



Universidad
Zaragoza

Trabajo Fin de Grado

Evaluación de técnicas de procesamiento de flujos de
datos en ciudades inteligentes

Evaluation of techniques for data stream processing
in smart cities

Autor

Cristina Marzo Pardos

Director

Sergio Ilarri Artigas

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2024



DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe remitirse a seceina@unizar.es dentro del plazo de depósito)

D./D^a. Cristina Marzo Pardos ,

en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza, Declaro que el presente Trabajo de Fin de Estudios de la titulación de Grado en Ingeniería Informática (Título del Trabajo)

Evaluación de técnicas de procesamiento de flujos de datos en ciudades inteligentes

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 29 de noviembre de 2024

Firmado por MARZO PARDOS,
CRISTINA (FIRMA) el día
29/11/2024 con un
certificado emitido por AC
DNIE 006

Fdo:

AGRADECIMIENTOS

Me gustaría agradecer en primer lugar a Sergio Ilarri, director de este proyecto, su confianza para la realización del mismo, la ayuda que me ha proporcionado en todo momento, y su monitorización del estado del proyecto.

También quiero agradecer a mi familia su apoyo incondicional en los momentos difíciles vividos durante la realización de este trabajo, y también del grado que finaliza.

No me quiero olvidar de nombrar a todas las personas que han sido mis compañeros de viaje durante los 4 años del grado, compañeros de clase y prácticas, algunos de los cuales considero amigos, que con un granito de arena cada uno me han hecho un poco más fácil la experiencia universitaria.

Finalmente, también es importante agradecer el apoyo por parte del proyecto de I+D+i PID2020-113037RB-I00, financiado por MICIU/AEI/10.13039/501100011033 (proyecto NEAT-AMBIENCE) y liderado por la Universidad de Zaragoza.

Evaluación de técnicas de procesamiento de flujos de datos en ciudades inteligentes

RESUMEN

Gracias a los avances en las tecnologías de la información y la comunicación, el concepto de ciudad inteligente ha surgido como una manera de abordar los retos urbanos, mejorando así la eficiencia, sostenibilidad y calidad de vida de la población.

Un componente esencial de las ciudades inteligentes es la necesidad de generar, recopilar, procesar y analizar grandes cantidades de datos continuos en tiempo real, lo que supone un gran reto tecnológico. En este marco, el procesamiento de flujos de datos resulta fundamental para tomar decisiones basadas en datos de manera inmediata.

En este trabajo se ha realizado una evaluación de técnicas de procesamiento de flujos de datos en el contexto de las ciudades inteligentes, desarrollando una herramienta que simula la generación de flujos de datos continuos, permite la visualización de los datos cargados en los flujos, inicia consultas sobre ellos y permite la visualización de sus resultados sobre mapas, a modo de ejemplo de interfaz de usuario de un sistema de ayuda a la toma de decisiones en el ámbito urbano.

En concreto, la herramienta permite la ingesta de datos en los sistemas Apache Kafka, Apache Pulsar y Redis (mediante su estructura de datos *Streams*), en escenarios cotidianos en el ámbito de las ciudades inteligentes, como la movilidad de vehículos, la gestión de peajes (asimilable a las zonas de bajas emisiones en los centros de las ciudades) y accidentes en una autopista, y la gestión de aparcamientos.

El trabajo se encuadra dentro del proyecto de I+D+i PID2020-113037RB-I00, financiado por MICIU/AEI/10.13039/501100011033 (proyecto NEAT-AMBIENCE) y liderado por la Universidad de Zaragoza.

Evaluation of techniques for data stream processing in smart cities

ABSTRACT

Thanks to advances in information and communication technologies, the concept of the smart city has emerged as a way to address urban challenges, thereby improving efficiency, sustainability, and the quality of life of the population.

An essential component of smart cities is the need to generate, collect, process, and analyze large volumes of continuous, real-time data, which poses a significant technological challenge. In this context, data stream processing becomes fundamental for making immediate, data-driven decisions.

In this work, an evaluation of data stream processing techniques has been carried out in the context of smart cities. A tool was developed to simulate the generation of continuous data streams, enable the visualization of data loaded into these streams, initiate queries on them, and display query results on maps, as an example of a user interface for a decision-support system in the urban domain.

Specifically, the tool allows the ingestion of data into Apache Kafka, Apache Pulsar, and Redis (using its *Streams* data structure) systems in common scenarios related to smart cities, such as vehicle mobility, toll (comparable to low-emission zones in city centers) and accidents management in highways, and parking management.

This work has been carried out within the framework of the R&D project PID2020-113037RB-I00, funded by MICIU/AEI/10.13039/501100011033 (NEAT-AMBIENCE project) y led by the Universidad de Zaragoza.

Índice

1. Introducción y objetivos	1
1.1. Motivación y contexto del trabajo	1
1.2. Objetivos y alcance del trabajo	2
1.3. Metodología y herramientas	2
1.4. Estructura de la memoria	3
2. Contexto tecnológico y trabajos previos	5
2.1. Conceptos básicos	5
2.1.1. Sistemas Gestores de Bases de Datos	5
2.1.2. Big Data	6
2.1.3. Flujos de datos	8
2.2. Estado del arte de las tecnologías de manipulación y procesamiento de flujos de datos	9
2.2.1. Sistemas de mensajería y streaming de datos	9
2.2.2. Frameworks de procesamiento de datos distribuido	13
2.2.3. Frameworks de análisis de datos espaciales	18
2.2.4. Trabajos académicos previos	21
3. Planteamiento y desarrollo del sistema	25
3.1. Selección de las tecnologías de manipulación y procesamiento de flujos de datos a utilizar	25
3.1.1. Tecnologías descartadas	25
3.1.2. Tecnologías utilizadas	26
3.2. Arquitectura propuesta	27
3.3. Tecnologías utilizadas en la implementación de la arquitectura	28
3.3.1. Back-end	28
3.3.2. Front-end	28
3.4. Entorno utilizado	29
3.5. Desarrollo de la herramienta	30
3.5.1. Back-end	30

3.5.2. Front-end	35
4. Evaluación de técnicas de procesamiento de flujos de datos	37
4.1. Escenarios de aplicación considerados	37
4.2. Técnicas evaluadas	38
4.3. Consultas implementadas	39
4.3.1. Simulación de movilidad de vehículos en una ciudad	39
4.3.2. Simulación de gestión de peajes en una autopista	40
4.3.3. Simulación de gestión de un aparcamiento	40
4.4. Evaluación cualitativa	41
4.4.1. Programación de las consultas	41
4.4.2. Implementación de las consultas	42
4.5. Evaluación cuantitativa	45
4.5.1. Procesador de consultas con funciones de predicción	45
4.5.2. Rendimiento de los sistemas	46
5. Problemas encontrados y conclusiones del trabajo	51
5.1. Problemas encontrados	51
5.1.1. Base de datos	51
5.1.2. Conflictos de versiones Java	52
5.1.3. Conflicto de dependencias internas de bibliotecas	52
5.1.4. Generación de datos para el benchmark Linear Road	52
5.1.5. Paquete del componente back-end demasiado pesado	53
5.1.6. Generación del contenido estático del componente front-end	53
5.2. Conclusiones y trabajo futuro	54
5.2.1. Trabajo realizado y conclusión personal	54
5.2.2. Conclusiones del proyecto	57
5.2.3. Trabajo futuro	57
Bibliografía	59
Lista de Figuras	63
Lista de Tablas	67
Anexos	68
A. Análisis de requisitos	71
A.1. Requisitos no funcionales	71

