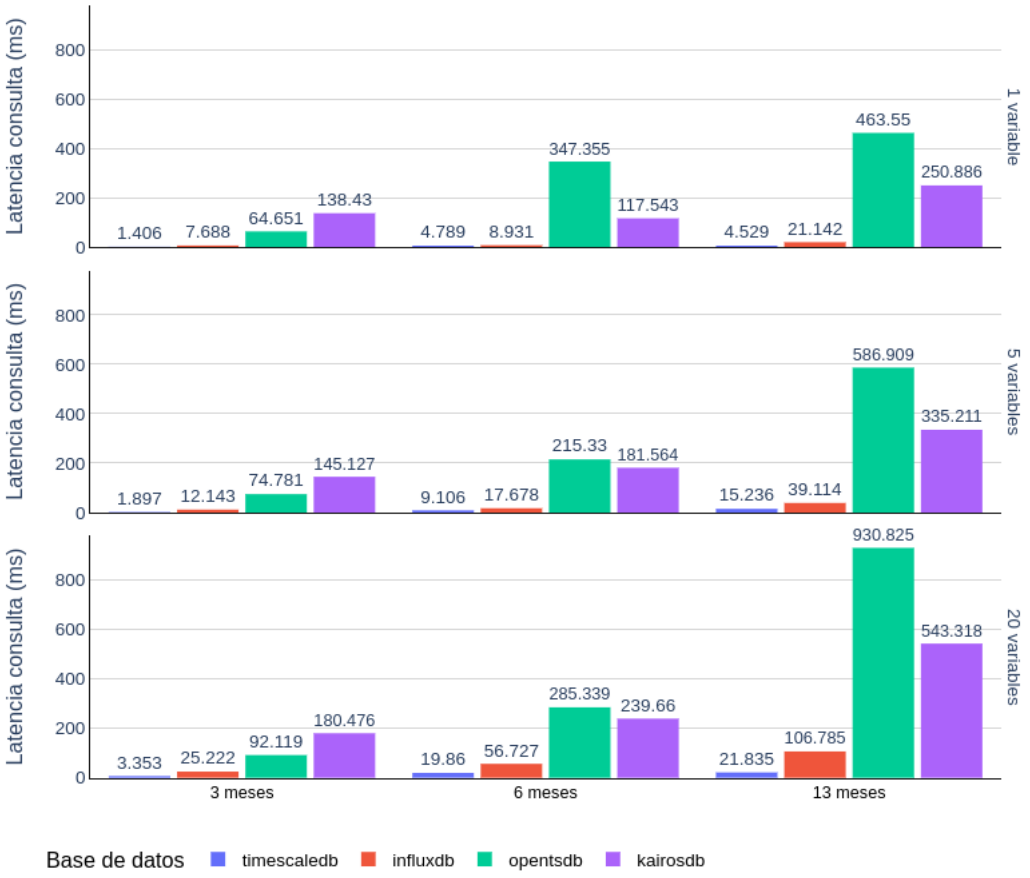


Figura 7.7: Gráfico de barras para la consulta 6 (conteo)

Si nos enfocamos en el número de variables consultadas por lo general todas las bases de datos presentan un aumento esperable de la latencia de consulta frente al aumento del número de variables consultadas.

Como era de suponer nos encontramos de nuevo con que los mejores resultados pertenecen al modelo relacional y que InfluxDB sigue presentando buenos resultados dentro de las opciones NoSQL.

Podríamos decir que la mejor opción es TimescaleDB, seguido de InfluxDB, KairosDB y OpenTSDB. KairosDB obtiene el penúltimo lugar a pesar de no completar la consulta ya que consideramos que es preferible pagar el coste de realizar un pequeño procesamiento cuyo tiempo añadido al de la consulta que tenemos no va a superar el tiempo de latencia de esta consulta en OpenTSDB.

T. del conjunto de datos	3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas	1	5	20	1	5	20	1	5	20	1	5	20
InfluxDB	8.529	12.854	26.139	9.241	17.217	52.165	15.292	34.845	87.564	18.745	44.584	117.704
TimescaleDB	1.583	1.845	3.704	4.863	11.702	22.635	7.095	15.626	31.292	6.699	19.752	64.171
OpenTSDB	74.936	77.550	102.201	339.194	342.803	287.679	354.987	556.243	542.373	473.480	745.822	919.644
KairosDB	98.700	131.284	354.398	118.880	172.249	186.989	123.529	236.830	535.306	126.557	331.105	507.356

Tabla 7.9: Tiempos para obtener la media de las mediciones (en milisegundos)

T. del conjunto de datos	3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas	1	5	20	1	5	20	1	5	20	1	5	20
InfluxDB	8.270	12.409	26.513	12.268	17.048	49.146	14.024	33.096	69.046	17.115	38.120	107.616
TimescaleDB	1.541	1.877	3.129	4.448	10.307	22.321	6.519	14.582	32.435	6.409	23.870	39.685
OpenTSDB	59.586	64.486	97.560	346.551	190.682	243.348	312.998	529.716	632.653	469.598	740.705	1155.195
KairosDB	118.619	90.961	162.755	152.320	136.868	260.589	167.763	194.092	336.965	190.925	216.731	509.819

Tabla 7.10: Tiempos para obtener la desviación estándar de las mediciones (en milisegundos)

T. del conjunto de datos	3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas	1	5	20	1	5	20	1	5	20	1	5	20
InfluxDB	7.688	12.143	25.222	8.931	17.678	56.727	15.479	23.489	54.094	21.142	39.114	106.785
TimescaleDB	1.406	1.897	3.353	4.789	9.106	19.860	3.925	13.847	21.819	4.529	15.236	21.835
OpenTSDB	64.651	74.781	92.119	347.355	215.330	285.339	313.970	515.935	673.512	463.550	586.909	930.825
KairosDB*	138.430	145.127	180.476	117.543	181.564	239.660	129.845	473.024	434.686	250.886	335.211	543.318

Tabla 7.11: Tiempos para obtener el conteo de las mediciones (en milisegundos)

Consulta de agregación: Valores únicos

Al realizar la consulta para obtener los valores únicos que toman las mediciones de energía sólo podemos comparar InfluxDB y TimescaleDB ya que en las dos otras bases de datos no se puede realizar esta consulta. En la Tabla 7.12 podemos ver que TimescaleDB es con gran diferencia mejor que InfluxDB.

La peor opción, por descarte y también porque está justificado, es InfluxDB ya que vemos que a parte de ofrecernos unos tiempos exageradamente grandes a medida que aumenta el tamaño del conjunto de datos el tiempo de respuesta de la consulta se dispara llegando a ser en ocasiones dos órdenes de magnitud peor que TimescaleDB.

Si nos enfocamos en el número de variables consultadas nos encontramos con que ambas bases de datos reflejan un crecimiento exponencial del tiempo de respuesta respecto al aumento del número de variables consultadas que podemos ver en la Figura 7.8.

En base a los resultados obtenidos, definitivamente el modelo relacional nos ofrece mejores resultados que el modelo NoSQL.

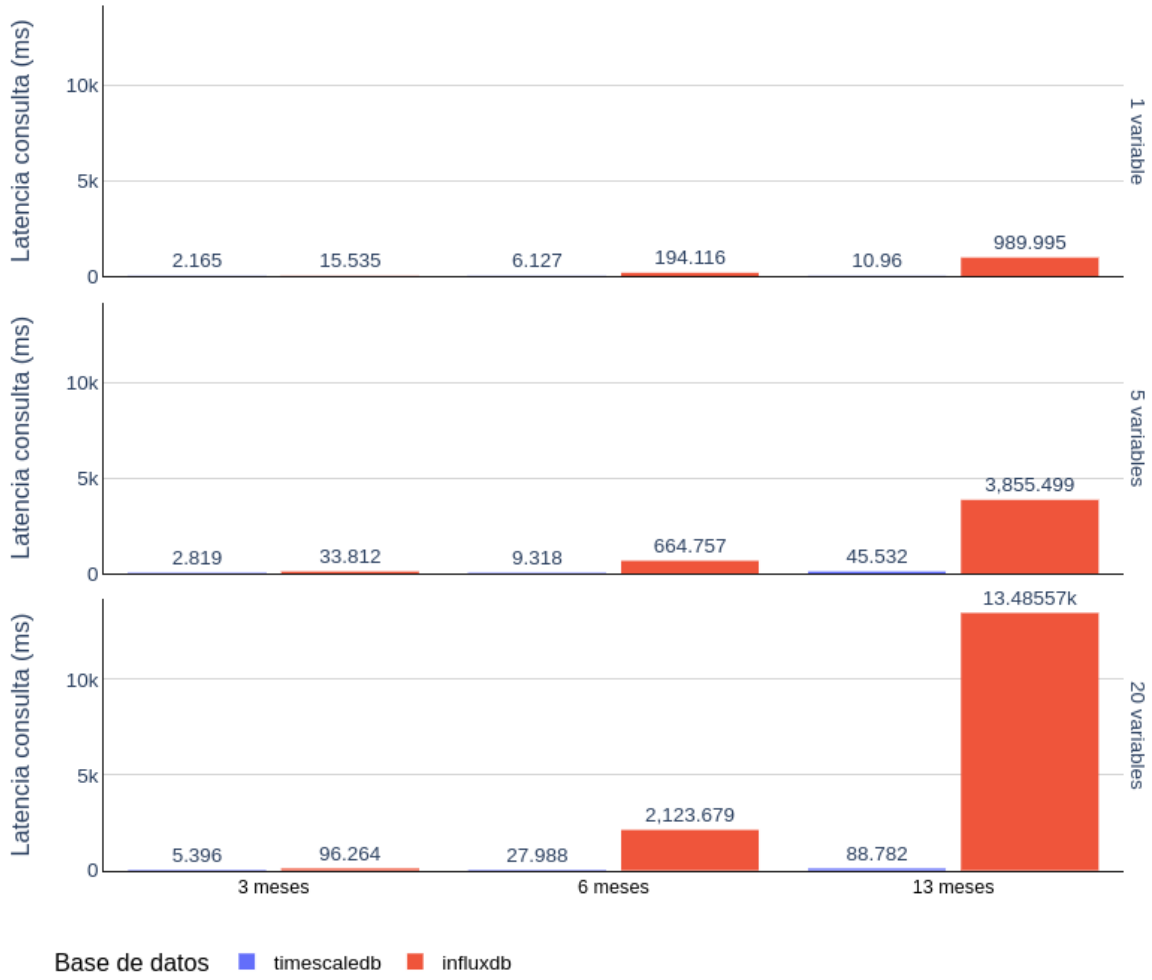


Figura 7.8: Gráfico de barras para la consulta 7 (valores únicos)

Consulta de agrupación: Horas, semanas y meses

Al realizar las consultas de agrupación probamos con diferentes tamaños de ventana de tiempo para ver cómo afecta esta nueva variable a la latencia de las consultas y conocer que nivel de granularidad de agrupación es más eficiente para cada base de datos.

En la Figura 7.9 podemos ver que para la consulta que agrupa en intervalos de una hora y calcula la media, TimescaleDB es la base de datos que mejores resultados nos da mostrando grandes diferencias con respecto de las demás. También vemos en color rojo que InfluxDB es la que peor comportamiento tiene tanto al aumentar el tamaño del conjunto de datos como el número de variables consultadas, presentando un factor de crecimiento exponencial.

En la Figura 7.10 observamos que para la consulta que agrupa en intervalos de una semana y calcula la media, nuevamente tenemos que la mejor es TimescaleDB y en este caso, OpenTSDB es la peor ya que consigue mayor número de peores casos y unos tiempos muy altos especialmente en conjuntos de datos grandes y para la recuperación de 1 y 5 variables. También destaca que KairosDB es la que peor se comporta para un tamaño de datos pequeño e InfluxDB cuando se consultan 20 variables. De forma general en esta consulta todas mejoran sus tiempos con respecto a la anterior siendo bastante más rápidas agrupando en semanas que en días.

En la Figura 7.11 que corresponde a la consulta con la que agrupamos los datos en meses y calculamos la media, vemos que de nuevo la que mejor rendimiento nos ofrece frente a las demás es TimescaleDB. En este caso, la base de datos que peores resultados da es OpenTSDB con 7/12 peores casos ya que aunque tenga un comportamiento aceptable para un conjunto de datos pequeño, cuando estos aumentan de tamaño la latencia de consulta crece considerablemente.

Desde un punto de vista general, en la Tabla 7.13 se puede ver que TimescaleDB en la que mejor se comporta es en la consulta de agrupación por meses ya que coinciden la granularidad de la base de datos que habíamos fijado en meses con el periodo de agrupación de la consulta, que InfluxDB ofrece unos resultados muy malos para la consulta de 20 variables y que no se comporta bien en la agrupación por horas pero mejora mucho al ampliar este rango, y que, salvo alguna excepción concreta todas las bases de datos mejoran el rendimiento de la consulta cuando se utiliza una ventana de tiempo del tamaño de un mes.

En base a los resultados obtenidos, el modelo relacional nos ofrece mejores resultados que el modelo NoSQL.

Podríamos decir que la mejor opción es TimescaleDB, seguido de KairosDB, OpenTSDB e InfluxDB. Aclarar que le otorgamos el segundo puesto a KairosDB ya que al compararlo detalladamente con OpenTSDB e InfluxDB, vemos que obtiene un recuento menor de peores casos.

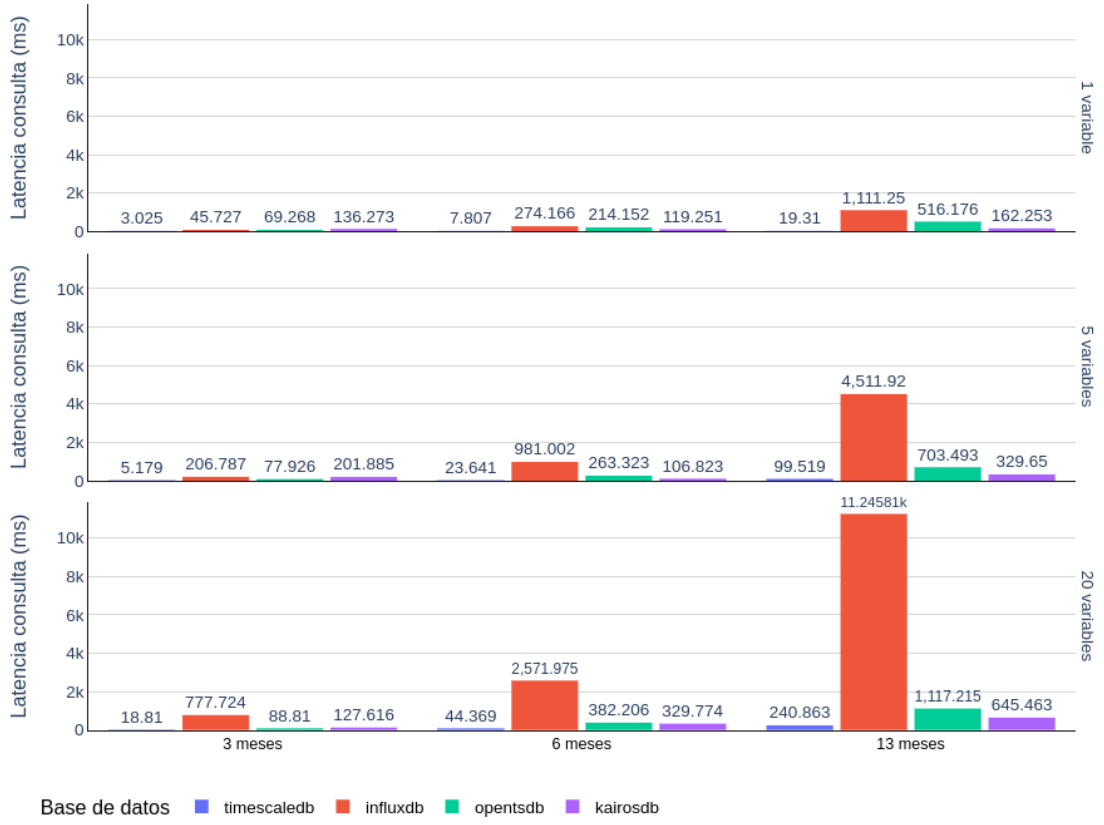


Figura 7.9: Gráfico de barras para la consulta 8_1 (horas)

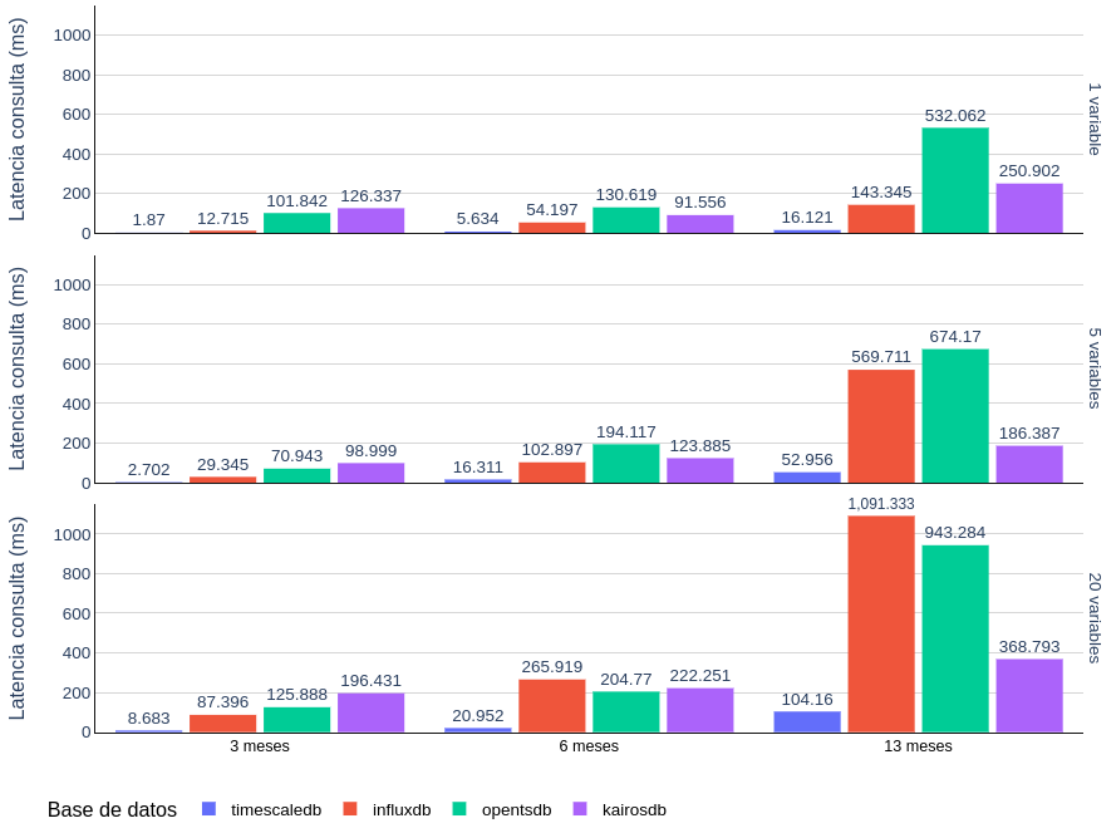


Figura 7.10: Gráfico de barras para la consulta 8_2 (semanas)