A.2. Requisitos funcionales	71
B. Fragmentos de código destacados	73
C. Installation guide	77
C.1. External systems (database, data streaming systems)	77
C.2. Back-end	78
C.2.1. Docker	78
C.2.2. Native (java + gradle)	79
C.3. Front-end	79
C.3.1. Docker (web application)	79
C.3.2. Native (web application)	80
C.3.3. Native (desktop application)	80
D. User's manual	81
D.1. Introduction	81
D.2. Starting the application	81
D.3. Home screen	81
D.4. Lateral panel	82
D.4.1. Streaming servers	82
D.5. Streaming data	83
D.6. Simulations	85
D.6.1. Select a simulation	85
D.6.2. Import simulations	86
D.6.3. Export simulations	87
D.7. Continuous queries	87
D.8. Logging	88
D.9. Configuration	89

Capítulo 1

Introducción y objetivos

En este capítulo se introduce la motivación del trabajo (Sección 1.1), sus objetivos (Sección 1.2), la metodología que se ha seguido para realizarlo (Sección 1.3) y la estructura de la memoria (Sección 1.4).

1.1. Motivación y contexto del trabajo

Debido a los avances tecnológicos producidos en las últimas décadas, ha surgido el concepto de ciudad inteligente como una forma de abordar los desafíos urbanos y mejorar la calidad de vida de sus habitantes, así como el uso racional de los recursos disponibles.

Uno de los pilares fundamentales de las ciudades inteligentes es la capacidad de producir, recopilar, procesar y analizar grandes volúmenes de datos en tiempo real. En este contexto, el procesamiento de flujos de datos se vuelve crucial para tomar decisiones informadas en tiempo real.

En el contexto del proyecto de investigación NEAT-AMBIENCE (<http://webdiis.unizar.es/~silarri/NEAT-AMBIENCE/>) se plantea el desarrollo de técnicas de gestión de datos novedosas para ayudar a las personas a tomar mejores decisiones ante los desafíos a los que nos enfrentamos como sociedad. En particular, en este TFG se evalúan técnicas de gestión de datos que pueden utilizarse para procesar flujos de datos (*data streams*) en ciudades inteligentes (*smart cities*).

1.2. Objetivos y alcance del trabajo

Los objetivos sobre los que se ha elaborado este proyecto son:

1. Examinar el estado del arte de las tecnologías de manipulación y procesamiento de flujos de datos.
2. Plantear escenarios de aplicación en ciudades inteligentes donde se exploten datos como apoyo a la toma de decisiones.
3. Desarrollar una herramienta para evaluar técnicas de gestión de flujos de datos en ciudades inteligentes.
4. Desarrollar una herramienta para simular los escenarios de aplicación y procesar distintos tipos de preguntas (espaciales y espacio-temporales) sobre los flujos de datos generados.

Por lo tanto, el objetivo principal es la evaluación de técnicas de procesamiento de flujos de datos, creando una herramienta apropiada para ello, contrastando varias tecnologías elegidas de entre las estudiadas en el estado del arte sobre los escenarios de aplicación escogidos, simulando la generación de datos y permitiendo el procesamiento de consultas continuas sobre ellos.

1.3. Metodología y herramientas

El desarrollo del trabajo se ha dividido en las siguientes fases:

1. Estudio del estado del arte de las tecnologías de manipulación y procesamiento de flujos de datos y de trabajos previos.
2. Selección de las tecnologías a utilizar.
3. Desarrollo y validación de la herramienta.
4. Evaluación y validación en diferentes escenarios.
5. Elaboración de la documentación.

En las fases tercera y cuarta, para poder abordar este trabajo de forma eficiente respecto al tiempo invertido, se han utilizado metodologías *kanban* y *scrum*.

Para ello se han definido iteraciones scrum a partir de la división del trabajo a realizar en diversas fases. De esta forma, tras tener un producto mínimo viable funcional, se han ido añadiendo nuevas características, correspondientes a los requisitos pendientes.

1.4. Estructura de la memoria

Esta memoria inicia introduciendo conceptos básicos sobre los que se sustentan las tecnologías evaluadas en el trabajo, y presentando el estado del arte realizado sobre ellas, con el Capítulo 2.

Después, se presenta el diseño del sistema, con las tecnologías finalmente escogidas para implementarlo, la arquitectura que va a seguir, el entorno en el que se ha validado, y un resumen de los elementos más destacados de su desarrollo, con el Capítulo 3.

Seguidamente, se aborda el propósito fundamental de este trabajo, que es evaluar la facilidad de implementación y el soporte al usuario técnico que ofrecen los sistemas escogidos, y comparar los rendimientos obtenidos con ellos, que se recoge en el Capítulo 4.

Para finalizar con la parte principal de esta memoria, se detallan los problemas y errores más importantes que han surgido durante la realización del proyecto junto a las conclusiones finales del trabajo, en el Capítulo 5.

Comenzando ya con los anexos, se detallan los requisitos de la aplicación desarrollada, en el Anexo A; algunos fragmentos de código fuente referenciados desde la memoria, con el Anexo B; una guía de instalación de la aplicación (en inglés, puesto que se ofrece una interfaz en dicho idioma) en el Anexo C; y, por último, un manual de usuario, también de nuevo en inglés, en el Anexo D.

Capítulo 2

Contexto tecnológico y trabajos previos

En la Sección 2.1 se abordan los fundamentos técnicos utilizados por las tecnologías de manipulación y procesamiento de flujos de datos descritas en la Sección 2.2. Por último, la Sección 2.2.4 presenta los aspectos clave de trabajos previos realizados en el ámbito de este proyecto.

2.1. Conceptos básicos

Para comprender las sistemas y frameworks detallados en la Sección 2.2, es esencial familiarizarse con algunos conceptos básicos comunes a todos ellos, como la estructura y el funcionamiento de sistemas de procesamiento de datos, nociones relacionadas con Big Data, y los principios fundamentales del análisis geoespacial.

2.1.1. Sistemas Gestores de Bases de Datos

A continuación se va a resumir los hitos más relevantes en la evolución histórica de los Sistemas Gestores de Bases de Datos.

Modelo relacional

En 1970, el modelo relacional planteado en [1] revolucionó el tratamiento de información, solucionando los problemas más destacados de los modelos jerárquico y en red [2] de los sistemas navegacionales [3], es decir, la dependencia de las aplicaciones frente a los cambios en la representación interna de la información y las inconsistencias de datos dentro del mismo modelo.

A partir de este modelo surgen los sistemas gestores de bases de datos relacionales (SGBDR / RDBMS, *Relational DataBase Management Systems*), que han sido durante décadas la plataforma tecnológica por excelencia de la gestión de datos, gracias a su

capacidad para manejar datos garantizando su integridad y consistencia a través de las transacciones **ACID** (Atomicidad *Atomicity*, Consistencia *Consistencia*, Aislamiento (*Isolation*) y Durabilidad *Durabilidad*). Además, con la publicación del estándar SQL-86 [4] en 1987, se estableció un lenguaje de consulta estándar para interactuar con ellas.

Modelo relacional extendido

Tras las distintas demandas de nuevos tipos de SGBD en la década de 1990, como el modelo orientado a objetos [5], una extensión del modelo relacional que permita modelar objetos [6], o un nuevo lenguaje de base de datos relacional [7] [8]; se publicó la versión SQL-99 [9] que incluyó operadores OLAP [10] (*OnLine Analytical Process*) y el modelo objeto-relacional, correspondiente al segundo manifiesto referenciado en [6].

NoSQL

El crecimiento exponencial en la cantidad de datos a procesar y la necesidad de hacerlo en tiempo real y de forma distribuida ha propiciado la aparición de los sistemas gestores de bases de datos NoSQL (no relacionales / *Not Only SQL*) [11].

Flexibilizando algunos requisitos de las transacciones ACID siguiendo el teorema CAP [12], están diseñados para escalar horizontalmente en arquitecturas distribuidas. Además, cada sistema con su enfoque particular (e.g. documentos, columnas o clave-valor) ofrece un modelo de datos más flexible, permitiendo incluso en algunos casos tener diferentes elementos en la misma entidad con distintos atributos.

NewSQL

Precisamente el incumplimiento de las transacciones ACID y la complejidad en la gestión de la consistencia de los datos en los sistemas NoSQL llevaron al surgimiento de los sistemas gestores de bases de datos NewSQL [13] [14], buscando combinar lo mejor de ambos mundos: escalabilidad y flexibilidad con garantías transaccionales y la familiaridad del modelo relacional.

Estos sistemas siguen siendo relacionales, pero proporcionan mejores opciones de escalabilidad para OLTP (*OnLine Transaction Processing*), aprovechando las tecnologías de almacenamiento distribuido y procesamiento paralelo.

2.1.2. Big Data

Con estas necesidades también surge el concepto de **Big Data** [15], caracterizado por el manejo de conjuntos de datos de gran Volumen, Velocidad y Variedad, conocidas

como las 3Vs del Big Data [16]. Este cambio de escala en el manejo de datos también demandó nuevos enfoques y herramientas, dando lugar a la utilización de problemas/paradigmas de formas innovadoras.

MapReduce

El paradigma MapReduce [17], popularizado por Google y posteriormente implementado en sistemas como Apache Hadoop, divide el procesamiento de datos en dos fases principales:

- **Map**, en la que los datos se dividen en pares clave-valor y se procesan en paralelo.
- **Reduce**, en la que los resultados de la fase Map se procesan para obtener el resultado final.

Además, también se realiza una fase intermedia, denominada **Shuffle**, que agrupa todos los valores resultantes de la fase Map (aunque los procesos Map estén distribuidos entre diversos computadores), en función de sus claves, y sirve como entrada a la fase Reduce.

En la Figura 2.1 se muestra un esquema del funcionamiento del paradigma MapReduce.

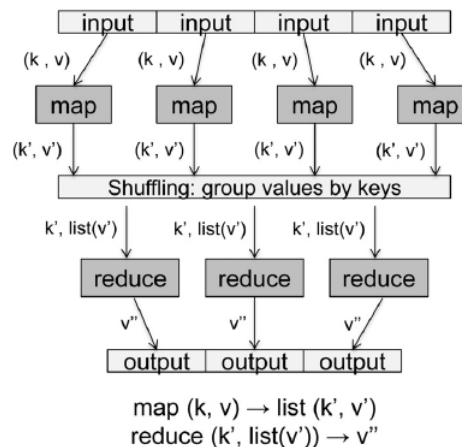


Figura 2.1: El modelo de programación MapReduce. Fuente: [18]

Productor-Consumidor

Otro paradigma fundamental para el procesamiento en tiempo real es el **productor-consumidor**, planteado por Dijkstra en [19]. En este modelo dos procesos, uno como productor y otro como consumidor, comparten un buffer de tamaño finito, sin que el productor exceda la capacidad del buffer ni el consumidor intente obtener

información del buffer vacío. Este modelo es especialmente útil en sistemas donde los datos son generados continuamente y necesitan ser procesados de inmediato para garantizar una respuesta rápida.

2.1.3. Flujos de datos

En este contexto, el concepto de **flujos de datos** (o *data streams*) adquiere particular relevancia. Son secuencias continuas de datos que se generan y transmiten en tiempo real. A diferencia del almacenamiento y procesamiento en lotes tradicional, los flujos de datos son cambiantes y deben ser analizados en tiempo real para poder extraer información.

El procesamiento de flujos de datos en tiempo real requiere de técnicas especializadas, como el procesamiento en memoria y el uso de ventanas temporales. Sin embargo, también presenta nuevas problemáticas, como la latencia, la necesidad de escalabilidad y el manejo adecuado de eventos desordenados o retrasados. La integridad de los datos y la capacidad para realizar análisis en tiempo real son los aspectos más críticos.

También se han dado definiciones conjuntistas exhaustivas de los flujos de datos y sus operadores, como las que se pueden consultar en [20] y [21].

Sistemas Gestores de Flujos de Datos

La necesidad de procesamiento de datos en tiempo real, impulsada por estas definiciones formales y los desafíos asociados, llevó a la aparición de los sistemas gestores de flujos de datos (DSMS, *Data Stream Management Systems*) y los sistemas de procesamiento de flujos de datos en tiempo real, cuyas características esenciales se establecen en [22].

2.2. Estado del arte de las tecnologías de manipulación y procesamiento de flujos de datos

La capacidad de procesar y manipular grandes volúmenes de datos en tiempo real se ha convertido en un factor crucial para el éxito de muchas empresas. La evolución de las tecnologías de procesamiento y manipulación de datos ha permitido abordar estos desafíos con soluciones cada vez más eficientes y escalables. Esta sección ofrece una visión general del estado del arte en este campo, explorando las principales tecnologías utilizadas actualmente para el procesamiento y manipulación de datos, debiendo destacar que todas ellas son de código abierto.