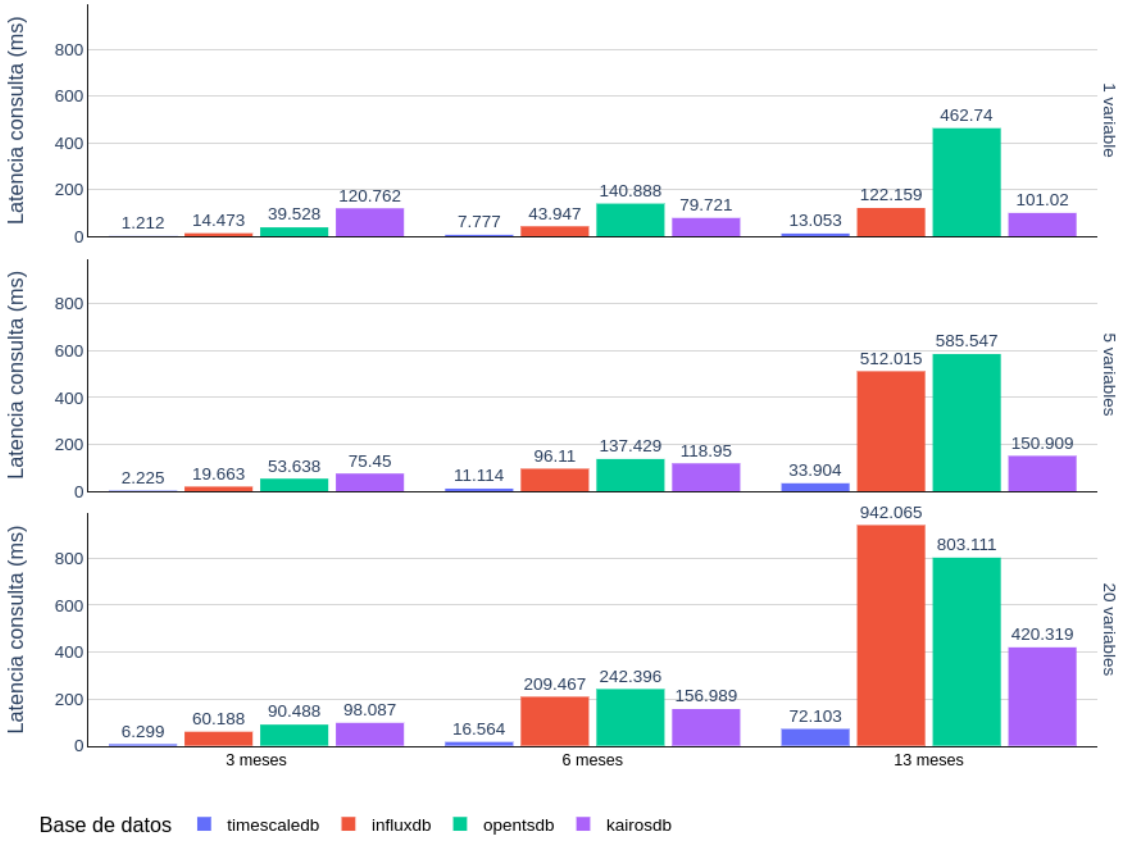


Figura 7.11: Gráfico de barras para la consulta 8_3 (meses)

T. del conjunto de datos	3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas	1	5	20	1	5	20	1	5	20	1	5	20
InfluxDB	15.535	33.812	96.264	194.116	664.757	2123.679	566.894	2580.693	7510.281	989.995	3855.499	13485.565
TimescaleDB	2.165	2.819	5.396	6.127	9.318	27.988	8.917	33.129	67.773	10.960	45.532	88.782
OpenTSDB	✗											
KairosDB	✗											

Tabla 7.12: Tiempos para obtener los valores únicos de las mediciones (en milisegundos)

T. del conjunto de datos		3 meses			6 meses			10 meses			13 meses		
Nº de variables consultadas		1	5	20	1	5	20	1	5	20	1	5	20
H.	**InfluxDB**	45.727	206.787	777.724	274.166	981.002	2571.975	747.082	2649.016	7277.769	1111.250	4511.920	11245.812
	TimescaleDB	3.025	5.179	18.810	7.807	23.641	44.369	14.161	76.099	157.194	19.310	99.519	240.863
	OpenTSDB	69.268	77.926	88.810	214.152	263.323	382.206	340.135	585.254	681.249	516.176	703.493	1117.215
	KairosDB	136.273	201.885	127.616	119.251	106.823	329.774	129.131	213.476	426.405	162.253	329.650	645.463
S.	**InfluxDB**	12.715	29.345	87.396	54.197	102.897	265.919	88.245	288.219	828.281	143.345	569.711	1091.333
	TimescaleDB	1.870	2.702	8.683	5.634	16.311	20.952	11.051	39.197	57.360	16.121	52.956	104.160
	OpenTSDB	101.842	70.943	125.888	130.619	194.117	204.770	281.685	449.023	555.888	532.062	674.170	943.284
	KairosDB	126.337	98.999	196.431	91.556	123.885	222.251	94.342	144.389	279.671	250.902	186.387	368.793
M.	**InfluxDB**	14.473	19.663	60.188	43.947	96.110	209.467	108.011	265.938	595.696	122.159	512.015	942.065
	TimescaleDB	1.212	2.225	6.299	7.777	11.114	16.564	11.636	25.455	43.141	13.053	33.904	72.103
	OpenTSDB	39.528	54.638	90.488	140.888	137.429	242.396	277.439	470.951	518.373	462.740	585.547	803.111
	KairosDB	120.762	75.450	98.087	79.721	118.950	156.989	77.556	134.103	205.703	101.020	150.909	420.319

Tabla 7.13: Tiempos para obtener la media de las mediciones tras agrupar en rangos de tiempo (en milisegundos)

Consulta de agrupación: Intervalos de outliers o tercer cuartil

Al realizar esta consulta, es necesario aclarar que estaba pensada desde un principio para conseguir intervalos de tiempo donde las mediciones de energía no estuviesen en un rango definido, es decir el objetivo era poder conseguir con una consulta aquellas mediciones y marcas de tiempo asociadas que nos informasen de un mal comportamiento del analizador de red seleccionado. Como se puede ver en la Tabla 7.14, una vez que comenzamos a trabajar con las diferentes bases de datos pudimos comprobar que la consulta inicial sólo era capaz de realizarla TimescaleDB así que para las demás bases de datos llegamos hasta donde nos permitía cada una.

Obviamente TimescaleDB es la que mejor rendimiento nos ofrece completando la consulta y arrojando buenos tiempos. Para comparar las demás bases de datos contamos con la Figura 7.12 en las que realizamos una consulta para conseguir el tercer cuartil utilizado por el método del rango intercuartílico[41] para obtener intervalos de valores atípicos. InfluxDB es la que mejores tiempos nos ofrece en comparación con las demás y OpenTSDB la que peor obteniendo 8/12 peores casos.

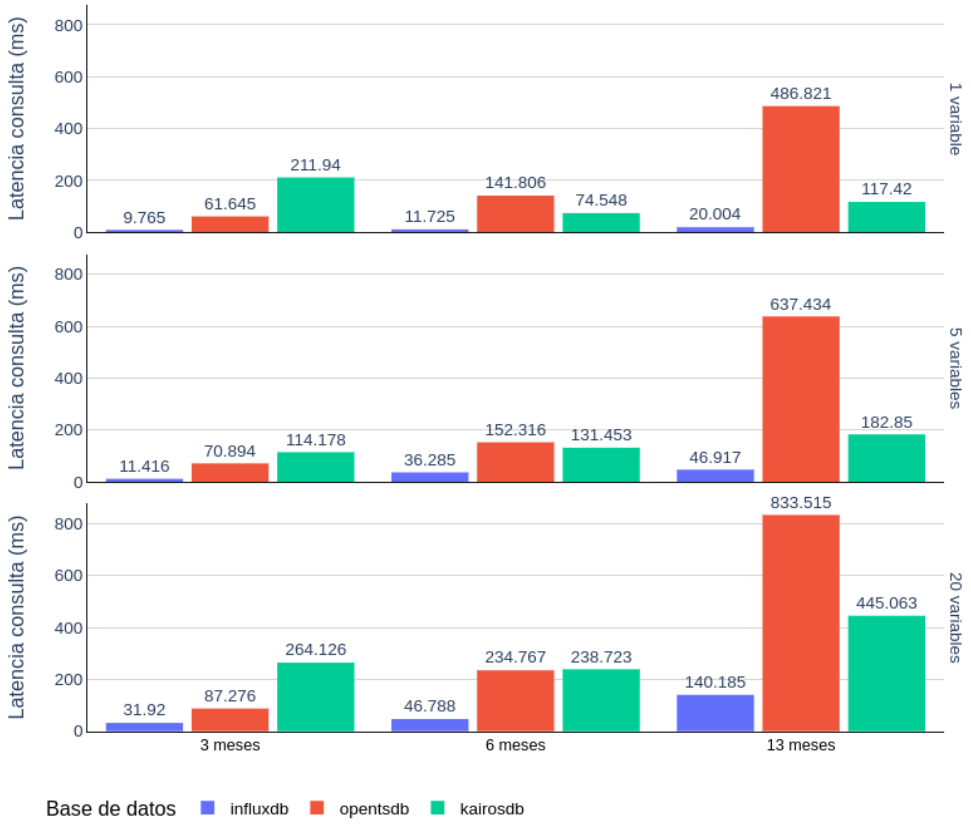


Figura 7.12: Gráfico de barras para la consulta 9 (tercer cuartil)

Vemos que las tres bases de datos presentan una latencia de consulta con un crecimiento exponencial que es más acentuado en OpenTSDB. Por otro lado, si nos enfocamos en el número de variables consultadas nos encontramos en la Tabla 7.14 con que todas las bases de datos presentan un aumento lógico de la latencia de consulta al aumentar el número de variables consultadas.

En base a los resultados obtenidos, podríamos decir que el modelo relacional nos ofrece mejores resultados que el modelo NoSQL, ya que este último no nos ha permitido realizar la consulta deseada.

Podríamos decir que la mejor opción es TimescaleDB, seguido de InfluxDB, KairosDB y OpenTSDB.

Consulta de rangos de tiempo: Un día, más de un día, un mes y más de un mes

Una de las consultas esenciales en el ámbito de los datos de series temporales es la de recuperar datos a partir de intervalos temporales específicos. Con la intención de ver como afecta la variación temporal de los rangos en las distintas bases de datos decidimos establecer consultas para cuatro rangos de tiempo diferentes cuyos resultados podemos encontrar en la Tabla 7.15.

En la Figura 7.13 podemos ver los resultados de la consulta que solicita las mediciones de energía para el periodo de un día completo. La mejor base de datos en esta consulta es TimescaleDB con tiempos casi constantes de uno o dos milisegundos tanto con el aumento del tamaño del conjunto de datos como con el del número de variables consultadas. KairosDB es la que peores tiempos de respuesta arroja superando en gran medida a sus competidores además de presentar un comportamiento muy irregular y también destaca que InfluxDB parece ser sensible al aumento del número de variables consultadas ya que sus tiempos al aumentar el tamaño del conjunto no cambian mucho pero si nos fijamos en la gráfica al pasar de 1 a 5 y a 20 variables los tiempos se disparan.

En la Figura 7.14 observamos los tiempos de latencia tras ampliar un poco el rango de tiempo con respecto a la consulta anterior, en este caso queremos recuperar mediciones de energía en el periodo de tiempo de un día y unas horas del día siguiente. La mejor base de datos en esta consulta es TimescaleDB y podemos ver una mejoría de los tiempos en relación con la anterior consulta. KairosDB repite como la base de datos que peores tiempos nos ofrece pero en este caso, como refleja también la Tabla 7.15, los tiempos no presentan un crecimiento brusco en relación con aumento del número de variables consultadas.

En la Figura 7.15 vemos los resultados de la consulta que solicita las mediciones de energía para el periodo de un mes completo. TimescaleDB continúa siendo la mejor opción aunque para este rango de tiempo parece que influye más el número de variables solicitadas en la latencia de la consulta, además de presentar un comportamiento extraño cuando se trata de 20 variables ya que al aumentar el tamaño del conjunto los tiempos de consulta se reducen y es algo que no ocurre para 1 o 5 variables. Para esta consulta la base de datos que peor se comporta es InfluxDB con tiempos similares al resto de bases de datos cuando se solicita una variable pero muy superiores al resto para el resto de casos, dónde se ve que los tiempos de latencia escalan geométricamente al aumentar el número de variables consultadas.

Por último, contamos con la consulta con la que recuperamos los datos para un rango temporal de un mes y unas semanas. En la Figura 7.16 podemos observar que TimescaleDB es la mejor y sigue mostrándose sensible en cuanto al número de variables consultadas a medida que crece el periodo de tiempo consultado. InfluxDB repite como la peor base de datos para esta consulta ya que nos ofrece unos tiempos de respuesta muy altos y muy dispares con respecto a las demás.

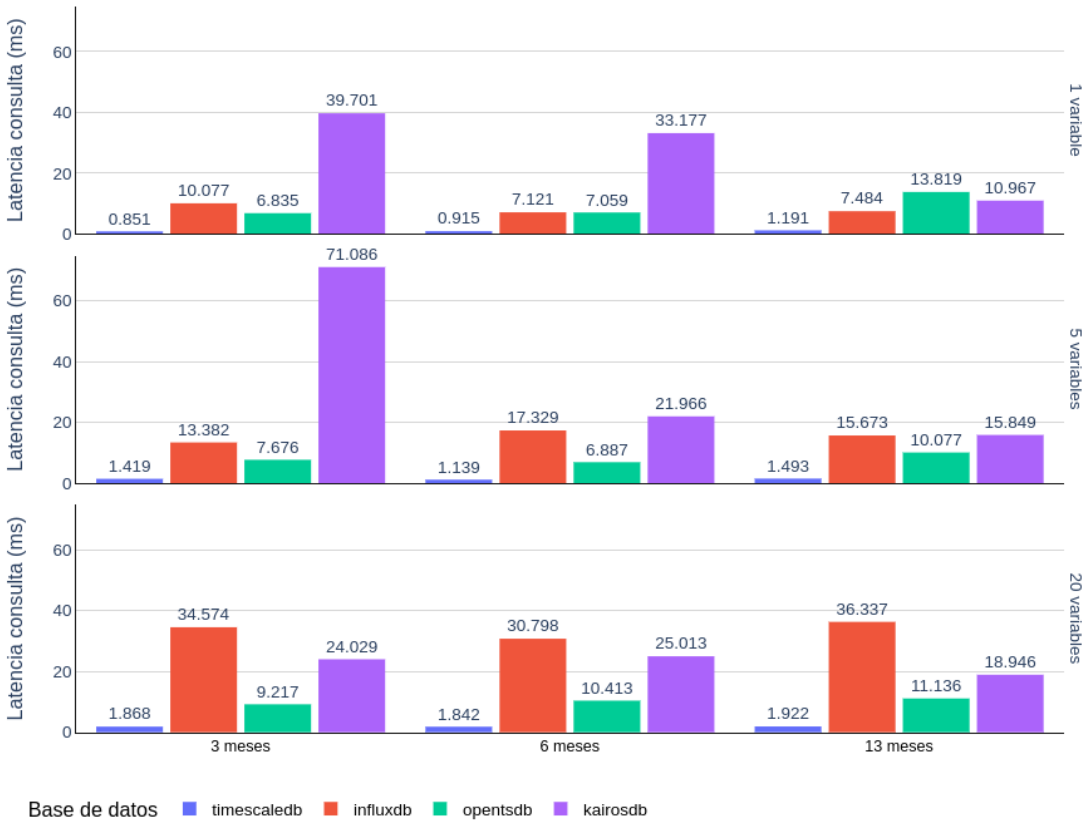


Figura 7.13: Gráfico de barras para la consulta 10_1 (1 día exacto)

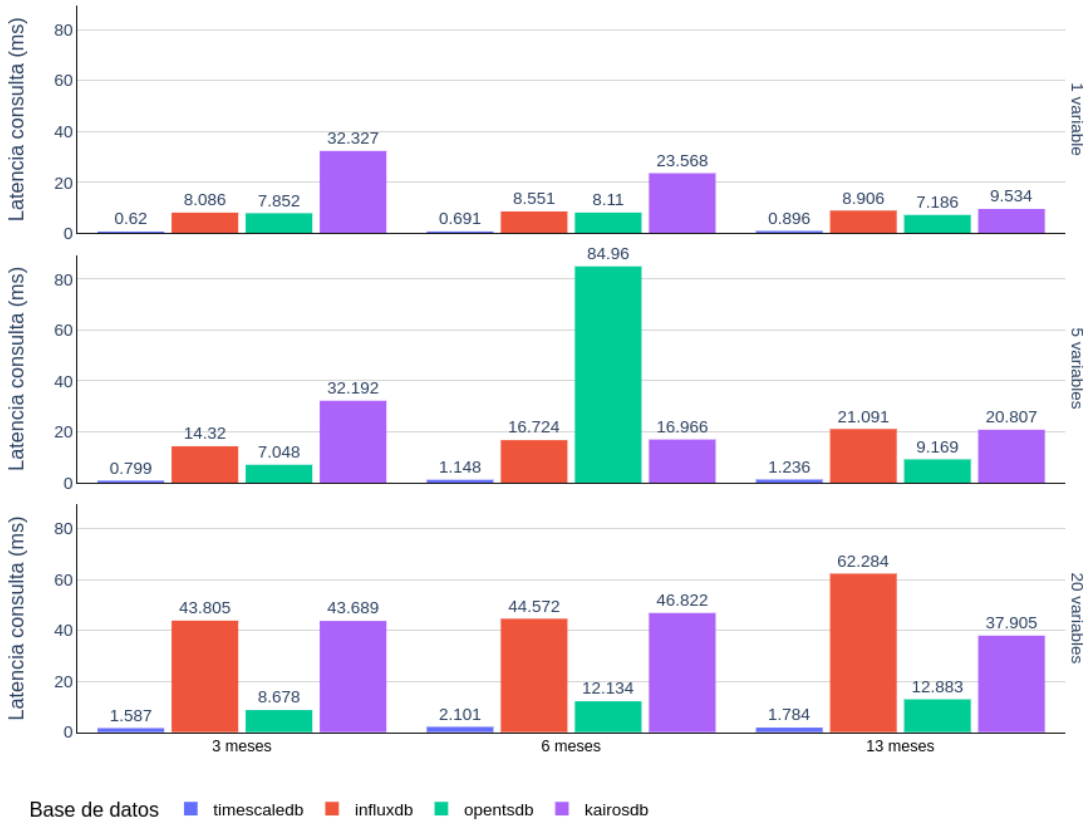


Figura 7.14: Gráfico de barras para la consulta 10_2 (1 día y unas horas)

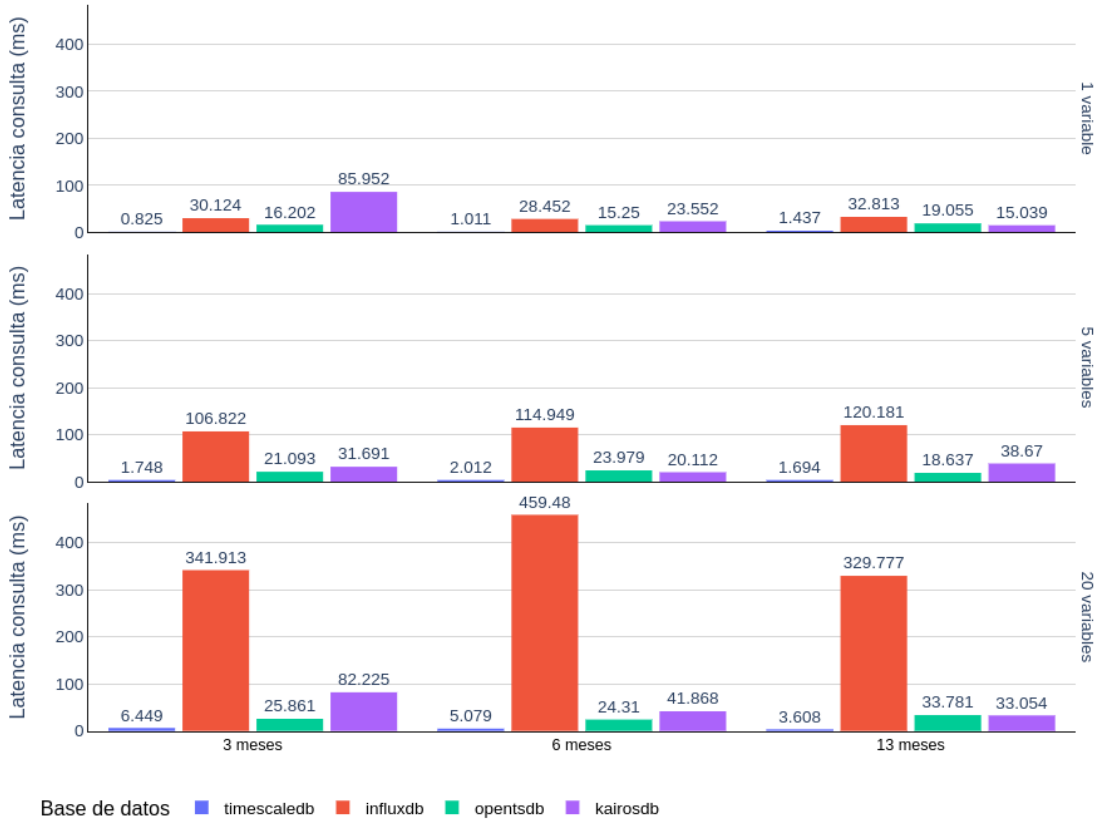


Figura 7.15: Gráfico de barras para la consulta 10_3 (1 mes exacto)

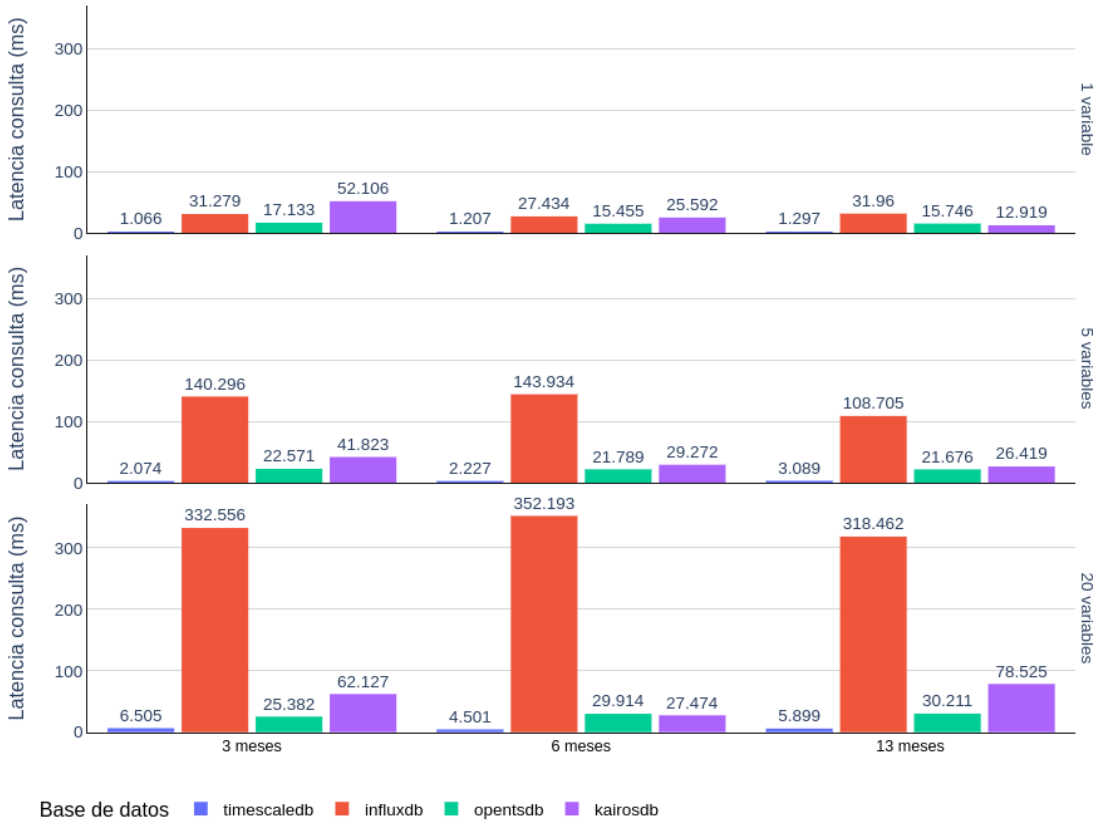


Figura 7.16: Gráfico de barras para la consulta 10_4 (1 mes y unas semanas)

Desde un punto de vista general, observando la Tabla 7.15 podemos concluir que TimescaleDB a medida que se amplia el rango de tiempo consultado el número de variables solicitadas repercute de manera significativa en la latencia de consulta, que KairosDB a pesar de ser peor que otras bases de datos en rangos de tiempo pequeños mantiene unos tiempos de respuesta más estables estables que otras bases de datos aunque muy altos en comparación con la mejor opción, que InfluxDB cuando el rango de tiempo es amplio su comportamiento es inaceptable ya que el aumento de los tiempos de respuesta se multiplican por una cifra significativa, y que, OpenTSDB ofrece unos tiempos en general tolerables que experimentan un aumento esperable en relación con el aumento del rango de tiempo consultado, del tamaño del conjunto y del número de variables consultadas.

En base a los resultados obtenidos, el modelo relacional es el que nos ofrece mejores resultados con tiempos inferiores a seis milésimas de segundo. Las bases de datos que siguen un modelo NoSQL presentan un comportamiento más irregular viéndose más afectadas por los cambios en el número de variables consultadas o en el tamaño del conjunto, además de ofrecer resultados significativamente superiores.

Podríamos decir que la mejor opción es TimescaleDB, seguido de OpenTSDB, KairosDB e InfluxDB. Cabe decir que InfluxDB obtiene el último puesto a parte de porque presenta 29 casos negativos frente a los 15 de KairosDB, porque produce unos tiempos de latencia exageradamente elevados para cuando se consulta un elevado número de variables.

Tamaño del conjunto de datos		3 meses			6 meses			10 meses			13 meses		
Número de variables consultadas		1	5	20	1	5	20	1	5	20	1	5	20
Consulta completa	**TimescaleDB**	6.705	8.196	12.423	19.601	29.351	56.171	64.222	133.551	116.594	119.464	143.933	263.189
Sólo calculan un cuartil	**InfluxDB**	9.765	11.416	31.920	11.725	36.285	46.788	17.173	28.396	70.066	20.004	46.917	140.185
	OpenTSDB	61.645	70.894	87.276	141.806	152.316	234.767	375.337	448.381	569.502	486.821	637.434	833.515
	KairosDB	211.940	114.178	264.126	74.548	131.453	238.723	149.569	121.797	332.078	117.420	182.850	445.063

Tabla 7.14: Tiempos para obtener los intervalos de outliers de las mediciones (en milisegundos)

Tamaño del conjunto de datos		3 meses			6 meses			10 meses			13 meses		
Número de variables consultadas		1	5	20	1	5	20	1	5	20	1	5	20
1 día	**InfluxDB**	10.077	13.382	34.574	7.121	17.329	30.798	9.719	13.692	31.780	7.484	15.673	36.337
	TimescaleDB	0.851	1.419	1.868	0.915	1.139	1.842	1.095	1.480	1.694	1.191	1.493	1.922
	OpenTSDB	6.835	7.676	9.217	7.059	6.887	10.413	6.965	7.665	7.572	13.819	10.077	11.136
	KairosDB	39.701	71.086	24.029	33.177	21.966	25.013	13.442	29.100	45.337	10.967	15.849	18.946
Más de 1 día	**InfluxDB**	8.086	14.320	43.805	8.551	16.724	44.572	8.869	16.839	40.279	8.906	21.091	62.284
	TimescaleDB	0.620	0.799	1.587	0.691	1.148	2.101	0.780	1.377	1.493	0.896	1.236	1.784
	OpenTSDB	7.852	7.048	8.678	8.110	84.960	12.134	14.641	43.671	12.667	7.186	9.169	12.883
	KairosDB	32.327	32.192	43.689	23.568	16.966	46.822	13.626	21.100	34.392	9.534	20.807	37.905
1 mes	**InfluxDB**	30.124	106.822	341.913	28.452	114.949	459.480	36.989	107.197	336.627	32.813	120.181	329.777
	TimescaleDB	0.825	1.748	6.449	1.011	2.012	5.079	1.319	2.290	3.122	1.437	1.694	3.608
	OpenTSDB	16.202	21.093	25.861	15.250	23.979	24.310	24.304	51.070	28.751	19.055	18.637	33.781
	KairosDB	85.952	31.691	82.225	23.552	20.112	41.868	22.115	14.767	54.521	15.039	38.670	33.054
Más de 1 mes	**InfluxDB**	31.279	140.296	332.556	27.434	143.934	352.193	39.088	111.881	537.226	34.960	108.705	318.4629
	TimescaleDB	1.066	2.074	6.505	1.207	2.227	4.501	1.343	3.222	4.289	1.297	3.089	5.899
	OpenTSDB	17.133	22.571	25.382	15.455	21.789	29.914	16.418	26.244	32.487	15.746	21.676	30.211
	KairosDB	52.106	41.823	62.127	25.592	29.272	27.474	24.468	27.224	49.918	12.919	26.419	78.525

Tabla 7.15: Tiempos para obtener las mediciones de diferentes rangos de tiempo (en milisegundos)

Capítulo 8

Herramienta de visualización desarrollada

8.1. Definición de requisitos

Durante la fase de análisis antes de implementar nuestro sistema, determinaremos los servicios que este debe proporcionar (requisitos funcionales) y sus limitaciones (requisitos no funcionales).

8.1.1. Requisitos funcionales

Los requisitos funcionales enuncian los servicios que debe proporcionar el sistema, estos engloban cómo debería reaccionar el sistema ante determinadas entradas o en situaciones específicas y, en ocasiones, también lo que no debería hacer dicho sistema.

RF-1	**Acceder a los datos guardados**
Dependencias	Ninguna
Característica	El sistema deberá ser capaz de acceder a los resultados de las pruebas realizadas
Descripción	El sistema deberá acceder a los datos solicitados por el usuario para generar una visualización
Interfaz del servicio	No
Importancia	Alta
Prioridad	Alta
Precondición	El usuario deberá haber ejecutado el script que realiza las consultas o bien tener los resultados anteriores en una carpeta llamada *'results'*
Postcondición	Se cargarán los datos en la visualización
Comentarios	Si no se ha ejecutado el script o no existen resultados guardados el sistema informará de ello

Tabla 8.1: RF-1: Acceder a los datos almacenados

RF-2	Mostrar resultados de las consultas
Dependencias	RF-1 (8.1)
Característica	El sistema deberá mostrar al usuario el resultado de las consultas realizadas
Descripción	El sistema proporcionará una interfaz que mostrará el resultado de las consultas realizadas
Interfaz del servicio	Sí
Importancia	Alta
Prioridad	Alta
Precondición	El usuario deberá haber ejecutado el script que realiza las consultas o bien tener los resultados anteriores en una carpeta llamada *'results'*
Postcondición	Se mostrará una pestaña con la visualización
Comentarios	Si no se ha ejecutado el script o no existen resultados guardados se mostrará una pestaña vacía

Tabla 8.2: RF-2: Mostrar resultados de las consultas

RF-3	Mostrar resultados de rendimiento
Dependencias	RF-1 (8.1)
Característica	El sistema deberá mostrar al usuario el resultado de las pruebasde rendimiento realizadas
Descripción	El sistema proporcionará una interfaz que mostrará el resultado de las pruebas de rendimiento realizadas
Interfaz del servicio	Sí
Importancia	Alta
Prioridad	Alta
Precondición	El usuario deberá haber ejecutado el script que realiza las pruebas o bien tener los resultados anteriores en una carpeta llamada *'results'*
Postcondición	Se mostrará una pestaña con la visualización
Comentarios	Si no se ha ejecutado el script o no existen resultados guardados se mostrará una pestaña vacía

Tabla 8.3: RF-3: Mostrar resultados de rendimiento

RF-4	Modificar parámetros de la visualización
Dependencias	RF-2 (8.2), RF-3 (8.3)
Característica	El sistema deberá mostrar elementos para modificar los parámetros que aparecen en la visualización
Descripción	El sistema proporcionará unos ajustes de configuración en la interfaz para que el usuario pueda decidir los parámetros que quiere ver representados en la visualización
Interfaz del servicio	Sí
Importancia	Alta
Prioridad	Alta
Precondición	Deberán existir resultados que visualizar
Postcondición	Se mostrará en la pestaña la configuración y la visualización
Comentarios	Si no se ha ejecutado el script o no existen resultados guardados se mostrará una pestaña vacía sin configuración

Tabla 8.4: RF-4: Modificar parámetros de la visualización

RF-5	Mostrar información concreta
Dependencias	RF-2 (8.2), RF-3 (8.3)
Característica	El sistema deberá mostrar al usuario los valores de cada punto de la visualización al pasar por encima
Descripción	El sistema proporcionará una interfaz que mostrará un cuadro flotante con información
Interfaz del servicio	Si
Importancia	Media
Prioridad	Media
Precondición	El usuario deberá haber ejecutado el script que realiza las pruebas o bien tener los resultados anteriores en una carpeta llamada *'results'*
Postcondición	Se mostrarán información específica del punto representado
Comentarios	Si no se ha ejecutado el script o no existen resultados guardados se mostrará una pestaña vacía

Tabla 8.5: RF-5: Mostrar múltiples visualizaciones

RF-6	Ajustar configuración a los recursos disponibles
Dependencias	RF-4 (8.4)
Característica	El sistema deberá mostrar en el ajuste de configuración los parámetros que se puedan modificar en función de los resultados existentes
Descripción	El sistema deberá conocer los parámetros que se pueden modificar en función de los resultados existentes mediante una función
Interfaz del servicio	No
Importancia	Alta
Prioridad	Alta
Precondición	Deberán existir resultados que visualizar
Postcondición	Tras este proceso la configuración mostrará sólo los parámetros de la visualización que sean modificables
Comentarios	

Tabla 8.6: RF-6: Ajustar configuración a los recursos disponibles

RF-7	Mostrar varias visualizaciones simultáneamente
Dependencias	RF-2 (8.2), RF-3 (8.3)
Característica	El sistema deberá mostrar visualizaciones en diferentes pestañas
Descripción	El sistema deberá permitir al usuario ver varias visualizaciones a la vez
Interfaz del servicio	No
Importancia	Media
Prioridad	Media
Precondición	Deberán existir resultados que visualizar
Postcondición	Se podrá contar con más de una visualización en la pantalla
Comentarios	El usuario podrá abrir la dirección web en la que se encuentra la visualización las veces que quiera y modificar cada gráfica como desee

Tabla 8.7: RF-7: Guardar la visualización

RF-8	Guardar la visualización
Dependencias	RF-2 (8.2), RF-3 (8.3)
Característica	El sistema deberá tener la opción de guardar la visualización
Descripción	El sistema deberá permitir al usuario guardar la gráfica con los datos representados
Interfaz del servicio	Si
Importancia	Media
Prioridad	Media
Precondición	Deberán existir resultados que visualizar
Postcondición	Se guardará la visualización en el sistema de almacenamiento local
Comentarios	

Tabla 8.8: RF-8: Guardar la visualización

8.1.2. Requisitos no funcionales

Los requisitos no funcionales se encargan de presentar las características de funcionamiento del sistema, estos suelen aplicarse al sistema como un todo describiendo las limitaciones sobre los servicios o funciones que este nos ofrece.