A continuación, se detallan algunas de las herramientas y framework más relevantes en este ámbito, incluyendo sus características y aplicaciones específicas.

2.2.1. Sistemas de mensajería y streaming de datos

Los sistemas de mensajería y streaming de datos son fundamentales para la gestión y procesamiento de grandes volúmenes de datos en tiempo real. Estos sistemas permiten manejar flujos continuos de datos, facilitando la integración y análisis de información proveniente de diversas fuentes.

En la Tabla 2.1, se presentan y comparan las características principales de tres soluciones destacadas en este campo: Apache Kafka, Apache Pulsar y Redis Streams. Esta tabla resume aspectos clave como el propósito principal, el modelo de procesamiento, la escalabilidad, y la facilidad de uso de cada sistema.

Apache Kafka

Apache Kafka (<https://kafka.apache.org/>) es una plataforma de transmisión de eventos distribuida principalmente utilizada para implementar pipeline de datos de alto rendimiento, analítica sobre flujos de datos, integración de datos y aplicaciones críticas de tiempo real. Ofrece interfaces para los lenguajes de programación Java, Scala, C, C++, Python, Go y .NET/C#.

Los eventos se organizan y almacenan de manera persistente en *topics*. Utiliza el modelo productor-consumidor, de modo que a los *topics* se puede añadir datos desde cualquier número de productores, y los datos que un *topic* contiene pueden ser leídos por cualquier número de consumidores. Cada *topic* tiene varias particiones distribuidas a lo largo del clúster.

Además, ofrece Kafka Streams (<https://kafka.apache.org/documentation/streams/>), una biblioteca para Java y Scala para implementar procesamiento de flujos de datos en el lado del cliente, es decir, ejecutándose en el programa que realice la consulta, no en el servidor de Apache Kafka.

Apache Pulsar

Apache Pulsar (<https://pulsar.apache.org/>) es un sistema de mensajería distribuido diseñado para manejar grandes volúmenes de datos en tiempo real con alta escalabilidad y baja latencia. Destaca por su separación entre almacenamiento y procesamiento, delegando el almacenamiento a Apache BookKeeper (<https://bookkeeper.apache.org/>), lo que permite escalar de manera independiente las dos partes. Proporciona interfaces para los lenguajes de programación Java, C, C++, Python, Go y .NET/C#.

Proporciona tres formas de suscribirse a un **topic**:

- **Exclusive**, que solamente permite a un consumidor utilizar la suscripción.
- **Failover**, en la que varios consumidores pueden utilizar la misma suscripción, pero solo uno de ellos elegido al azar es el que recibe los mensajes, hasta que se desconecte del sistema.
- **Shared**, en la que varios consumidores pueden utilizar la misma suscripción, y los mensajes se entregan siguiendo el algoritmo **Round Robin** [23], garantizando que cada mensaje solamente se entrega a uno de ellos.

Además, proporciona Pulsar functions (<https://pulsar.apache.org/docs/next/functions-overview/>), un framework de procesamiento de flujos de datos que se ejecuta en el servidor de Apache Pulsar. Estas funciones se pueden programar en Java, Python o Go, pero una vez programadas se tienen que cargar en el servidor.

Redis

Redis (<https://redis.io/>) es un Sistema Gestor de Bases de Datos (DBMS, *DataBase Management System*) distribuido de alto rendimiento de tipo NoSQL y, en particular, con un almacenamiento basado en tuplas clave-valor en memoria, aunque ofrece distintos niveles de persistencia en disco. Ofrece interfaces para los lenguajes de programación Java, Python, Go, C#/.NET y Node.js, además de bibliotecas OM (Object Mapping) para Java (con el framework Spring), Python, C#/.NET y Node.js.

Suele utilizarse como cache, base de datos documental y/o vectorial y como motor de transmisión de flujos de datos, dado que ofrece una gran variedad de tipos de datos.

Se ha incluido en este apartado debido a su tipo de datos **Streams** (<https://redis.io/docs/latest/develop/data-types/streams/>), una estructura de datos de tipo log que permite almacenar y procesar secuencias de eventos en tiempo real. También ofrece Redis functions (<https://redis.io/docs/latest/develop/interact/programmability/functions-intro/>), que es una API para gestionar código ejecutado en el servidor de Redis. En anteriores versiones, Redis admite el uso de scripts programados en Lua, y aunque teóricamente Redis functions da un paso más y «no hace suposiciones sobre el lenguaje de implementación» de la función, en la práctica Redis solamente incluye un motor de ejecución de Lua. El código de la función tiene que ser subido al servidor antes de intentar ejecutarla.

Tabla 2.1: Comparativa de características de Apache Kafka, Apache Pulsar y Redis Streams

Característica	Apache Kafka	Apache Pulsar	Redis Streams
Propósito Principal	Plataforma de streaming de datos	Sistema de mensajería y streaming en tiempo real	Gestión de flujos de datos en memoria
Modelo de Procesamiento	Streaming	Streaming	Streaming
Compatibilidad de Datos	Estructurados y no estructurados	Estructurados y no estructurados	Estructurados y no estructurados
Escalabilidad	Alta	Alta	Media
Procesamiento en Tiempo Real	Sí	Sí	Sí
Lenguaje de Programación Soportado	Java, Scala, C, C++, Python, Go, .NET/C#	Java, C, C++, Python, Go, .NET/C#	Varios (a través de clientes)
Casos de Uso Típicos	Ingesta y procesamiento de flujos de datos	Mensajería, transmisión de datos en tiempo real, procesamiento de eventos	Transmisión de datos en tiempo real, pub/sub
Integración con otras herramientas	Alta (Spark, Flink)	Alta (Flink, Storm, Spark)	Alta (con otros servicios de Redis, Spark, Flink)
Almacenamiento Persistente	Sí	Sí	Opcional
Modelo de Consumo	Productor-Consumidor	Productor-Consumidor	Productor-Consumidor

2.2.2. Frameworks de procesamiento de datos distribuido

El procesamiento de datos distribuido es una técnica crucial en la era del Big Data, permitiendo el manejo y análisis de grandes volúmenes de información de manera eficiente a través de múltiples nodos en un clúster. Los frameworks de procesamiento de datos distribuido han evolucionado para abordar las crecientes demandas de escalabilidad, rendimiento y flexibilidad en el tratamiento de datos.

La Tabla 2.2 proporciona una comparación exhaustiva de los frameworks detallados en la sección, destacando aspectos clave como el propósito principal, el modelo de procesamiento, la compatibilidad de datos, y las capacidades de escalabilidad.

Apache Beam

Apache Beam (<https://beam.apache.org/>) es un modelo de programación unificado para definir y ejecutar flujos de trabajo de procesamiento de datos, tanto por lotes como en tiempo real. Permite crear pipeline de datos utilizando una única API que puede implementarse en múltiples motores de ejecución, como Apache Flink o Apache Spark, a través de sus *runners*.