RNF-1	Tiempos de respuesta
Descripción	El sistema deberá mostrar la visualización deseada por el usuario en menos de 1 minuto
Importancia	Alta
Prioridad	Alta

Tabla 8.9: RNF-1: Tiempos de respuesta

RNF-2	Información de fallos
Descripción	El sistema deberá mostrar un mensaje de información si no existen resultados para visualizar
Importancia	Media
Prioridad	Media

Tabla 8.10: RNF-2: Información de fallos

RNF-3	Facilidad de uso
Descripción	El sistema deberá deberá tener una interfaz fácil e intuitiva para el usuario
Importancia	Alta
Prioridad	Alta

Tabla 8.11: RNF-3: Facilidad de uso

8.2. Modelado del sistema

En esta sección nos ocuparemos del desarrollo de modelos abstractos del sistema utilizando el Lenguaje de Modelado Unificado (UML). Recogeremos gráficamente las características más

destacables del sistema mediante diferentes diagramas que nos ayudarán a tomar decisiones en la etapa de diseño.

8.2.1. Modelo de dominio

Como podemos ver en la Figura 8.1 tenemos una representación de clases conceptuales significativas para abordar el dominio de nuestro problema. Partimos de la base de que nuestros datos están formados por las mediciones de los analizadores de red del edificio que queremos almacenar para realizar las pruebas de rendimiento. Como estas pruebas de rendimiento consisten en realizar diferentes consultas a las bases de datos seleccionadas tenemos una clase *Query* que representa las consultas que llevaremos a cabo y que está formada por:

- *id_query*: identificador de consulta para saber si es de carga, almacenamiento o de recuperación de datos.
- *db_exp*: base de datos en la que se ejecuta la consulta.
- *params*: parámetros de la consulta que incluyen rangos de tiempo, funciones o identificadores de analizadores de red para los que recuperamos datos entre otros.
- *size_exp*: tamaño del conjunto de datos para saber la cantidad de datos sobre la que realizamos la consulta.
- *type_q*: tipo de la consulta para identificar si es de punto exacto, análisis básico, agrupación o rangos de tiempo.
- *size_q*: tamaño de la consulta que hace referencia a la amplitud de la consulta para las consultas de rangos o agrupación para saber si la consulta abarca todos los datos, una semana u otras opciones.

Tras la realización de las pruebas de rendimiento que consiste en lanzar distintas consultas a todas las bases de datos sobre distintos tamaños del conjunto de datos llegamos a que una consulta está relacionada con uno o más resultados, es decir, tenemos una clase *Result* con los siguientes atributos:

- *id_exp*: mantiene una numeración de los experimentos realizados.
- *db_exp*: base de datos en la que se ejecuta la consulta.
- *size_exp*: tamaño del conjunto de datos para saber la cantidad de datos sobre la que realizamos la consulta.
- *result_exp*: hará referencia a los dos resultados que produce una consulta, por un lado, tendremos los resultados de rendimiento y ,por otro, los resultados de la propia consulta recuperados de la base de datos.
- *n_var*: número de analizadores de red sobre los que queremos recuperar datos en la consulta.

En la figura podemos ver una clase *DataView* para definir las vistas que el usuario podrá generar en función de los parámetros que seleccione en la configuración disponible de la visualización. Entre la clase *Result* y *DataView* tenemos una relación uno a cero o más, ya que podemos generar diferentes vistas a partir de unos resultados como no generar ninguna porque no existan datos recuperados por la consulta. Los atributos de esta clase representan las bases de datos, los tamaños del conjunto de datos y la cantidad de identificadores de red sobre los que hemos ejecutados las consultas, así como la consulta que queremos visualizar y a partir de todo lo anterior un resultado que consiste en la visualización generada.

La relación de herencia que aparece en la figura representa los dos tipos de visualizaciones que podemos generar en función de los resultados que queremos visualizar; resultados de rendimiento con un gráfico de barras, que bien puede ser de tiempos de respuesta, o bien de tamaño de bases de datos si es la consulta de almacenamiento; o resultados de consultas con un gráfico de líneas que nos presenta mediciones de energía o el resultado de una función analítica aplicada a estas.

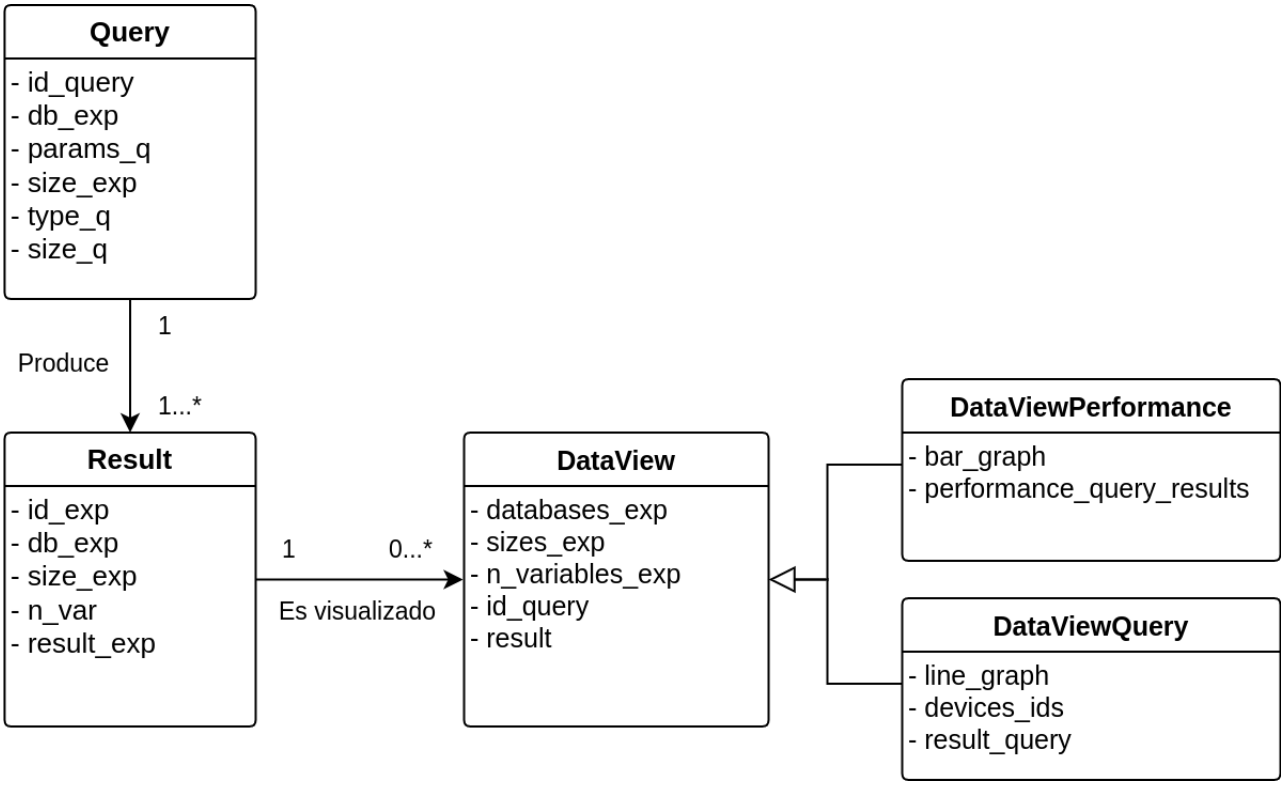


Figura 8.1: Modelo de dominio (Figura de elaboración propia)

8.2.2. Casos de uso

Los diagramas de caso de uso nos permiten representar de forma gráfica las funciones más importantes de un sistema y ver cómo un actor interactúa con el sistema de forma sencilla. En la Figura 8.2 vemos las cinco funcionalidades que el usuario espera obtener del sistema.

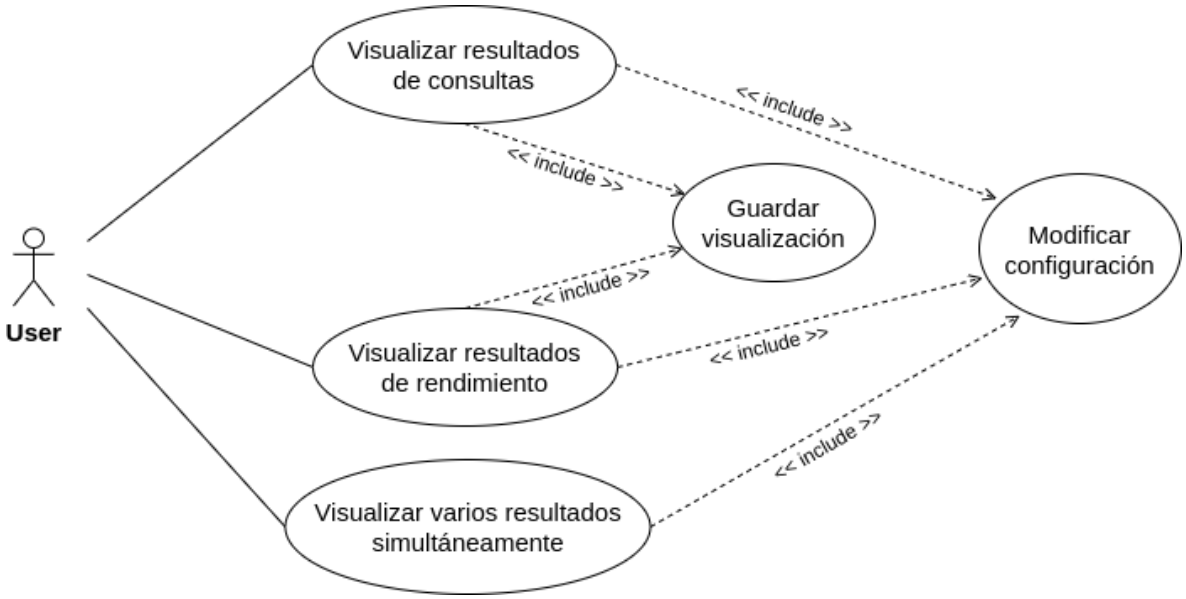


Figura 8.2: Casos de uso (Figura de elaboración propia)

8.2.3. Diagramas de secuencia

Por último, antes de pasar al diseño de nuestra herramienta, podemos ver en la Figura 8.3 un diagrama de secuencia que nos permite modelar las interacciones principales entre los actores y los objetos de nuestro sistema.

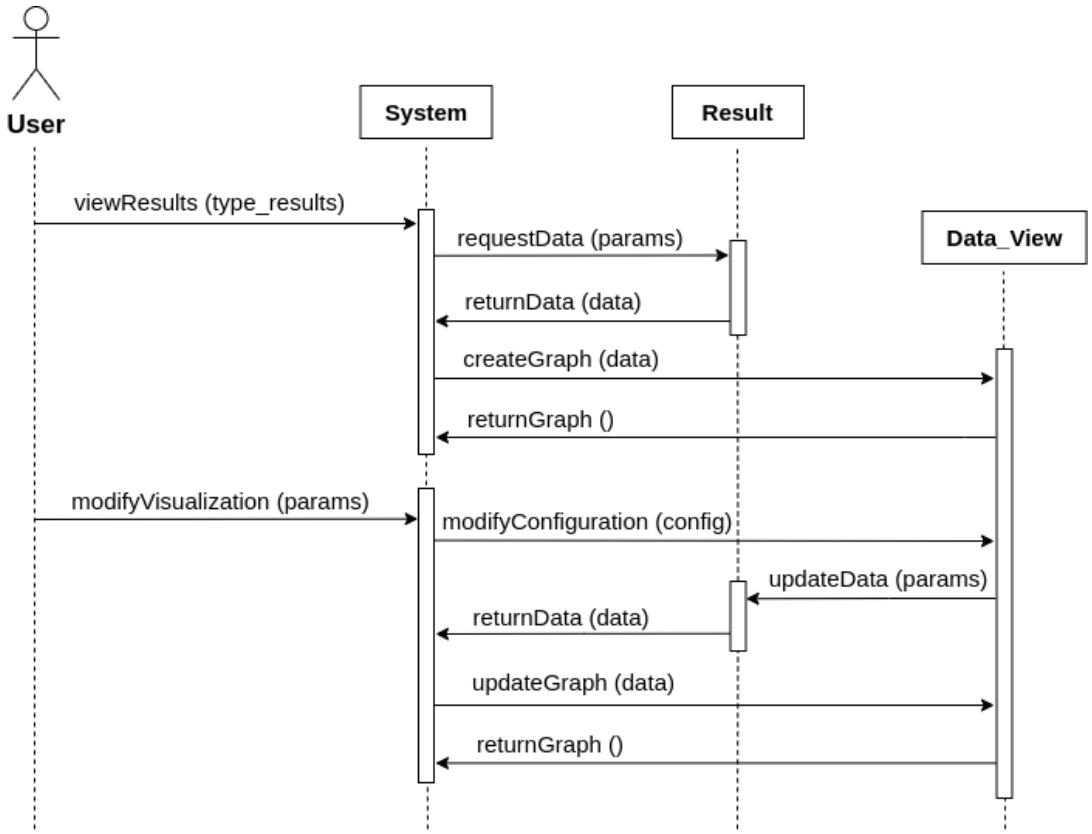


Figura 8.3: Diagrama de secuencia (Figura de elaboración propia)

8.3. Diseño

En esta sección nos centraremos en los aspectos relacionados con el diseño de software, decidiremos el patrón de diseño de nuestra aplicación, los prototipos de la interfaz de usuario y por último, decidiremos las herramientas que utilizaremos para implementar nuestro servicio.

8.3.1. Diseño arquitectónico

El diseño arquitectónico es una parte muy importante en el proceso de desarrollo de software ya que se encarga de comprender la organización necesaria para un sistema y determinar su estructura. Durante este proceso se deben tomar decisiones que afectarán al sistema, una de ellas será la elección de la arquitectura en la que se basará nuestro sistema. La arquitectura de software elegida se basará en el uso de uno o varios patrones arquitectónicos consistentes en una descripción abstracta simplificada de buena práctica que ha tenido éxito en sistemas previos.

Nuestra idea es desarrollar una pequeña aplicación web que muestre la visualización de los resultados de las consultas y la posibilidad de modificar determinados parámetros para adaptar la gráfica y obtener diferentes comparaciones, por ello creemos conveniente utilizar un patrón MVC (Modelo - Vista - Controlador).

El patrón MVC consiste en crear una separación entre la presentación y la interacción de los datos del sistema. Como vemos en la Figura 8.4, este patrón consta de tres componentes lógicos:

- Modelo: opera sobre los datos del sistema y las acciones sobre estos.
- Vista: gestiona la presentación de los datos al usuario del sistema.
- Controlador: dirige las interacciones del usuario con el sistema, informando de esto al Modelo y a la Vista.

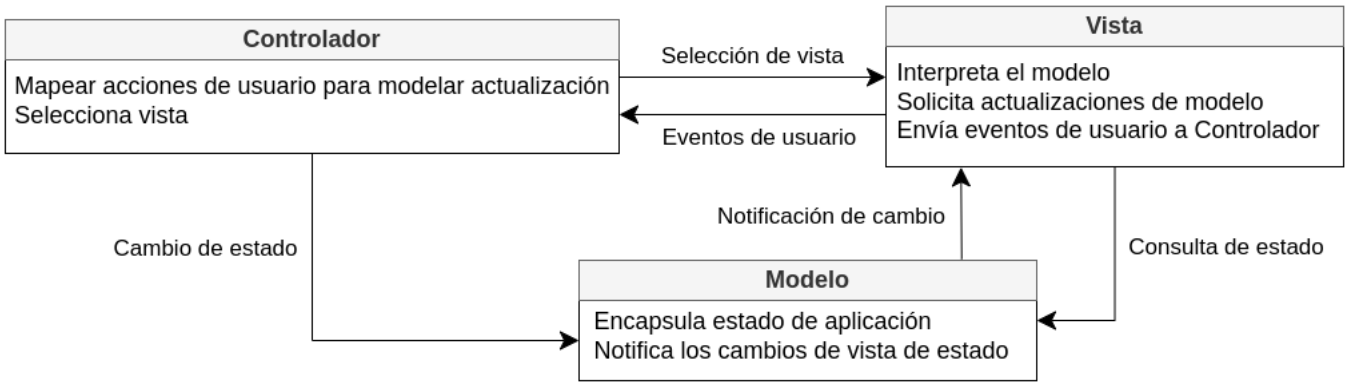


Figura 8.4: Organización del MVC (Fuente: parte 1, capítulo 6. Diseño arquitectónico[45])

Es un modelo muy utilizado a día de hoy y nos ofrece ventajas como la separación de responsabilidades utilizando una estructura organizada, la facilidad de comprensión del código y la reutilización de este.

8.3.2. Diseño de interfaz de usuario

Con el objetivo de plasmar todos los requisitos funcionales definidos en la Sección 8.1 simularemos la interfaz de usuario que queremos conseguir realizando diferentes prototipos. Emplearemos un patrón de diseño que sigue el principio KISS (Keep it Stupidly Simple) y que busca obtener una interfaz sencilla en la que el usuario no se encuentre con barreras de identificación, pueda retroceder si lo desea y pueda comprender el propósito de la aplicación y su funcionamiento de un vistazo.

En la Figura 8.5 podemos observar la primera vista que se encontrará el usuario al ejecutar la aplicación y que le dará a elegir entre visualizar resultados de las consultas o de las pruebas de rendimiento.

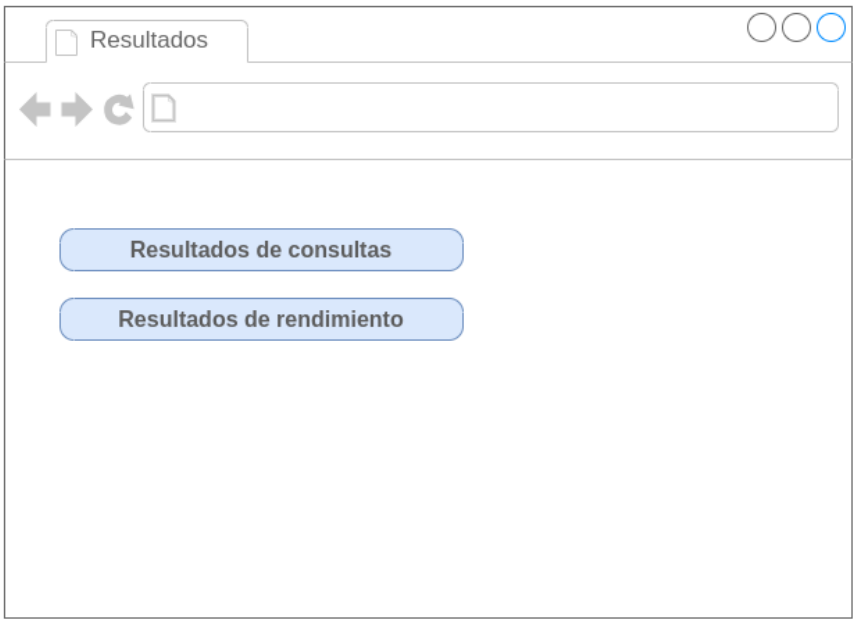


Figura 8.5: Boceto pantalla principal de la interfaz

En la Figura 8.6 el usuario podrá tener acceso a una gráfica de puntos o de líneas en función de la consulta seleccionada, ya que en algunos casos se mostrará una variable cuantitativa discreta (por ejemplo, en la consulta para la medición de una determinada marca de tiempo o la consulta del conteo de mediciones) y en otros casos, se representará una variable cuantitativa continua (por ejemplo, en consultas de rangos de tiempo donde mostraremos las mediciones ligadas a la variable tiempo de ese rano). La configuración de los parámetros en este caso nos permitirá seleccionar una consulta, una base de datos, un tamaño y un número de variables determinado para cada visualización y después filtrar esta por cada identificador de analizador de red que esté disponible para los datos seleccionados.

En la Figura 8.7 tenemos la vista que se le mostrará al usuario si quiere ver los resultados de las pruebas de rendimiento de este proyecto representados mediante gráficos de barras agrupadas. La configuración de los parámetros permitirá seleccionar la consulta de la que se quieren mostrar los resultados y hará posible la actualización del gráfico en función de la selección múltiple de las bases de datos, los tamaños y el número de variables consultadas.

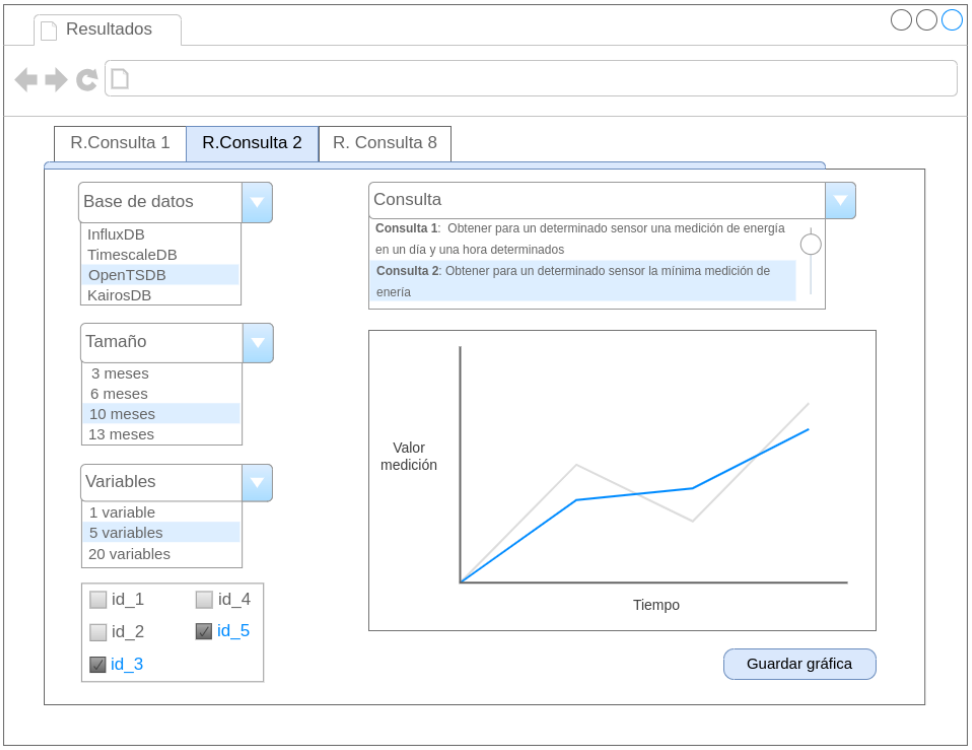


Figura 8.6: Boceto pantalla de resultados de consultas de la interfaz

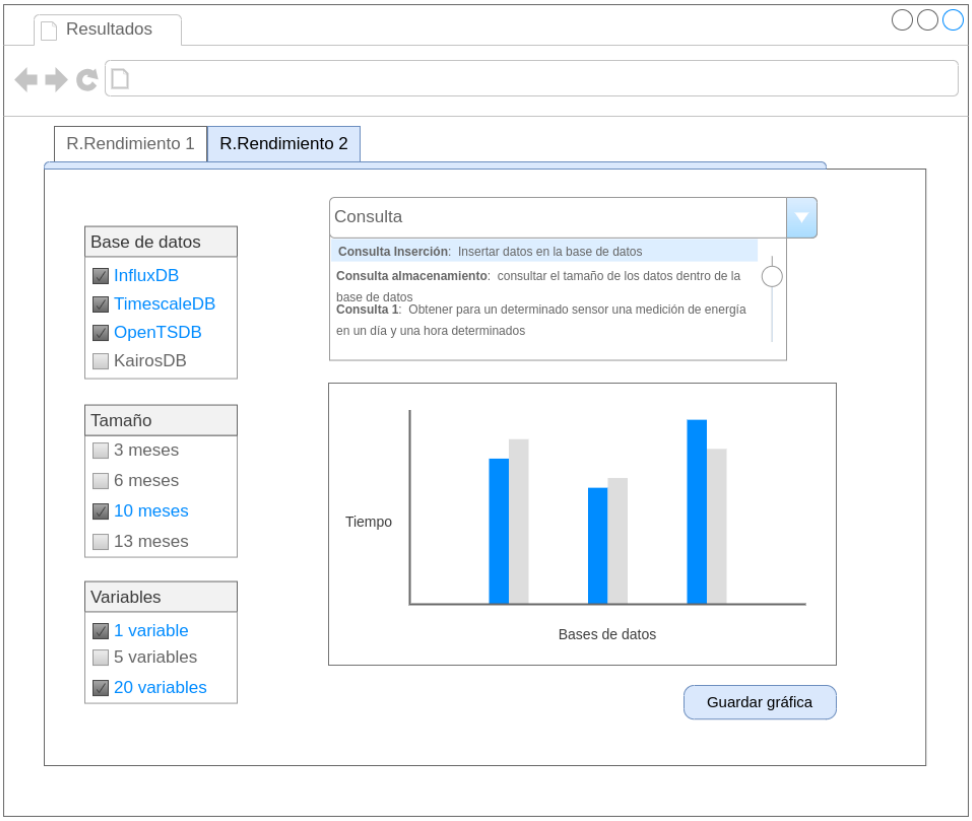


Figura 8.7: Boceto pantalla de resultados de rendimiento de la interfaz

8.3.3. Implementación de la herramienta

Tras analizar un estudio comparativo entre varios frameworks web de Python[13] a la hora de desarrollar nuestra aplicación nos decidimos a probar Flask[39], un microframework de código abierto escrito en Python, muy útil para proyectos pequeños y con una curva de aprendizaje baja. El funcionamiento de Flask es muy sencillo ya que utiliza una biblioteca de aplicaciones web WSGI (Web Server Gateway Interface) denominada Werkzeug y un motor de plantillas Jinja2 (librería que nos permite renderizar páginas web)[37]. Cabe resaltar que a pesar de tratarse de un marco de trabajo minimalista, una de las ventajas de Flask es la de ser capaz de aumentar sus funcionalidades de una forma modular mediante el uso de extensiones y bibliotecas de terceros.

El funcionamiento de este framework es perfectamente compatible con el patrón MVC ya que cubre las necesidades del controlador y la vista, y es posible abarcar el modelo mediante una extensión ya que por defecto no se encarga de esta parte. Además, emplea una arquitectura cliente-servidor en la cual desde el navegador se hace una petición a una dirección URL que el WSGI interpreta, y solicita la plantilla HTML (HyperText Markup Language) y los datos asociados a dicha dirección para ser enviados de vuelta al navegador[26].

Nuestra aplicación estará formada por un archivo *'controller.py'* que se encargará de definir los decoradores de Flask que se encargan de asociar una ruta de una URL de la aplicación con una función que debe ejecutarse. Así tendremos tres rutas, una para el inicio de la aplicación que nos mostrará las opciones disponibles, una para la visualización de los resultados de las consultas y otra para la visualización de los resultados de rendimiento. En esas funciones el controlador se encargará de tomar los datos introducidos por el usuario en un formulario y solicitar información al modelo.

A su vez, tendremos el archivo *'models.py'* que será el responsable de recuperar los resultados generados en los experimentos de rendimiento y filtrarlos para generar las gráficas que le solicite el controlador en base a las decisiones del usuario. También tendremos un conjunto de plantillas HTML, un archivo de estilo CSS y un archivo con una función en JavaScript para la funcionalidad de guardar la visualización.

A continuación, podemos ver las pantallas principales de la aplicación que nos muestran las visualizaciones. En la Figura 8.8 podemos ver un gráfico de barras en el que para la base de datos, el tamaño, las variables y la consulta seleccionada disponemos de datos para los 5 identificadores de analizador de red de los que solo decidimos mostrar 2. Podríamos tener el caso de que para los tres meses de datos seleccionados no tuviésemos mediciones de los 5 identificadores y de esta manera la lista se reduciría.

En la Figura 8.9 nos encontramos con un gráfico de barras en el que hemos seleccionado todas las bases de datos y todos los tamaños posibles para la consulta seleccionada. En este caso, a la hora de mostrar resultados de rendimiento nos encontramos que para la consulta de inserción no tenemos la posibilidad de elegir el número de variables consultadas ya que no está disponible.

Sin embargo, en el caso de la Figura 8.10 podemos ver que en el caso de seleccionar una consulta que sí dispone del parámetro configurable del número de variables consultadas, en caso de que no se seleccionen será la pantalla que nos muestre la aplicación hasta elegir en la

lista que aparece debajo de la consulta sobre qué número o números de variables consultadas queremos realizar el gráfico comparativo.

En la Figura 8.11 podemos ver la pantalla que nos mostrará la aplicación si no existe la carpeta resultados o si existe pero no se encuentra en la ubicación correcta. Hay que mencionar, que recurrimos a esta opción en lugar de dar a elegir al usuario la dirección de los resultados del lado del servidor, ya que conllevaría realizar un desarrollo más complejo teniendo en cuenta medidas de seguridad y configuración de permisos que nos llevaría más tiempo y nos alejaría del objetivo principal del proyecto.

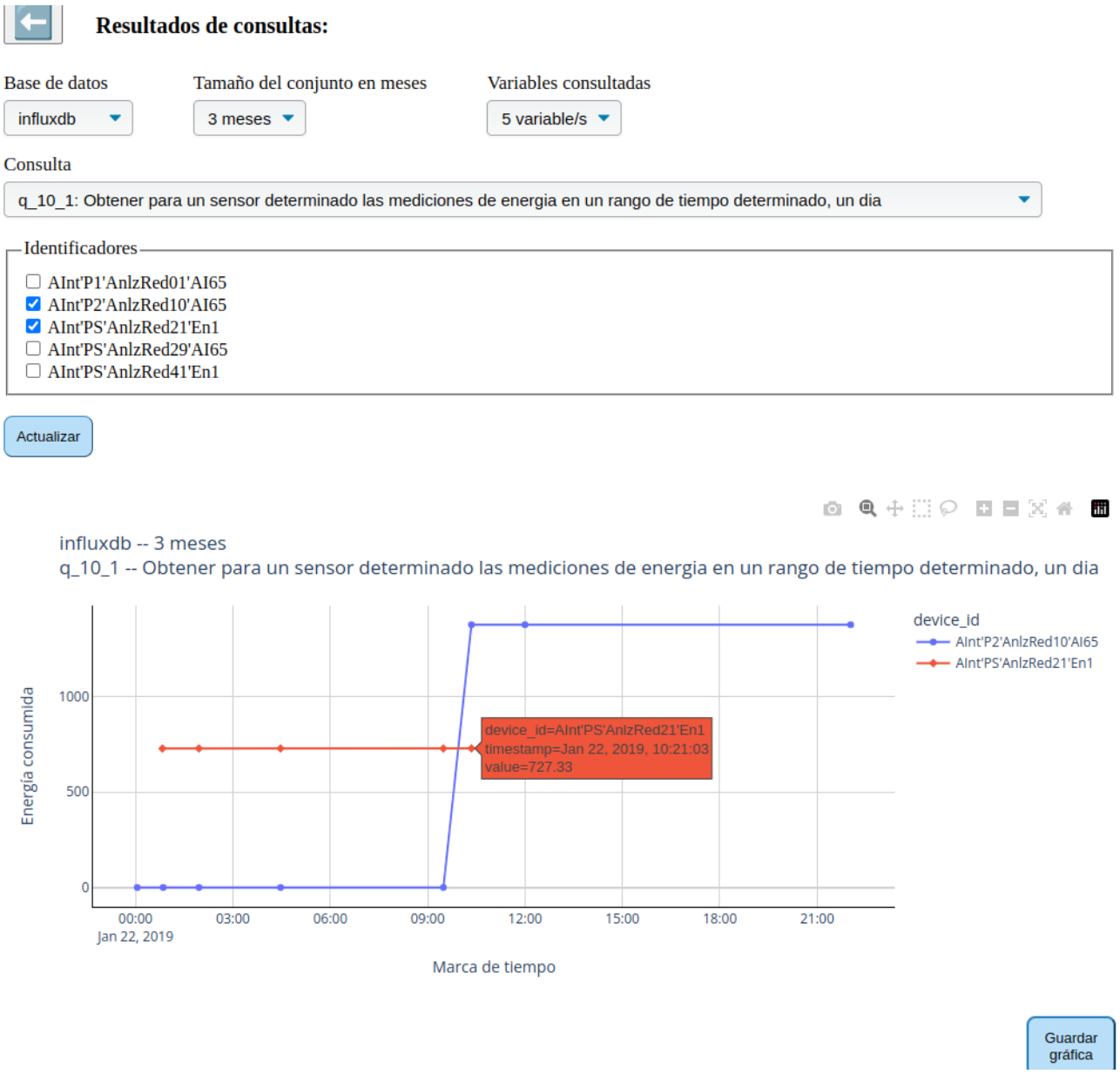


Figura 8.8: Pantalla de resultados de consultas de la aplicación

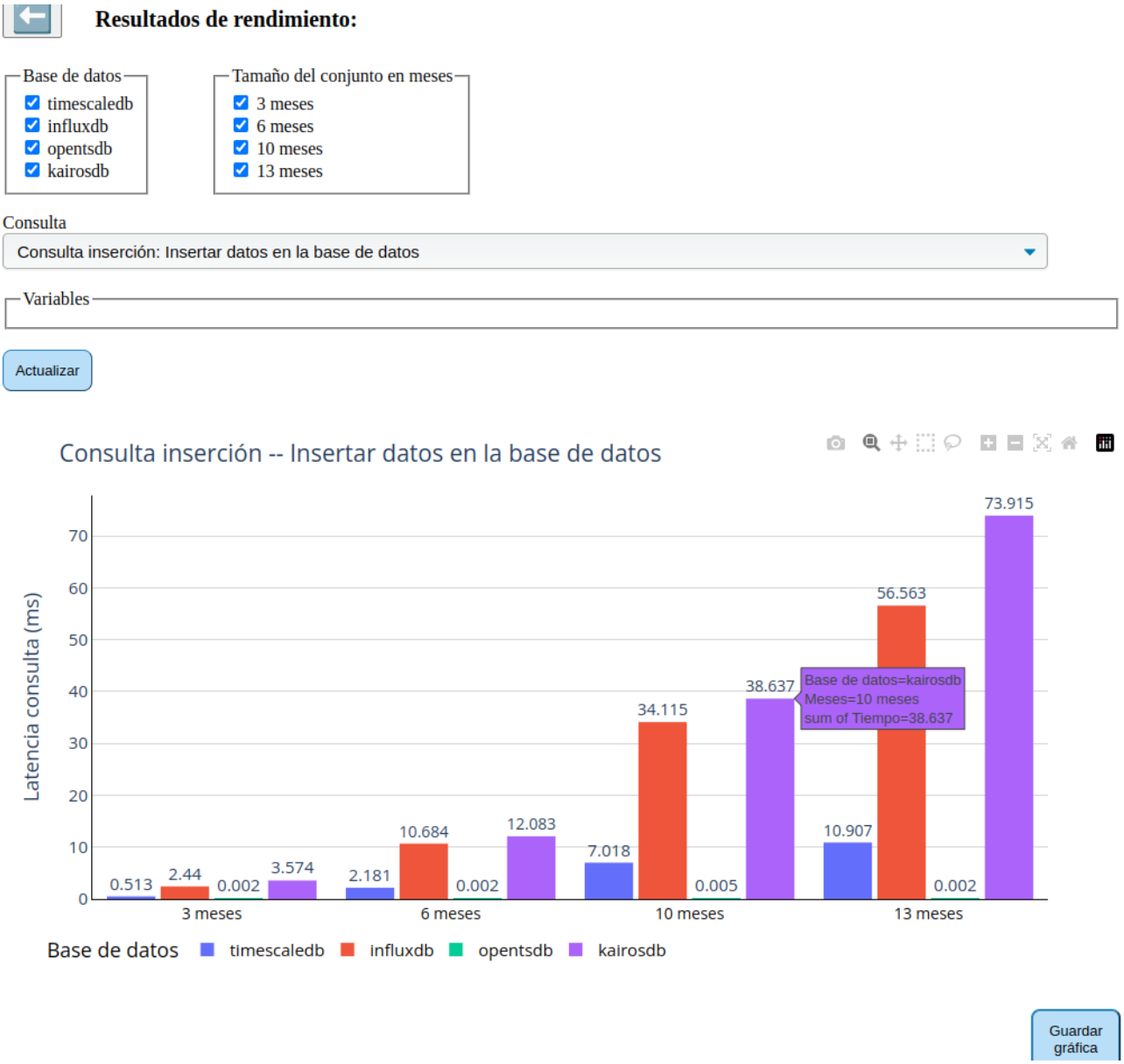


Figura 8.9: Pantalla de resultados de rendimiento de la aplicación

Figura 8.10: Pantalla de resultados de rendimiento (sin selección del número de variables) de la aplicación

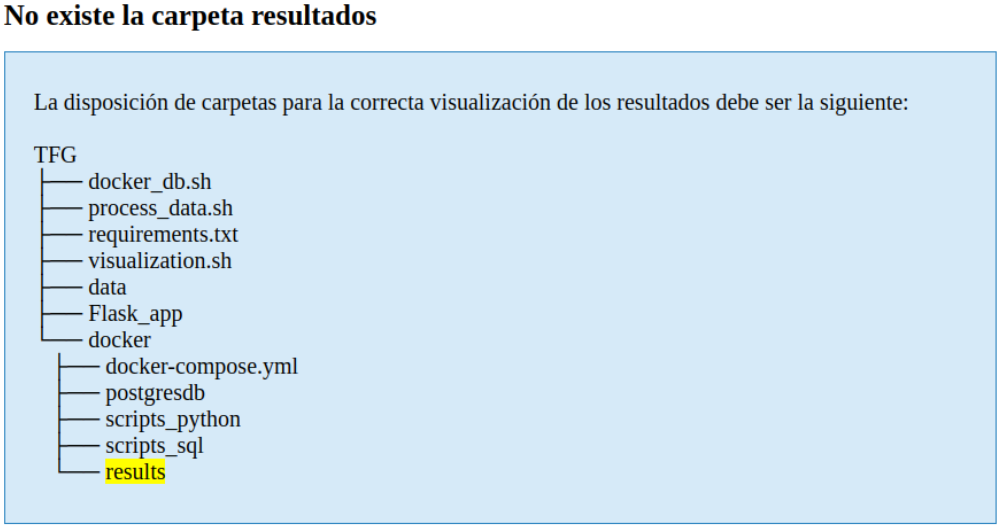


Figura 8.11: Pantalla de error de la aplicación

Es importante añadir que utilizamos el servidor WSGI para desarrollo y en caso de que queramos llevarlo a producción para que la aplicación web fuese pública habría que modificar la configuración de Flask para utilizar un servidor Gunicorn, Nginx u otro[38].