Ofrece interfaces para los lenguajes de programación Java, Python, Go y TypeScript, proporcionando soporte para operadores de procesamiento en paralelo, operadores de ventana y disparadores. Además, permite implementar procesamiento con estado y temporizadores.

El modelo de procesamiento de datos en Apache Beam se basa en los siguientes conceptos clave:

- **Pipeline:** engloba todo el procesamiento de datos que se va a realizar con Apache Beam.
- **PCollection:** conjunto de datos distribuido sobre el que opera una *Pipeline*.
- **PTransform:** operación de procesamiento de datos en una *Pipeline*.
- **I/O Transforms:** *Ptransforms* de entrada/salida.

Apache Flink

Apache Flink (<https://flink.apache.org/>) es un framework y motor de procesamiento de datos distribuido para realizar computación distribuida con estado, escalable y tolerancia a fallos sobre flujos de datos en tiempo real y por lotes. Ofrece interfaces para los lenguajes de programación Java, Scala y Python.

Se diferencia de otros framework similares, como Apache Spark, en que fundamentalmente explota la optimización de los planes de ejecución. [18]

Apache Flink ofrece varias API en función del nivel de abstracción deseado, que ordenadas de mayor a menor son las siguientes:

- **SQL:** representa programas como consultas SQL. Esta API interactúa con la *Table API* para poder ejecutar las consultas sobre *tables*.
- **Table API:** DSL declarativo centrado en *tables*, que pueden cambiar dinámicamente al representar flujos de datos. Esta API sigue el modelo relacional extendido, dado que cada *table* tiene un esquema asociado y ofrece operaciones como *select*, *project*, *join*, *group by*, *aggregate*, etc.
- **DataStream API:** ofrece operadores de *join*, *aggregate*, *window*, etc.
- **DataSet API:** ofrece operadores adicionales para procesamiento por lotes, como iteradores y bucles.
- **Stateful Stream Processing:** ofrece procesamiento de flujos de datos con estado y en tiempo real. Está embebido en la API de *DataStream* a través de la función *ProcessFunction*. Permite procesar eventos de uno o más flujos de datos con consistencia, tolerancia a fallos y estado.

Apache Hadoop

Apache Hadoop (<https://hadoop.apache.org/>) es un framework que permite el procesamiento y almacenamiento distribuido de grandes conjuntos de datos. Está inspirado en el sistema de ficheros distribuido de Google y en el paradigma MapReduce. Ofrece interfaz para el lenguaje de programación Java.

Apache Hadoop se divide en 4 módulos principales:

- **Hadoop Common:** contiene las bibliotecas comunes que usan el resto de módulos.
- **Hadoop Distributed File System (HDFS):** sistema de ficheros distribuido en el que se almacenan los datos a procesar.
- **Hadoop YARN:** sistema de gestión de recursos de Hadoop, que permite la gestión y monitorización de los recursos del clúster y, por tanto, la asignación de tareas.

- **Hadoop MapReduce**: sistema de procesamiento paralelo basado en el paradigma MapReduce.

Fue una de las primeras herramientas en aparecer en el ámbito del procesamiento distribuido de datos y cuenta con un extenso ecosistema que gira en torno a él, con proyectos como **Apache HBase** (<https://hbase.apache.org/>), **Apache Cassandra** (<https://cassandra.apache.org/>), **Apache Hive** (<https://hive.apache.org/>), **Apache Pig** (<https://pig.apache.org/>) y **Apache Zookeeper** (<https://zookeeper.apache.org/>).

Apache Samza

Apache Samza (<https://samza.apache.org/>) es un framework de procesamiento de flujos de datos en tiempo real de alto rendimiento. Tiene una sencilla integración con Apache Kafka y ofrece una interfaz para el lenguaje de programación Java.

Ofrece 3 API distintas, además de la posibilidad de ser un *runner* de Apache Beam:

- **Streams API**: ofrece operadores como *map*, *filter*, etc.
- **Task API**: de más bajo nivel de abstracción, permite una mayor flexibilidad para definir la lógica de procesamiento.
- **Samza SQL**: ofrece una interfaz declarativa SQL para realizar consultas sobre los datos, con operadores como *join* y *group by*.

Apache Spark

Apache Spark (<https://spark.apache.org/>) es un motor de computación en clúster diseñado para el procesamiento de datos a gran escala. Proporciona una interfaz de programación unificada que combina el procesamiento de datos en memoria y en disco, y tiene un modelo de datos que específicamente proporciona tolerancia a fallos (*Resilient Distributed Datasets*, *RDD*). Además, ofrece interfaces para los lenguajes de programación Java, Scala, Python y R.

Tiene varias bibliotecas integradas para diferentes tipos de tareas de procesamiento de datos:

- **Spark SQL**: permite utilizar consultas SQL sobre los datos procesados, e incluso ofrece conectividad JDBC/ODBC.
- **Spark Streaming**: permite realizar procesamiento de flujos de datos en tiempo real.

- **MLib**: permite implementar algoritmos de aprendizaje automático en clústeres distribuidos.
- **GraphX**: permite el procesamiento de grafos distribuido y en paralelo.

Apache Storm

Apache Storm (<https://storm.apache.org/>) es un sistema de computación en tiempo real distribuido, diseñado para procesar grandes cantidades de datos de manera rápida y confiable. Proporciona interfaces para los lenguajes de programación Java y Clojure. Además, soporta integraciones con varias tecnologías como Apache Kafka y Redis, entre otras.

El modelo de procesamiento de datos en Apache Storm se basa en los siguientes conceptos clave:

- **Topology**: define el flujo de datos y cómo se procesará.
- **Spout**: fuente de datos que emite tuplas de datos en el sistema.
- **Bolt**: componente que procesa el flujo de datos.
- **Nimbus**: controlador principal que gestiona las topologías, distribuye el código a los nodos del clúster y supervisa el estado de la ejecución.
- **Supervisor**: nodo que ejecuta los procesos de *spouts* y *bolts* asignados por *Nimbus*.