8.4. Manual del instalador

El objetivo principal de este proyecto es realizar un estudio sobre el rendimiento de diferentes bases de datos de series temporales.

¡Estos programas deben ser ejecutados en Linux!

Para ejecutar este proyecto se necesitan los archivos de este repositorio y el directorio con los datos originales que se tendrán que solicitar de manera privada si se dispone de los permisos necesarios.

Instalación

Para ejecutar este proyecto debemos instalar las siguientes herramientas de software:
Para ejecutar este proyecto deberemos comprobar que la versión que tenemos de Python es la 3 y tener el instalador de paquetes 'pip'. Para ello podrá ejecutar las siguientes líneas de código en su terminal de Linux:

```
sudo apt install python3
sudo apt install pip
```

También tendremos que instalar Docker. Para utilizar esta herramienta sin privilegios de superusuario podemos ejecutar lo siguiente:

```
sudo groupadd docker
sudo usermod -aG docker $USER
sudo systemctl restart docker
sudo chmod 666 /var/run/docker.sock
```

Pre-requisitos

En el archivo *'requirements.txt'* podemos encontrar las bibliotecas de Python que hemos utilizado durante el desarrollo del trabajo. Para instalarlas deberemos ejecutar el siguiente comando y si lo desea puede crear antes un entorno virtual para que no afecten estás versiones a otras que tenga instaladas en su máquina.

```
pip install -r requirements.txt
```

Ejecución

Teniendo en cuenta que hemos seguido los pasos anteriores y tenemos todo el software necesario, el primer paso será descargar este repositorio y obtener la carpeta *'raw_data'* con los datos originales y que tendremos que ubicar en *'TFG_PatriciaAguado/data/'*.

A continuación, tendremos que dar permisos de ejecución al usuario para los scripts de Bash:

```
sudo chmod u+x docker_db.sh
sudo chmod u+x process_data.sh
sudo chmod u+x visualization.sh
```

En primer lugar, abrimos una terminal en *'TFG_PatriciaAguado'* y ejecutamos el script *'process_data.sh'*:

```
./process_data.sh
```

Este se encarga de ejecutar en orden los programas de Python que se encuentran en el directorio *'data'*. Estos consisten en:

- *'read_data.py'*: leer todos los archivos que se encuentran en el directorio *'data/raw_data'* y juntarlos para generar el archivo *'data/silver_data.xlsx'*.
- *'clean_data.py'*: realizar sobre *'silver_data.xlsx'* un proceso de limpieza de datos y generar en el directorio *'data/gold_data'* un archivo con todos los datos de calidad y uno por cada tamaño especificado (3, 6, 10 y 13 meses).
- *'json_schemas.py'*: crear un directorio *'data/json_schemas'*, en el que se crea un directorio por cada base de datos y en cada uno de ellos un archivo de formato JSON para cada tamaño.

En segundo lugar, una vez que ya tenemos todos los archivos preparados con los datos que queremos insertar para las pruebas de rendimiento en el directorio *'data/json_schemas'* ejecutamos el script *'docker_db.sh'*:

```
./docker_db.sh
```

Este primero monta un contenedor de Docker para nuestra base de datos personal, a partir del script SQL *'docker/scripts_sql/postgresdb.sql'*. Como esta base de datos almacenará los resultados, se genera un directorio de persistencia de datos denominado *'docker/postgresdb'*.

Después, el script continúa montando un contenedor de Docker, realizando pruebas de rendimiento (es decir, ejecutando los programas Python que se encuentran en *'docker/scripts_python'*), guardando los datos recuperados en cada consulta, insertando los resultados de rendimiento en nuestra base de datos y desmontando el contenedor para cada servicio definido en el archivo *'docker/docker-compose.yml'* de forma recursiva.

Al finalizar, se ejecuta el programa *'results_performance.py'* que recupera los resultados de PostgreSQL en un documento XLSX que contiene las tablas con los resultados de las pruebas de rendimiento (*'results_performance.xlsx'*) y que almacena en el directorio *'results'* junto con los resultados de cada consulta en cada experimento.

En último lugar, ejecutamos el script *'visualization.sh'* que nos abrirá una página en nuestro navegador por defecto con el puerto 5000, que es en el que se encuentra el servidor de nuestra aplicación web de visualización de datos:

```
./visualization.sh
```

Utilización de la aplicación

Al abrirse la aplicación web nos encontramos con dos opciones:

- **Resultados de consultas**: obtendremos gráficas de puntos o de líneas en función de la consulta sobre la que queramos observar los resultados. Podremos filtrar los datos a mostrar seleccionando la base de datos, el tamaño de meses y el número de variables consultadas que queremos. Una vez pulsemos el botón de actualizar podremos ver el gráfico deseado, pero si queremos por ejemplo seleccionar unos identificadores de sensor diferentes podremos elegirlos en las casillas de verificación pertinentes y volver a pulsar el botón de actualizar.
- **Resultados de rendimiento**: obtendremos gráficas de barras agrupadas por tamaños para los resultados de las pruebas de rendimiento realizadas. Nos encontraremos casillas de verificación para seleccionar las bases de datos y los tamaños del conjunto en meses que queremos comparar. Una vez seleccionado esto y la consulta para la que queremos graficar los resultados, una vez que pulsemos el botón actualizar en caso de que sea necesario elegir el número de variables consultadas a comparar (porque realizamos consultas preguntando por 1, 5 y 20 variables) aparecerá una visualización vacía que nos indicará que debemos seleccionar este último parámetro que antes igual no estaba visible. Sólo tendremos que pulsar el botón de actualizar y volverá a estar disponible.

En ambos nos encontraremos con un botón para ir atrás y con un botón situado debajo del gráfico en la parte derecha para poder descargar la visualización en formato PNG. Además, los gráficos son interactivos, podemos ampliarlos, modificar las escalas de los ejes o ver información detalla al pasar por encima del dato representado. Hay que mencionar que, en caso de que no exista la carpeta *'docker/results'* nos aparecerá una pantalla de error con indicaciones.

En caso de querer ver varios gráficos en diferentes pestañas, podemos abrir manualmente la misma dirección (http://127.0.0.1:5000/) en otra pestaña o en otro navegador, siempre y cuando no hayamos detenido el servidor en la terminal.

Si queremos cerrar la aplicación basta con cerrar el navegador y presionar en la terminal de Linux *'CTRL+C'*.

Construido con

Las herramientas principales que hemos utilizado para llevar acabo este proyecto son las siguientes:

- Python - Lenguaje de programación
- Docker - Plataforma basada en la virtualización de entornos aislados para facilitar el despliegue de aplicaciones
- Sublime Text - Usado para generar todos los scripts del trabajo
- Flask - Usado para la aplicación web

- Plotly - Usado para generar gráficos interactivos

Imágenes utilizadas en Docker

Las imágenes de Docker utilizadas para definir los servicios de bases de datos en el archivo *'docker-compose.yml'* son las siguientes:

- PostgreSQL - Imagen oficial de Docker para PostgreSQL
- InfluxDB - Imagen oficial de Docker para InfluxDB
- TimescaleDB - Imagen de Docker para TimescaleDB
- OpenTSDB - Imagen de Docker para OpenTSDB
- KairosDB - Imagen de Docker para KairosDB
- Cassandra - Imagen oficial de Docker para Cassandra

8.5. Herramientas utilizadas

En esta sección se explicará brevemente el software utilizado en la realización de este proyecto.

- **Lenguajes**
  - Python: lenguaje de programación eficiente, con una gran variedad de bibliotecas y con un amplio uso en el ámbito de la ciencia de datos y el desarrollo de aplicaciones web. Se utilizará en los principales programas que procesan y limpian los datos originales, así como en los scripts que se comunican con las bases de datos seleccionadas y el backend de nuestra aplicación web.
  - Bash: intérprete de comandos y lenguaje de programación de shell que utilizamos para ejecutar los programas relacionados con el procesamiento de datos, los programas que ejecutan las pruebas de rendimiento montando los contenedores de Docker y para lanzar el servidor de nuestra aplicación.
  - HTML, CSS y JavaScript: lenguajes muy utilizados en desarrollo web y utilizados conjuntamente con Flask para nuestra aplicación de visualización de datos.
  - SQL: lenguaje de consulta estructurado utilizado para comunicarnos con las bases de datos PostgreSQL y TimescaleDB.
  - Flux: lenguaje de consulta y manipulación de datos en la base de datos InfluxDB.
- **Bibliotecas de Python**
  - Time: biblioteca que nos permite trabajar con mediciones de tiempo y que utilizamos para medir la latencia de consultas en las pruebas de rendimiento, de esta manera el método utilizado es el mismo en todas las bases de datos para su posterior análisis y comparación.

  - Pandas y Numpy: bibliotecas muy utilizadas en el procesamiento y análisis de grandes conjuntos de datos, la primera nos proporciona principalmente estructuras de datos flexibles y la segunda nos facilita realizar operaciones numéricas y un análisis estadístico de los datos.
  - Psycopg2: biblioteca con la que podemos conectarnos a PostgreSQL desde Python.
  - Influxdb-client: biblioteca para comunicarnos con la base de datos InfluxDB a partir de la versión 2 y utilizando Flux como lenguaje de consulta.
  - Docker-py: SDK de Docker que nos permite comunicar un cliente de Python con los contenedores de Docker y que utilizamos en las pruebas de rendimiento para conocer cuánto ocupan los datos una vez insertados en cada base de datos.
  - Requests: biblioteca con la que podemos realizar peticiones HTTP de manera sencilla y que nos sirve para enviar las consultas a las bases de datos KairosDB y OpenTSDB.
  - Flask y Plotly: ambos utilizados para el desarrollo de nuestra aplicación web, donde el primero nos permite crear instancias de una aplicación Flask, y el segundo nos permite crear gráficos interactivos.

- **Jupyter Notebook**: aplicación web que permite crear documentos, denominados Notebooks, con código, visualizaciones y explicaciones en formato texto. Esta herramienta nos permite ejecutar código Python en celdas y ver el resultado de manera inmediata debajo de ésta. Este entorno de programación ha sido muy utilizado durante la realización del grado y debido a su comodidad, rapidez y facilidad de uso lo utilizamos en este proyecto para la codificación de los programas antes de convertirlos en scripts de Python.

- **Sublime Text**: editor de texto que se ha utilizado para generar todos los scripts del trabajo. Esta herramienta no es de código abierto pero se puede descargar y su periodo de prueba no tiene fecha límite.

- **Overleaf**: herramienta online que permite redactar, editar y publicar documentos de forma colaborativa utilizando el lenguaje LaTeX, y que se ha utilizado para crear la memoria de este trabajo.

- **Docker**: plataforma que nos permite crear los contenedores necesarios para probar cada una de las bases de datos seleccionadas para este proyecto de forma rápida.

- **GitHub**: servicio de control de versiones basado en la nube utilizado para alojar el código de este trabajo.

- **Draw.io**: aplicación web gratuita utilizada para realizar algunos diagramas explicativos de este proyecto.

8.5.1. Organización del código

En la Figura 8.12 podemos observar la estructura del proyecto en su totalidad tras ejecutar las pruebas de rendimiento. El proceso de ejecución resumido es el siguiente:

En primer lugar, ejecutamos el script *'process_data.sh'* que se encarga de ejecutar en orden los programas de Python que se encuentran en el directorio *'data'*. Tras esta ejecución tendremos un directorio *'json_schemas'* con archivos en formato JSON divididos en función del

número de meses sobre los que contienen datos y la base de datos a la que van dirigidos.

En segundo lugar, una vez que ya tenemos todos los archivos preparados con los datos que queremos insertar para las pruebas de rendimiento, ejecutamos el script *'docker_db.sh'* que monta un contenedor de Docker para nuestra base de datos personal que va a almacenar los resultados. Una vez hecho esto, el script continúa montando un contenedor de Docker, realizando pruebas de rendimiento, guardando los datos recuperados en cada consulta y desmontando el contenedor para cada servicio definido en el archivo *'docker-compose.yml'* de forma repetitiva. Al finalizar, tendremos en el directorio *'results'* los resultados de PostgreSQL en un documento XLSX que contiene las tablas con los resultados de las pruebas de rendimiento y archivos JSON con los resultados de cada consulta en cada experimento.

En último lugar, podremos ejecutar el script *'visualization.sh'* que nos abrirá una página en nuestro navegador web con el puerto 5000, en la que tendremos acceso a la interfaz de la aplicación y en la que podremos visualizar los resultados del proyecto.

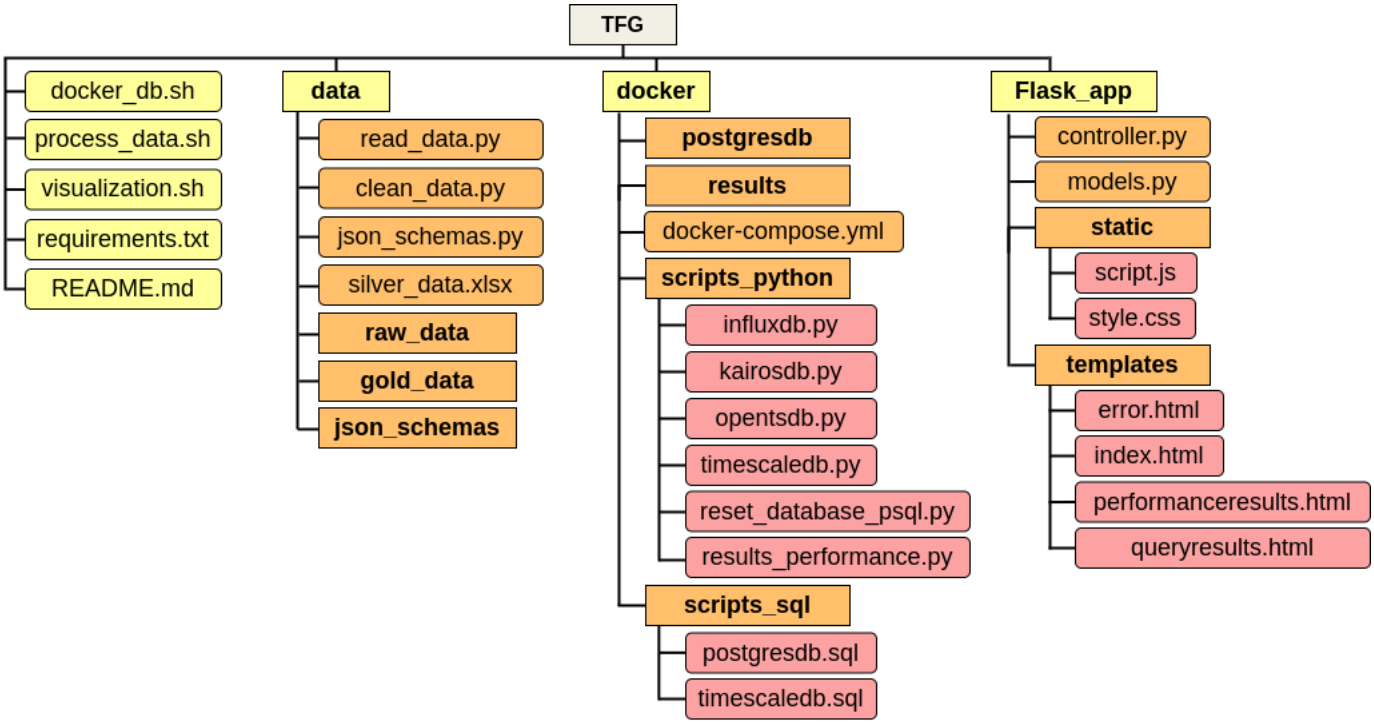


Figura 8.12: Estructura del proyecto

Contamos con un archivo *'requirements.txt'* que define todas las bibliotecas de Python necesarias para la correcta ejecución de todos los programas. Además, el archivo *'README.md'* contiene todas las indicaciones necesarias para poner este trabajo en funcionamiento y que podemos encontrar en la Sección 8.4

Es importante mencionar que todos estos archivos se encuentran disponibles en el repositorio de GitHub `https://github.com/Patagualab/TFG_PatriciaAguado`. Para que los programas de este directorio funcionen correctamente es necesario disponer de la carpeta *raw_data* que contiene información confidencial y que deberá solicitarse al director del trabajo de fin de grado.

Capítulo 9

Conclusiones y trabajo futuro

Este proyecto presenta un análisis del rendimiento de diferentes bases de datos temporales y una herramienta de visualización interactiva de los resultados obtenidos en las diferentes pruebas realizadas. Como resultado de la limpieza de los datos de los que disponíamos, optimizar la utilización de cada base de datos con los recursos que cada una ofrecía y tras establecer tres métricas de rendimiento evaluadas sobre estas bases de datos en distintos escenarios llegamos a varias conclusiones que expondremos a continuación.

Si dirigimos nuestra atención en conseguir el mayor rendimiento de carga posible, TimescaleDB es la mejor opción ya que esta base de datos destaca con diferencia por encima de las demás tanto en la inserción como en la consulta de datos aunque muestre una deficiencia en la compactación que realiza sobre los datos al almacenarlos, que en parte estaría justificado ya que el formato tabular que emplea al ser relacional hace que ocupe mucho más espacio. La peor opción en este caso sería OpenTSDB aunque en general tampoco hemos obtenido buenos resultados en las consultas frente al resto de bases de datos como se puede ver en la Figura 9.1.

En base a los experimentos, centrándonos en el consumo de espacio de los datos, hemos podido observar que en la comparación por el cambio de formato el fichero JSON de TimescaleDB es con diferencia el que menos ocupa porque es de los que menos campos necesita especificar, pero es un precio que no conviene pagar ya que acaba resultando que es la base de datos que peor compacta los datos. Pero en definitiva, si lo que necesitamos priorizar es la obtención del menor tamaño de almacenamiento que la base de datos necesita para hacer frente al desafío actual de almacenar los grandes volúmenes de datos existentes, la mejor de las opciones será KairosDB ya que es indiscutible que Cassandra utiliza métodos óptimos de compactación en el backend de esta base de datos. También merece la pena mencionar que InfluxDB sería otra buena opción para este cometido ya que como comentamos en la Sección 6.1 utiliza un proceso muy meticuloso para reducir el tamaño de los archivos de datos que recibe. Para estas dos últimas bases de datos mencionadas podemos aceptar que aumente el espacio consumido por los datos a la hora de cambiar el formato si a cambio conseguimos que sean las bases de datos que menos ocupan y que mejor comprimen los datos que almacenan.

Si hablamos de las consultas que recuperan datos de forma general, como vemos en la Figura 9.1 tenemos un claro ganador que es TimescaleDB ya que consigue muy buenos resultados en todas las consultas porque como ya comentamos en la Sección 6.1, emplea el concepto de "agregados continuos"; tenemos el uso de un modelo de hipertablas que indexa los datos por la

variable tiempo y a su vez, generamos un índice sobre la columna del identificador de la variable temporal almacenada al crear la base de datos que resulta en una mejora del rendimiento sobre todo cuando intervienen estos dos parámetros en la consulta.

En resumen, a partir de lo expuesto anteriormente y de lo que podemos ver en la Figura 9.1 podemos llegar a las siguientes conclusiones generales:

- El modelo relacional de la mano de TimescaleDB es el que más destaca tras los experimentos realizados aunque no presente resultados competitivos frente al resto de bases de datos a la hora de optimizar el espacio ocupado por los datos dentro de la base de datos.

- El modelo NoSQL de las bases de datos temporales que hemos elegido es el que mejor resultados de almacenamiento nos ofrece porque surgen con la idea de hacer frente al Big Data, pero por otro lado hemos visto que a pesar de estar diseñadas para manejar datos de series temporales hemos observado que KairosDB y OpenTSDB nos han dado resultados bastante malos en la realización de consultas de análisis básico aunque podría deberse a que estén más enfocadas a la recuperación de datos y posterior análisis fuera de la base de datos.

- KairosDB y OpenTSDB han resultado ser buenas para consultas sobre rangos de tiempo y agrupaciones pero los resultados no han estado a la altura de lo esperado para el resto de consultas.

- InfluxDB al contrario que sus competidores dentro de las bases de datos NoSQL ha resultado ser muy eficiente insertando datos, optimizando el almacenamiento y aplicando funciones de análisis básico en las consultas pero nos ha ofrecido malos resultados cuando se trataba de consultas sobre rangos de tiempo y agrupaciones.

Tipo	Consulta	Posicionamiento			
		1º	2º	3º	4º
Inserción	Consulta de carga	TimescaleDB	InfluxDB	KairosDB	OpenTSDB
Almacenamiento	Consulta de almacenamiento	KairosDB	InfluxDB	OpenTSDB	TimescaleDB
Rangos de tiempo	1.A - consulta 1	TimescaleDB	KairosDB	OpenTSDB	InfluxDB
	4.A - consulta 10	TimescaleDB	OpenTSDB	KairosDB	InfluxDB
Análisis básico	2.A - consulta 2	TimescaleDB	InfluxDB	KairosDB	OpenTSDB
	2.B - consulta 3	TimescaleDB	InfluxDB	KairosDB	OpenTSDB
	2.C - consulta 4	TimescaleDB	InfluxDB	KairosDB	OpenTSDB
	2.D - consulta 5	TimescaleDB	InfluxDB	KairosDB	OpenTSDB
	2.E - consulta 6	TimescaleDB	InfluxDB	KairosDB	OpenTSDB
	2.F - consulta 7	TimescaleDB	InfluxDB	X	X
	3.B - consulta 9	TimescaleDB	InfluxDB	KairosDB	OpenTSDB
Agrupación	3.A - consulta 8	TimescaleDB	KairosDB	OpenTSDB	InfluxDB

Figura 9.1: Posicionamiento de las bases de datos con respecto a las consultas realizadas

Líneas de trabajo futuro

En este apartado sugerimos líneas de investigación para la mejora del presente trabajo que consideramos interesantes pero cuyo alcance excede del planificado.

Una posible ampliación del trabajo sería agregar otras bases de datos con un modelo relacional más puro para buscar diferencias con las que ya tenemos y quizás poder comprobar hasta que punto son mejores los sistemas gestores de bases de datos temporales que nos venden con el pretexto de que están enfocadas para optimizar el manejo de este tipo de datos. También se podrían determinar nuevas pruebas que monitoricen la utilización de la CPU (Central Processing Unit) para cada base de datos con el fin de conocer cuál consume más recursos, o bien pruebas de concurrencia para evaluar la capacidad que tiene cada base de datos para manejar múltiples solicitudes, así como profundizar más en la optimización de cada base de datos o comprobar la eficiencia de estas para trabajar en entornos distribuidos.

Otra posible mejora estaría relacionada con la herramienta desarrollada y consistiría en mejorar aspectos de la interfaz de usuario, aplicando nuevas guías de diseño para favorecer la accesibilidad y que todas las personas independientemente de sus habilidades tengan una buena experiencia con la herramienta. Además, podríamos incluir una nueva funcionalidad que le permita al usuario a través de la aplicación diseñar su propia consulta para ejecutarla en las diferentes bases de datos en tiempo real.

Bibliografía

[1] A.Meier and M.Kaufmann. *SQL and NoSQL databases*. Springer Vieweg, 2019.

[2] The OpenTSDB Authors. How does opentsdb work?, 2010. URL: `http://opentsdb.net/overview.html`. Accedido 12-07-2023.

[3] Apache Cassandra. Página principal del software apache cassandra, 2009. URL: `https://cassandra.apache.org/_/index.html`. Accedido 13-07-2023.

[4] European Commission. Making our homes and buildings fit for a greener future. 2021. URL: `https://ec.europa.eu/commission/presscorner/detail/en/fs_21_6691`. Accedido 16-05-2023.

[5] El Norte de Castilla. El edificio lucia de la uva, el primero reconocido a nivel mundial como protegido de contagio de virus, 2020. URL: `https://www.elnortedecastilla.es/valladolid/edificio-lucia-primero-20200630165954-nt.html`. Accedido 22-05-2023.

[6] Paul Dix. Why time series matters for metrics, real-time analytics and sensor data. *An InfluxData Case Study*, pages 9–10, 2021. URL: `https://www.influxdata.com/time-series-database/`. Accedido 05-07-2023.

[7] T. Dunning and E. Friedman. *Time Series Database. New ways to store an access data*. O'Reilly, 2014.

[8] European Commission Department: Energy. Energy efficiency in buildings. 2020. URL: `https://commission.europa.eu/news/focus-energy-efficiency-buildings-2020-02-17_en`. Accedido 16-05-2023.

[9] P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys*, pages 1–32, 2012.

[10] J. Fan, F. Han, and H. Liu. Challenges of big data analysis. *National Science Review*, 2014.

[11] Peter Gardfjäll and Elastisys Company. Kairosdb docker image. URL: `https://hub.docker.com/r/elastisys/kairosdb`. Accedido 09-10-2023.

[12] Lars George. *HBase: The Definitive Guide*. O'Reilly, 2011. URL: `https://github.com/Jayarami123/hadoop-cookbook/blob/master/HBase%EF%BC%9AThe%20Definitive%20Guide.pdf`. Accedido 11-07-2023.

[13] Devndra Ghimire. *Comparative study on Python web frameworks: Flask and Django.* Metropolia University of Applied Sciences, 2010.

[14] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *PODC 2000*, 2004.

[15] Peter Grace. Opentsdb docker image. URL: `https://hub.docker.com/r/petergrace/opentsdb-docker`. Accedido 09-10-2023.

[16] Tianon Gravi, Joseph Ferguson, and contribuidores. Postgresql docker official image. URL: `https://hub.docker.com/_/postgres`. Accedido 09-10-2023.

[17] Brian Hawkins. Query performance, 2022. URL: `https://github.com/kairosdb/kairosdb/wiki/Query-Performance`. Accedido 13-07-2023.

[18] Universidad Jaime I. Apuntes ficheros y bases de datos, 2003. URL: `https://www3.uji.es/~aliaga/e44/Tema_07.pdf`. Accedido 26-05-2023.

[19] InfluxData. Influxdb docker official image. URL: `https://hub.docker.com/_/influxdb`. Accedido 09-10-2023.

[20] InfluxDB. Página principal del software influxdb, 2013. URL: `https://www.influxdata.com/influxdb/`. Accedido 05-07-2023.

[21] Project Management Institute. *Guía de los fundamentos para la dirección de proyectos (Guía del PMBOK®) Tercera edición.* Project Management Institute, 2004.

[22] Solid IT. Clasificación para sistemas gestores de bases de datos, 2012. URL: `https://db-engines.com/en/ranking/time+series+dbms`. Accedido 27-06-2023.

[23] C. S. Jensen and R. T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering, Vol. 11*, pages 1–9, 1999.

[24] KairosDB. Página principal del software kairosdb, 2013. URL: `https://kairosdb.github.io/`. Accedido 13-07-2023.

[25] Simon Kemp. Digital 2023: Global overview report, 2023. URL: `https://datareportal.com/reports/digital-2023-global-overview-report`. Accedido 23-05-2023.

[26] Italo Maia. *Building Web Applications with Flask.* Packt Publishing, 2015.

[27] Mercedes Marqués. *Bases de datos.* Castelló de la Plana: Publicacions de la Universitat Jaume I. Servei de Comunicació i Publicacions, 2011.

[28] Enrique García Miravalles. Tfm. determinación de los estados de funcionamiento y predicción del consumo energético del edificio lucia. URL: `https://uvadoc.uva.es/handle/10324/787/discover`. Accedido 20-06-2023.

[29] The University of Arizona. Christian s. jensen y richard t. snodgrass. URL: `https://timecenter.cs.aau.dk/`. Accedido 31-05-2023.

[30] The University of Arizona. Richard thomas snodgrass biography. URL: `http://www2.cs.arizona.edu/~rts/`. Accedido 31-05-2023.

[31] OpenTSDB. Página principal del software opentsdb, 2010. URL: `http://opentsdb.net/`. Accedido 10-07-2023.

[32] OpenTSDB. Querying or reading data, 2010. URL: `http://opentsdb.net/docs/build/html/user_guide/query/index.html`. Accedido 12-07-2023.

[33] Oracle. ¿qué es big data?, 2021. URL: `https://www.oracle.com/big-data/what-is-big-data/`. Accedido 16-05-2023.

[34] Pandas. pandas.dataframe, 2008. URL: `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html`. Accedido 22-06-2023.

[35] R. Ramakrishnan and J. Gehrke. *Database Management Systems, Second Edition*. McGraw-Hill, 1999. URL: `https://xuanhien.files.wordpress.com/2011/04/database-management-systems-raghu-ramakrishnan.pdf`. Accedido 26-05-2023.

[36] Catherine M. Ricardo. *Bases de datos*. Mc Graw Hill, 2009.

[37] Armin Ronacher. Software jinja2, 2008. URL: `https://jinja.palletsprojects.com/en/3.1.x/`. Accedido 26-10-2023.

[38] Armin Ronacher. Flask. deploying to production, 2010. URL: `https://flask.palletsprojects.com/en/3.0.x/deploying/`. Accedido 26-10-2023.

[39] Armin Ronacher. Framework web flask, 2010. URL: `https://flask.palletsprojects.com/en/3.0.x/`. Accedido 26-10-2023.

[40] David Salomon. *Data Compression. 3rd Edition. The Complete Reference*. Springer, 2005.

[41] Songwon Seo. A review and comparison of methods for detecting outliers in univariate data sets, 2006. URL: `https://d-scholarship.pitt.edu/7948/1/Seo.pdf`. Accedido 28-12-2023.

[42] Bonil Shah, P.M. Jat, and Kalyan Sashidhar. Performance study of time series databases. *arXiv preprint arXiv:2208.13982*, 2022. URL: `https://arxiv.org/abs/2208.13982`. Accedido 26-06-2023.

[43] Siemens. Sistema desigo - construyendo el futuro hoy, 1999. URL: `https://new.siemens.com/es/es/productos/building-technology/automatizacion/desigo.html`. Accedido 19-05-2023.

[44] Richard Snodgrass. The temporal query language tquel. *ACM Transactions on Database Systems, Vol. 12*, pages 1–52, 1987.

[45] Ian Sommerville. *Ingeniería de Software*. Pearson, 2011.

[46] Roger S.Pressman. *Ingeniería del software. Un enfoque práctico*. Mc Graw Hill, 2010.

[47] Andreas Steiner. A generalisation approach to temporal data models and their implementations, 1998. URL: `https://www.timeconsult.com/Publications/Literature.html`. Accedido 01-06-2023.

[48] Rachel Stephens. The state of the time series database market, 2018. URL: `https://redmonk.com/rstephens/2018/04/03/the-state-of-the-time-series-database-market/`. Accedido 02-06-2023.

[49] Petroc Taylor. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025. statista., 2022. URL: `https://www.statista.com/statistics/871513/worldwide-data-created/`. Accedido 24-05-2023.

[50] Kairos DB Team. Documentation. querying data, 2013. URL: `https://kairosdb.github.io/docs/QueryingData.html`. Accedido 13-07-2023.

[51] S. W. Thomas, R. T. Snodgrass, and T. Zhang. τbench: Extending xbench with time. *A TimeCenter Technical Report*, pages 13–26, 2013. URL: `http://www2.cs.arizona.edu/~rts/publications.html`. Accedido 01-06-2023.

[52] Timescale. Timescaledb docker image. URL: `https://hub.docker.com/r/timescale/timescaledb`. Accedido 09-10-2023.

[53] TimeScaleDB. Querying data timescaledb, 1. URL: `https://docs.timescale.com/use-timescale/latest/query-data/`. Accedido 03-07-2023.

[54] TimeScaleDB. Página principal del software timescaledb, 2017. URL: `https://www.timescale.com/`. Accedido 28-06-2023.

[55] TimeScaleDB. Schema management timescaledb, 2023. URL: `https://docs.timescale.com/use-timescale/latest/schema-management/`. Accedido 29-06-2023.

[56] N. Tran, P. Dix, A. Lamb, and M. Mikulicic. Influxdb 3.0: System architecture, 2023. URL: `https://www.influxdata.com/blog/influxdb-3-0-system-architecture/`. Accedido 06-07-2023.

[57] Marta Martínez Vera. La certificación leed a través de sus edificios. edificio lucia, 2020. Trabajo de Fin de Grado. Escuela Técnica Superior de Arquitectura. Universidad de Valladolid. URL: `https://uvadoc.uva.es/handle/10324/44695?locale-attribute=pt_BR`. Accedido 18-05-2023.

[58] Diego Tamayo Alonso y colaboradores. Edificio lucia - universidad de valladolid. URL: `http://edificio-lucia.blogspot.com/`. Accedido 22-05-2023.

Apéndice A

Anexos

A.1. Archivo servicios Docker

Sección que presenta el fichero 'docker-compose.yml' a través del que la herramienta Compose pone en marcha los diferentes servicios definidos en él. El esquema que sigue consiste en definir el nombre que tomará el servicio, la imagen que desea montar en el contenedor y que buscará en Docker Hub, parámetros para el manejo de ficheros internos del contenedor, nombre que le queremos dar al contenedor, variables de entorno que queremos pasar al contenedor, volumen para la persistencia de datos mientras está en ejecución el contenedor y el mapeo del puerto de nuestra máquina local al contenedor de Docker. Es importante mencionar que si no queremos persistencia de datos una vez se pare el contenedor podemos obviar la línea que define un volumen de datos y lo mapea a la máquina local.

```
services:
  postgresdb:
    image: postgres:16.0
    logging:
      options:
        max-size: "10m"
        max-file: "5"
    container_name: postgres_db_container
    environment:
      - POSTGRES_DB=postgres_db
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=admin
    volumes:
      - ./postgresdb/postgres_db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  influxdb:
    image: influxdb:2.7
    logging:
```

```
    options:
      max-size: "10m"
      max-file: "5"
  container_name: influx_db_container
  environment:
    - DOCKER_INFLUXDB_INIT_MODE=setup
    - DOCKER_INFLUXDB_INIT_USERNAME=admin
    - DOCKER_INFLUXDB_INIT_PASSWORD=admin123456789
    - DOCKER_INFLUXDB_INIT_ORG=myorg
    - DOCKER_INFLUXDB_INIT_BUCKET=mybucket
    - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=mytoken
  ports:
    - "8086:8086"

timescaledb:
  image: timescale/timescaledb:latest-pg15
  logging:
    options:
      max-size: "10m"
      max-file: "5"
  container_name: timescale_db_container
  environment:
    - POSTGRES_DB=timescale_db
    - POSTGRES_USER=admin
    - POSTGRES_PASSWORD=admin
  ports:
    - "5434:5432"

opentsdb:
  image: petergrace/opentsdb-docker:latest
  logging:
    options:
      max-size: "10m"
      max-file: "5"
  container_name: opents_db_container
  environment:
    - TSDB_ENABLE_STATS=true
  ports:
    - "4242:4242"

kairosdb:
  image: elastisys/kairosdb:1.2.1
  depends_on:
    - cassandra
  logging:
    options:
      max-size: "10m"
      max-file: "2"
```

```
    container_name: kairos_db_container
    environment:
      - CASSANDRA_HOSTS=cassandra
      - CASSANDRA_PORT=9042
    ports:
      - "8080:8080"

  cassandra:
    image: cassandra:3.11
    logging:
      options:
        max-size: "10m"
        max-file: "2"
    container_name: cassandra_db_container
    environment:
      - CASSANDRA_CLUSTER_NAME=mycluster
    ports:
      - "9042:9042"

volumes:
  postgres_db_data:
```

A.2. Consultas para las bases de datos

A.2.1. Consultas para la base de datos InfluxDB

Sección que contiene las consultas realizadas en las pruebas de InfluxDB utilizando Flux, un lenguaje funcional diseñado específicamente para trabajar con datos de series temporales.