Tabla 2.2: Comparativa de características de Apache Beam, Apache Flink, Apache Hadoop, Apache Samza, Apache Spark y Apache Storm

Característica	Apache Beam	Apache Flink	Apache Hadoop	Apache Samza	Apache Spark	Apache Storm
Propósito Principal	Unificación de procesamiento batch y streaming	Procesamiento de datos en tiempo real y batch	Almacenamiento y procesamiento de datos masivos	Procesamiento de streaming distribuido	Procesamiento masivo de datos y análisis en tiempo real	Procesamiento de eventos en tiempo real
Modelo de Procesamiento	Batch y streaming	Batch y streaming	Batch	Streaming	Batch y streaming	Streaming
Compatibilidad de Datos	Estructurados y no estructurados	Estructurados y no estructurados	Estructurados y no estructurados	Estructurados y no estructurados	Estructurados y no estructurados	Estructurados y no estructurados
Escalabilidad	Alta	Alta	Alta	Alta	Alta	Alta
Procesamiento en Tiempo Real	Sí	Sí	No	Sí	Sí	Sí
Lenguaje de Programación Soportado	Java, Python, Go, TypeScript	Java, Scala, Python	Java	Java	Java, Scala, Python, R	Java, Clojure
Casos de Uso Típicos	ETL, procesamiento de eventos	ETL, análisis de datos en tiempo real	Almacenamiento y procesamiento de grandes volúmenes de datos	Procesamiento de flujos de datos en tiempo real	ETL, análisis de datos en tiempo real	Procesamiento de eventos y alertas en tiempo real
Integración con otras herramientas	Alta (BigQuery, Pub/Sub, etc.)	Alta (Kafka, HDFS, etc.)	Alta (Pig, Hive, etc.)	Alta (Kafka, HDFS, etc.)	Alta (Hadoop, HDFS, Kafka, etc.)	Alta (Kafka, HDFS, etc.)

2.2.3. Frameworks de análisis de datos espaciales

El análisis de datos espaciales es esencial en una amplia gama de aplicaciones, desde la planificación urbana y la gestión de recursos naturales hasta la logística y el análisis de patrones de movilidad. Estos datos, que incluyen información geográfica, mapas y coordenadas, requieren herramientas especializadas para su procesamiento y análisis debido a su naturaleza compleja y a menudo masiva.

La Tabla 2.3 proporciona una comparación exhaustiva de los frameworks detallados en la sección, destacando aspectos clave como el propósito principal, las capacidades de análisis espacial, la integración con otras herramientas y el soporte para diversos formatos de datos.

Apache Sedona

Apache Sedona (<https://sedona.apache.org/1.6.0/>) es una biblioteca de procesamiento de datos espaciales que extiende sistemas de procesamiento de datos como Apache Spark, Apache Flink y Snowflake (<https://www.snowflake.com/content/snowflake-site/global/es>) para permitir el análisis de datos geoespaciales a gran escala. Proporciona modelos de datos espaciales, operaciones espaciales y de indexación espacial y soporta también formatos de datos geoespaciales.

Apache Sedona proporciona interfaces para los lenguajes de programación Java, Scala, Python y R, además de incorporar el estándar SQL/MM Part 3 Spatial SQL [24], a través de lo que se denomina como Sedona SQL (<https://sedona.apache.org/1.6.0/api/sql/Overview/>).

Permite realizar operaciones geoespaciales como consultas de rango (Range Query), consultas de unión por rango (Range Join Query), consultas de unión por distancia (Distance Join Query) y consultas de los K vecinos más cercanos (K Nearest Neighbor Query). También manipula índices espaciales, como R-Tree o Quad-Tree, y soporta varios formatos de datos geoespaciales, como GeoJSON.

GeoMesa

GeoMesa (<https://www.geomesa.org/>) es un sistema de procesamiento y análisis de datos geoespaciales diseñado para manejar grandes volúmenes de datos espaciales y temporales en tiempo real. Proporciona capacidades avanzadas de almacenamiento, indexación y consulta de datos espaciales dentro del ecosistema de Apache Hadoop, aunque también se puede conectar con Apache Spark, Apache Kafka y Redis, entre otros, además de sistemas locales de ficheros.

Ofrece una biblioteca configurable mediante JSON para realizar la fase de ingesta de datos, llamada GeoMesa Convert (<https://www.geomesa.org/documentation/stable/user/convert/index.html>). Además, también se puede utilizar GeoMesa para procesar datos con la arquitectura Map-Reduce, principalmente utilizando su conexión con Apache Hadoop.

Simba

Simba (<https://initialdlab.github.io/Simba/>) (*Spatial In-Memory Big data Analytics*) es un motor distribuido de análisis espacial en memoria sobre los datos procesados por Apache Spark. Amplía el motor de Apache Spark, Spark SQL para soportar consultas espaciales y análisis complejos.

Además, integra soporte de indexación nativa sobre RDDs para desarrollar operadores espaciales eficientes y extiende el optimizador de consultas de Spark SQL con optimizaciones lógicas y físicas primando el coste temporal y la memoria de los distintos planes disponibles.

LocationSpark

El proyecto LocationSpark (<https://github.com/purduedb/LocationSpark>) es un sistema de gestión de datos distribuidos en memoria, diseñado para manejar grandes volúmenes de datos espaciales. Está construido sobre Apache Spark y ofrece operadores espaciales. Utiliza índices espaciales en memoria para mejorar el rendimiento y también reduce la sobrecarga de comunicación en red.

Entre los operadores espaciales, permite realizar actualizaciones de RDD de Apache Spark (Update RDD), consultas de rango (Range Query), consultas de los K vecinos más cercanos (KNN Query), consultas espaciales y textuales (Spatial Textual Query), uniones espaciales (Spatial Join) y uniones de los K vecinos más cercanos (kNN Join).

Tabla 2.3: Comparativa de características de Apache Sedona, GeoMesa, Simba y LocationSpark

Característica	Apache Sedona	GeoMesa	Simba	LocationSpark
Propósito Principal	Análisis y procesamiento espacial	Almacenamiento y procesamiento espacial	Consulta espacial en tiempo real	Procesamiento espacial distribuido
Modelo de Procesamiento	Batch	Batch y streaming	Batch y streaming	Batch y streaming
Compatibilidad de Datos	Espaciales	Espaciales	Espaciales	Espaciales
Escalabilidad	Alta	Alta	Alta	Alta
Procesamiento en Tiempo Real	No	Sí	Sí	Sí
Lenguaje de Programación Soportado	Java, Python, Scala	Java	Java, SQL	Java, SQL
Casos de Uso Típicos	Análisis espacial, geoprocesamiento	Análisis espacial, geoprocesamiento	Análisis espacial, consultas en tiempo real	Análisis espacial, geoprocesamiento
Integración con otras herramientas	Alta /Spark, Hadoop, etc.)	Alta (HBase, Accumulo, etc.)	Alta (Spark, HDFS, etc.)	Alta (HDFS, Spark, etc.)

2.2.4. Trabajos académicos previos

A la hora de realizar este proyecto se han buscado referencias de trabajos realizados anteriormente en un ámbito similar, y los resultados que se han considerado más relevantes y que tienen una aplicación directa en el trabajo realizado se citan y detallan en esta sección.

Linear Road: A Stream Data Management Benchmark

En el artículo [25] se propuso un benchmark diseñado para evaluar sistemas de procesamiento de datos en tiempo real, centrándose en un caso de uso específico: un sistema inteligente de tráfico (ITS) que requiere el procesamiento continuo y en tiempo real de los datos de los peajes de una autopista, variando el coste de cada peaje de manera dinámica en función de la congestión de tráfico y la proximidad a accidentes.

Partiendo de eventos de posición periódicos de sensores de vehículos, se calcula en tiempo real el coste del peaje (si el vehículo ha pasado un peaje), se detectan accidentes (si uno o más vehículos siguen en la misma posición durante 4 eventos) y se emiten alertas para los vehículos afectados por los accidentes.

Se puede consultar más información en la web <https://www.cs.brandeis.edu/~linearroad/>, así como diversas herramientas para generar datos siguiendo el formato propuesto y validando los resultados de las consultas.

Uno de los casos de aplicación de este trabajo fin de grado está basado en este benchmark, dado que plantea retos tecnológicos que hoy en día siguen siendo complejos de resolver, y es todo un referente a la hora de plantear un conjunto de consultas de prueba para flujos de datos continuos.

A query processor for prediction-based monitoring of data streams

En el artículo [26] se propone un procesador de consultas para monitorizar valores en redes de sensores mediante funciones de predicción. De esta forma, los sensores solamente comunican sus valores si estos se desvían de las predicciones, reduciendo la frecuencia de transmisiones.

El procesador de consultas (sobre flujos de datos) desarrollado en el artículo ofrece dos tipos de restricciones: por valor y por ventana temporal. Además, se incorpora una estrategia de gestión de los *buffers*, que prioriza la precisión de las funciones de predicción de las tuplas.

Este procesador de consultas se va a utilizar como procesador de flujos de datos, y en el Capítulo 4 se evaluará contra otros tres sistemas de procesamiento ampliamente

utilizados.

Proyecto Fin de Carrera: Sistema de procesamiento de data streams

En el Proyecto Fin de Carrera [27] se desarrolló una herramienta de simulación de casos de uso de redes de sensores (sistemas GPS y cotizaciones en bolsa), además de permitir realizar simulaciones genéricas y sobre redes de sensores genéricos.

Por otro lado, en la misma herramienta se añadió una interfaz gráfica de usuario al procesador de consultas basado en funciones de predicción referido en la Subsección 2.2.4, además de incluir algunas operaciones adicionales, como la posibilidad de crear alarmas para gestionar las respuestas de las consultas realizadas.

De esta forma, este Proyecto Fin de Carrera ha servido como punto de partida para la integración del procesador de consultas basado en funciones de predicción en la herramienta desarrollada. También ha servido de referencia para el inicio del prototipado de la interfaz realizada.

SimulParking: A Simulator of Parking Lots to Evaluate Data Management Techniques

En el artículo [28] se describe SimulParking, una herramienta de simulación de datos para evaluar técnicas de gestión de datos para realizar sugerencias de plazas de aparcamiento en garajes públicos.

La herramienta contiene además un editor que permite definir la estructura de las plazas de aparcamiento y diseñar escenarios con varias plantas. Se puede consultar más información en la web <https://webdiis.unizar.es/~silarri/prot/SimulParking/>.

Uno de los casos de aplicación de este trabajo fin de grado está basado en esta herramienta, por lo que ha sido utilizada para generar datos de prueba a lo largo del desarrollo del trabajo.

MAVSIM+: Testing Data Management Techniques for Vehicular Networks and Parking Space Searching

En el artículo [29] se describe MAVSIM+, una herramienta de simulación de datos para evaluar técnicas de gestión de datos en búsqueda de aparcamiento, tanto plazas de aparcamiento en las calles como aparcamientos, gracias a su integración con SimulParking, la herramienta de la Subsección 2.2.4.

MAVSIM+ originalmente era un simulador diseñado para evaluar técnicas de gestión de datos en el contexto de redes de vehículos basada en agentes móviles, a la que

después se añadió la extensión sobre plazas de aparcamiento. Se puede consultar más información en la web <https://webdiis.unizar.es/~silarri/prot/MAVSIMPlus/>.

Otro de los casos de aplicación de este trabajo está basado en esta aplicación, por lo que también ha sido utilizada para generar datos de prueba a lo largo del desarrollo del trabajo.

Capítulo 3

Planteamiento y desarrollo del sistema

En este capítulo se van a comentar las decisiones de diseño más destacadas, como la selección de las tecnologías de manipulación y procesamiento de flujos de datos (Sección 3.1), la arquitectura del sistema (Sección 3.2), las tecnologías utilizadas en su implementación (Sección 3.3), el entorno utilizado (Sección 3.4) y el desarrollo de la herramienta (Sección 3.5).

3.1. Selección de las tecnologías de manipulación y procesamiento de flujos de datos a utilizar

En esta sección se van a comentar primero las tecnologías que han sido descartadas, junto con las razones que han llevado a esa decisión. Después, se especificarán las tecnologías que serán utilizadas, con el fin de facilitar una visión global del sistema.

3.1.1. Tecnologías descartadas

Los frameworks de procesamiento de datos distribuido que no se van a utilizar en el proyecto son los siguientes:

- Apache Hadoop, puesto que tampoco tiene soporte nativo para flujos continuos de datos.
- Apache NiFi, porque no tiene soporte para flujos continuos de datos.
- Apache Samza, debido a que no posee garantías de conectividad y solo dispone de integración con Apache Kafka.
- Apache Storm, dado que es similar al resultado que queremos alcanzar con este proyecto.

Además, los frameworks de análisis de datos espaciales que se han descartado en el proyecto han sido los siguientes:

- GeoMesa, debido a que necesita añadir un almacén de datos adicional respecto a las demás herramientas.
- LocationSpark, puesto que lleva 8 años discontinuado.
- Simba, debido a que la versión máxima de Spark que soporta está obsoleta, lleva 7 años discontinuado y solamente ofrece código fuente en Scala.

Aunque se hayan descartado, muchas de estas tecnologías son muy populares y podría considerarse su implantación en muchos proyectos de gestión de datos.

3.1.2. Tecnologías utilizadas

De forma complementaria al apartado anterior, se ha decidido realizar este proyecto utilizando las tecnologías que no se han descartado, todas ellas de código abierto y ampliamente utilizadas.

En cuanto a sistemas de mensajería y streaming de datos, se utilizan los siguientes:

- Apache Kafka, debido a que es el estándar de facto en los sistemas de streaming de datos.
- Apache Pulsar, debido a que es un sistema muy utilizado pero aporta bastantes diferencias respecto a Apache Kafka.
- Redis Streams, debido a que Redis es una base de datos muy utilizada y puede resultar muy interesante en arquitecturas software ya desplegadas explotar esta funcionalidad en lugar de añadir un sistema adicional.

Como frameworks de procesamiento de datos distribuido se usan:

- Apache Beam, debido a que es una opción muy utilizada para gestionar despliegues en cloud de procesamiento de datos distribuidos.
- Apache Flink, debido a que es el estándar de facto en el procesamiento de datos distribuido.
- Apache Spark: debido a que durante mucho tiempo fue el sistema de procesamiento de datos distribuido más utilizado.