Consulta 1: obtener para un determinado sensor una medición de energía en un día y una hora determinados.

```
from(bucket: "mybucket")
 |> range(start: time(v: "1970-01-01T00:00:00Z"), stop: now())
 |> filter(fn:(r) => r._measurement == "energy" and ({ identificadores }))
 |> filter(fn:(r) => r._time == time(v: "2019-03-02T10:21:20Z"))
```

Consulta 2, 3, 4, 5 y 6: obtener para un determinado sensor la mínima medición de energía (función *min()*), la máxima medición de energía (función *max()*), la media de sus mediciones (función *mean()*), la desviación estándar de sus mediciones (función *stddev()*) y el conteo de sus mediciones (función *count()*).

```
from(bucket: "mybucket")
 |> range(start: time(v: "1970-01-01T00:00:00Z"), stop: now())
 |> filter(fn: (r) => r._measurement == "energy" and ({ identificadores }))
```

```
 |> group(columns: ["device_id"])
 |> min(column: "_value")
 |> sort(columns: ["device_id"])
```

Consulta 7: obtener para un determinado sensor los valores sin duplicados que toman sus mediciones de energía.

```
from(bucket: "mybucket")
 |> range(start: time(v: "1970-01-01T00:00:00Z"), stop: now())
 |> filter(fn: (r) => r._measurement == "energy" and ({ identificadores }))
 |> group(columns: ["device_id"])
 |> distinct(column: "_value")
 |> sort(columns: ["device_id","_time"])
```

Consulta 8_1, 8_2 y 8_3: obtener para un determinado sensor la media de sus mediciones de energía por horas (función *window(every: 1h)*), semanas (función *window(every: 1w, offset: -3d)*) y meses (función *window(every: 1mo)*). En la consulta 8_2 necesitamos añadir el parámetro *offset* con valor '-3d' para añadir un desplazamiento ya que establece los tiempos en base a la época Unix y queremos que tome como inicio de semana el Lunes.

```
from(bucket: "mybucket")
 |> range(start: time(v: "1970-01-01T00:00:00Z"), stop: now())
 |> filter(fn: (r) => r._measurement == "energy" and ({ identificadores }))
 |> group(columns: ["device_id"])
 |> window(every: 1h)
 |> mean()
 |> sort(columns: ["device_id","_time"])
```

Consulta 9: obtener para un determinado sensor el tercer cuartil de sus mediciones de energía.

```
from(bucket: "mybucket")
 |> range(start: time(v: "1970-01-01T00:00:00Z"), stop: now())
 |> filter(fn: (r) => r._measurement == "energy" and ({ identificadores }))
 |> group(columns: ["device_id"])
 |> quantile(column: "_value", q: 0.75)
```

Consulta 10_1, 10_2, 10_3 y 10_4: obtener para un sensor determinado las mediciones de energía en un rango de tiempo determinado de un día, un día y unas horas, un mes, y un mes y unas horas. Obtenemos las diferentes consultas modificando las marcas de tiempo en los parámetros *start* y *stop*.

```
from(bucket: "mybucket")
 |> range(start: time(v: "2019-01-22T00:00:00Z"),
    stop: time(v: "2019-01-23T00:00:00Z"))
 |> filter(fn: (r) => r._measurement == "energy" and ({ identificadores }))
 |> group(columns: ["device_id","_time"])
```

A.2.2. Consultas para la base de datos TimescaleDB

Sección que contiene las consultas realizadas en las pruebas de TimescaleDB utilizando el lenguaje SQL.
Consulta 1: obtener para un determinado sensor una medición de energía en un día y una hora determinados.

```
SELECT device_id, timestamp, device_measurement
FROM energy
WHERE device_id IN ({ identificadores }) AND timestamp='2019-03-02T10:21:20'
```

Consulta 2, 3, 4, 5 y 6: obtener para un determinado sensor la mínima medición de energía (función *MIN()*), la máxima medición de energía (función *MAX()*), la media de sus mediciones (función *AVG()*), la desviación estándar de sus mediciones (función *STDDEV()*) y el conteo de sus mediciones (función *COUNT()*).

```
SELECT device_id, MIN(device_measurement) AS min_energy
FROM energy
WHERE device_id IN ({ identificadores })
GROUP BY device_id
ORDER BY device_id
```

Consulta 7: obtener para un determinado sensor los valores sin duplicados que toman sus mediciones de energía.

```
SELECT device_id, ARRAY_AGG(DISTINCT device_measurement) AS distinct_measurements
FROM energy
WHERE device_id IN ({ identificadores })
GROUP BY device_id
ORDER BY device_id
```

Consulta 8_1, 8_2 y 8_3: obtener para un determinado sensor la media de sus mediciones de energía por horas (función *time_bucket('1 hour', timestamp)*), semanas (función *time_bucket('1 week', timestamp,'1 week'::interval)*) y meses (función *time_bucket('1 month', timestamp)*). En la consulta 8_2 necesitamos añadir el parámetro *'1 week'::interval* para asegurarnos de que el Lunes sea el inicio de semana.

```
SELECT device_id, AVG(device_measurement) AS avg_energy_hour,
    time_bucket('1 hour', timestamp) AS hour
FROM energy
WHERE device_id IN ({ identificadores })
GROUP BY device_id, hour
ORDER BY device_id, hour
```

Consulta 9: obtener para un determinado sensor un intervalo en el que se encuentren los valores atípicos de sus mediciones de energía utilizando el rango intercuartílico. En esta consulta

creamos una tabla temporal calculando los estadísticos que necesitaremos para posteriormente filtrar los datos que queremos obtener, que son las mediciones que no pertenecen al intervalo definido utilizando los cuartiles y el rango intercuartílico.

```
WITH td AS (SELECT device_id,
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY device_measurement) AS q1,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY device_measurement) AS q3
FROM energy
WHERE device_id IN ({ identificadores })
GROUP BY device_id)

SELECT energy.device_id, energy.timestamp, energy.device_measurement
FROM energy INNER JOIN td ON energy.device_id = td.device_id
WHERE energy.device_measurement < (td.q1-(1.5*(td.q3-td.q1)))
OR energy.device_measurement > (td.q3+(1.5*(td.q3-td.q1)))
```

Consulta 10_1, 10_2, 10_3y 10_4: obtener para un sensor determinado las mediciones de energía en un rango de tiempo determinado de un día, un día y unas horas, un mes, y un mes y unas horas. Obtenemos las diferentes consultas modificando las marcas de tiempo en la clausula 'where'.

```
SELECT device_id, timestamp, device_measurement
FROM energy
WHERE device_id IN ({ identificadores })
    AND timestamp >= '2019-01-22T00:00:00'
    AND timestamp < '2019-01-23T00:00:00'
ORDER BY device_id, timestamp
```

A.2.3. Consultas para la base de datos OpenTSDB

Sección que contiene las consultas realizadas en las pruebas de OpenTSDB intercambiando datos a través de JSON mediante peticiones utilizando la API HTTP.
Consulta 1: obtener para un determinado sensor una medición de energía en un día y una hora determinados.

```
"start": 1551518479000,
"end": 1551518480000,
"queries": [{
    "aggregator": "none",
    "metric": "energy",
    "tags": {
        "device_id": { identificadores }
}}]
```

Consulta 2, 3, 4 y 5: obtener para un determinado sensor la mínima medición de energía (parámetros *"aggregator":"min"* y *"downsample":"0all-min"*), la máxima medición de energía

(parámetros *"aggregator":"max"* y *"downsample":"0all-max"*), la media de sus mediciones (parámetros *"aggregator":"avg"* y *"downsample":"0all-avg"*) y la desviación estándar de sus mediciones (parámetros *"aggregator":"dev"* y *"downsample":"0all-dev"*). Las consultas 4 y 5 presentan pequeñas diferencias de cálculo por el método utilizado por OpenTSDB con respecto a las demás bases de datos.

```
"start": "0",
"end": "now",
"queries": [{
    "aggregator": "min",
    "metric": "energy",
    "downsample": "0all-min",
    "tags": {
        "device_id": { identificadores }
}}]
```

Consulta 6: obtener para un determinado sensor el conteo de sus mediciones de energía.

```
"star": "0",
"end": "now",
"queries": [{
    "aggregator": "sum",
    "metric": "energy",
    "downsample": "0all-count",
    "tags": {
        "device_id": { identificadores }
}}]
```

Consulta 8_1, 8_2 y 8_3: obtener para un determinado sensor la media de sus mediciones de energía por horas (parámetro *"downsample":"1h-avg"*), semanas (parámetro *"downsample":"1w-avg"*) y meses (parámetro *"downsample":"1n-avg"*). Estas consultas presentan pequeñas diferencias de cálculo por el método utilizado por OpenTSDB con respecto a las demás bases de datos.

```
"start": "0",
"end": "now",
"queries": [{
    "aggregator": "avg",
    "metric": "energy",
    "downsample": "1h-avg",
    "tags": {
        "device_id": { identificadores }
}}]
```

Consulta 9: obtener para un determinado sensor el tercer cuartil de sus mediciones de energía.

```
"start": "0",
"end": "now",
"queries": [{
    "aggregator": "p75",
    "metric": "energy",
    "downsample": "0all-p75",
    "tags": {
        "device_id": { identificadores }
}}]
```

Consulta 10_1, 10_2, 10_3 y 10_4: obtener para un sensor determinado las mediciones de energía en un rango de tiempo determinado de un día, un día y unas horas, un mes, y un mes y unas horas. Obtenemos las diferentes consultas modificando las marcas de tiempo en los parámetros *start* y *end*.

```
"start": 1548111600000,
"end": 1548198000000,
"queries": [{
    "aggregator": "none",
    "metric": "energy",
    "tags": {
        "device_id": { identificadores }
}}]
```

A.2.4. Consultas para la base de datos KairosDB

Sección que contiene las consultas realizadas en las pruebas de KairosDB intercambiando datos a través de JSON mediante peticiones utilizando la API HTTP.
Consulta 1: obtener para un determinado sensor una medición de energía en un día y una hora determinados.

```
"start_absolute": 1551518480000,
"end_absolute": 1551518480000,
"metrics": [{
   "name": "energy",
   "tags": {
      "device_id": { identificadores }},
      "group_by": [{
         "name": "tag",
         "tags": ["device_id"]}]
}]
```

Consulta 2, 3, 4 y 5: obtener para un determinado sensor la mínima medición de energía (parámetro *"name":"min"*), la máxima medición de energía (parámetro *"name":"max"*), la media de sus mediciones (parámetro *"name":"avg"*) y la desviación estándar de sus mediciones (parámetro *"name": "dev"*). Estas consultas se consiguen muestreando el conjunto de datos, siendo en años el rango más amplio permitido en KairosDB.

```
"start_absolute": 0,
"end_absolute": { instante actual en milisegundos },
"metrics": [{
   "name": "energy",
   "tags": {
      "device_id": { identificadores }},
   "group_by": [{
      "name": "tag",
      "tags": ["device_id"]}],
   "aggregators": [{
      "name": "min",
      "align_sampling": True,
      "align_start_time": True,
      "sampling": {
         "value": 1,
         "unit": "years"}}]
}]
```

Consulta 6: obtener para un determinado sensor el conteo de sus mediciones de energía procesando el resultado, debido a que la consulta nos ofrece el conteo para cada aparición del par marca de tiempo y valor.

```
"start_absolute": 0,
"end_absolute": { instante actual en milisegundos },
"metrics": [{
   "name": "energy",
   "tags": {
      "device_id": { identificadores }},
   "group_by": [{
      "name": "tag",
      "tags": ["device_id"]}],
   "aggregators": [{
      "name": "count"}]
}]
```

Consulta 8_1, 8_2 y 8_3: obtener para un determinado sensor la media de sus mediciones de energía por horas (parámetro *"unit":"hours"*), semanas (parámetro *"unit":"weeks"*) y meses (parámetro *"unit":"months"*).

```
"start_absolute": 0,
"end_absolute": { instante actual en milisegundos },
"metrics": [{
   "name": "energy",
   "tags": {
      "device_id": { identificadores }},
   "group_by": [{
      "name": "tag",
```

```
      "tags": ["device_id"]}],
   "aggregators": [{
      "name": "avg",
      "align_sampling": True,
      "align_start_time": True,
      "sampling": {
         "value": 1,
         "unit": "hours"}}]
}]
```

Consulta 9: obtener para un determinado sensor el tercer cuartil de sus mediciones de energía.

```
"start_absolute": 0,
"end_absolute": { instante actual en milisegundos },
"metrics": [{
   "name": "energy",
   "tags": {
      "device_id": { identificadores }},
   "group_by": [{
      "name": "tag",
      "tags": ["device_id"]}],
   "aggregators": [{
      "name": "percentile",
      "percentile": 0.75,
      "align_sampling": True,
      "align_start_time": True,
      "sampling": {
         "value": 1,
         "unit": "years"}}]
}]
```

Consulta 10_1, 10_2, 10_3 y 10_4: obtener para un sensor determinado las mediciones de energía en un rango de tiempo determinado de un día, un día y unas horas, un mes, y un mes y unas horas. Obtenemos las diferentes consultas modificando las marcas de tiempo en los parámetros *start_absolute* y *end_absolute*.

```
"start_absolute": 1548111600000,
"end_absolute": 1548198000000,
"metrics": [{
   "name": "energy",
   "tags": {
      "device_id": { identificadores }},
   "group_by": [{
      "name": "tag",
      "tags": ["device_id"]}]
}]
```

A.2.5. Archivos de resultados

Tras realizar las pruebas de rendimiento obtenemos un fichero JSON para cada combinación de tamaño de datos, número de variables consultadas y base de datos con el resultado de las consultas realizadas como se puede ver en la Figura A.1. Por otro lado, obtenemos un fichero XLSX con los resultados de rendimiento, en la Figura A.2 podemos ver la unión de las diferentes páginas del libro, en total tendremos una para la métrica de inserción de datos ('m_data'), otra para la métrica de almacenamiento ('m_space') y cuatro para la métrica de latencia de consultas en función del tamaño de datos utilizado ('m_query').


```
{
  "query": "q_4",
  "size_q": 3,
  "info_query": "Obtener para un determinado sensor la media de sus mediciones de energia",
  "results": [
    {
      "device_id": "AInt'PS'AnlzRed21'En1",
      "timestamp": "null",
      "value": 736.3585826771654
    }
  ]
},
{
  "query": "q_5",
  "size_q": 3,
  "info_query": "Obtener para un determinado sensor la desviacion estandar de sus mediciones de energia",
  "results": [
    {
      "device_id": "AInt'PS'AnlzRed21'En1",
      "timestamp": "null",
      "value": 10.85726650349525
    }
  ]
},
{
  "query": "q_6",
  "size_q": 3,
  "info_query": "Obtener para un determinado sensor el conteo de sus mediciones de energia",
  "results": [
    {
      "device_id": "AInt'PS'AnlzRed21'En1",
      "timestamp": "null",
      "value": 254
    }
  ]
},
{
  "query": "q_7",
  "size_q": 3,
  "info_query": "Obtener para un determinado sensor los valores sin duplicados que toman sus mediciones de energia",
  "results": [
    {
      "device_id": "AInt'PS'AnlzRed21'En1",
      "timestamp": "null",
      "value": 738.26
    },
    {
      "device_id": "AInt'PS'AnlzRed21'En1",
      "timestamp": "null",
      "value": 738.28
    },
    {
      "device_id": "AInt'PS'AnlzRed21'En1",
      "timestamp": "null",
      "value": 738.36
    },
```


Figura A.1: Ejemplo de la salida para los resultados de las consultas en las pruebas para 3 meses y 1 variable

m_data

id_exp	db_exp	size_exp	time_exp_ms
1	timescaledb	3	536.990
2	timescaledb	3	696.913
3	timescaledb	3	1156.677
4	timescaledb	6	2518.505
5	timescaledb	6	2052.662
6	timescaledb	6	2130.463
		...	
7	influxdb	10	35671.598
8	influxdb	10	38035.283
9	influxdb	10	34102.799
10	influxdb	13	56441.211
		...	
1	opentsdb	3	2.358
		...	
1	kairosdb	3	4032.485
		...	

m_space

id_exp	db_exp	size_exp	space_exp_mb
1	jsontimescale	3	6.770
2	timescaledb	3	13.762
3	jsontimescale	3	6.770
4	timescaledb	3	13.716
5	jsontimescale	3	6.770
6	timescaledb	3	13.708
7	jsontimescale	6	31.116
8	timescaledb	6	27.052
		...	
17	jsoninflux	10	119.438
18	influxdb	10	12.043
19	jsoninflux	13	195.261
		...	
13	jsonopen	10	109.478
14	opentsdb	10	47.266
		...	
1	jsonkairos	3	7.809
2	kairosdb	3	1.816
		...	

m_query_X

id_exp	db_exp	type_q	n_q	n_var	size_q	time_exp_ms
1	timescaledb	ep	1	1	all	1.636
2	timescaledb	agg	2	1	all	2.951
3	timescaledb	agg	3	1	all	1.406
			...			
7	influxdb	agg	7	1	all	20.355
8	influxdb	gb	8	1	hour	52.751
			...			
24	opentsdb	gb	8	5	week	85.566
25	opentsdb	gb	8	5	month	56.697
26	opentsdb	gb	9	5	all	183.945
			...			
43	kairosdb	tr	10	20	dayhour	40.037
44	kairosdb	tr	10	20	month	49.046
			...			

Figura A.2: Ejemplo de la salida para los resultados de rendimiento en las pruebas (Figura de elaboración propia)