En cuanto a los frameworks de análisis de datos espaciales, se ha preferido usar Apache Sedona, puesto que ofrece integraciones con Apache Spark y Apache Flink.

3.2. Arquitectura propuesta

Para implementar el proyecto se ha decidido utilizar el modelo Cliente-Servidor con arquitectura multicapa y modelo-vista-controlador, dividiendo las responsabilidades entre una primera capa de acceso a los flujos de datos y lógica de negocio, y una segunda capa de interfaz gráfica de usuario.

El componente back-end está organizado en una arquitectura orientada a servicios y ofrece una interfaz API RESTful, basada en el protocolo HTTP y sin estado de sesión. Además, cuenta con una base de datos PostgreSQL para el almacenamiento persistente de información del sistema, lo que facilita el registro y la administración efectiva de forma eficiente de las entidades en la capa de modelos.

El componente front-end explota la API RESTful ofrecida por el componente back-end y muestra la información al usuario a través de una GUI de escritorio.

Todo esto se resume de forma visual en la Figura 3.1.

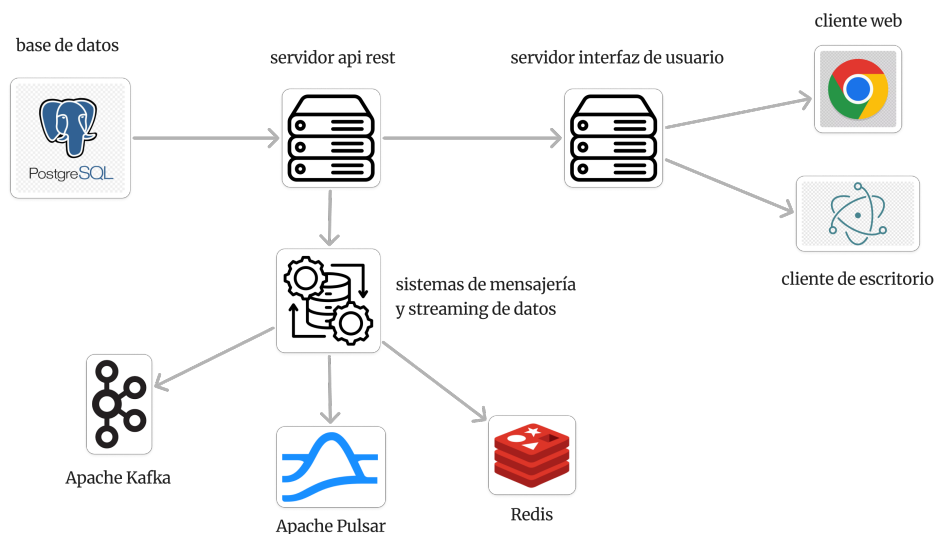


Figura 3.1: Diagrama de la arquitectura del sistema

3.3. Tecnologías utilizadas en la implementación de la arquitectura

Para implementar esta arquitectura, se han tomado diversas decisiones respecto a las tecnologías a emplear. Se van a abordar las decisiones de cada uno de los componentes por separado.

3.3.1. Back-end

El componente back-end se ha programado en lenguaje **Java** (versión 17), utilizando el framework para crear servicios web **Spring Boot**, con el servidor web Tomcat embebido en la aplicación desarrollada. Su código se ha organizado en diferentes capas, como modelos, repositorios, servicios y controladores.

Se ha seleccionado Java principalmente por su gran variedad de bibliotecas y frameworks e integraciones (de hecho, es el único lenguaje de programación común a todos los sistemas seleccionados en la Sección 3.1 junto a Python), su gestión automática de memoria y su orientación a objetos. Además, también se tuvo en cuenta la experiencia previa en el uso este lenguaje (gracias a las asignaturas Tecnología de programación, Ingeniería del Software, Inteligencia artificial, Sistemas de información, Sistemas de ayuda a la toma de decisiones y Sistemas de información distribuidos).

Tras la elección de Java, se decidió utilizar Spring Boot por su facilidad de configuración, su facilidad para implementar aplicaciones RESTful y la posibilidad de mantener todo lo necesario para ejecutar el componente en un único fichero ejecutable.

3.3.2. Front-end

Por otra parte, el componente front-end se ha programado en lenguaje **JavaScript**, con la biblioteca **React JS** y su framework **Next JS**, añadiendo la biblioteca **Material UI** con componentes de diseño sencillos de utilizar y el framework **Tailwind CSS** para personalizar la apariencia final de los componentes. Por encima se ha añadido el framework **Electron** para dar una interfaz de escritorio multiplataforma a la aplicación web, aunque sigue siendo posible acceder al mismo contenido a través de un navegador web.

Se ha seleccionado JavaScript principalmente porque ya se tenía experiencia previa con este lenguaje (gracias a la asignatura Sistemas y tecnologías web) y por su gran variedad de bibliotecas y frameworks.

Una vez elegido JavaScript, se seleccionó React por su facilidad de uso y también por la experiencia previa que se tenía, pero con el framework Next JS y tecnologías

adicionales de diseño que no se habían utilizado antes, pero son muy comunes y tienen grandes comunidades online.

Por último, se ha decidido utilizar Electron debido a que embebe Chromium y Node.js para permitir la ejecución de la aplicación web, y se obtiene una apariencia y comportamiento uniforme en todas las plataformas al mantener el motor de renderizado.

3.4. Entorno utilizado

A lo largo del desarrollo del proyecto, se han utilizado las siguientes herramientas para facilitar el desarrollo del código fuente del sistema:

- Gradle, para automatizar la compilación, empaquetado y ejecución de la aplicación java que constituye el back-end.
- Git y GitHub, para realizar control de versiones del código fuente del proyecto.
- Visual Studio Code, IntelliJ IDEA y WebStorm como IDEs.
- DataGrip y DBeaver para acceder a los contenidos de las bases de datos PostgreSQL y Redis de manera externa en las fases tempranas de la implementación del sistema.
- Docker para virtualizar los servidores de los sistemas externos utilizados.

El entorno en el que se ha comprobado el funcionamiento del sistema ha sido el siguiente:

- Un portátil con sistema operativo Ubuntu 22.04.5 LTS, con CPU Intel Core i5-7200U de 2 núcleos y 4 hilos, y 16 GB de memoria RAM, donde se han ejecutado los dos componentes de la aplicación, back-end y front-end, y se ha accedido a la interfaz de usuario.
- Un ordenador de sobremesa con sistema operativo Windows 11, con CPU Intel Core i5-10400 de 6 núcleos y 12 hilos, y 16 GB de memoria RAM, donde se han ejecutado los servidores de los sistemas externos utilizados: PostgreSQL, Apache Kafka, Apache Pulsar y Redis, virtualizados mediante contenedores Docker para facilitar su administración.
- Una red WLAN conectando ambos computadores.

Cabe destacar que se ha utilizado este entorno debido a la imposibilidad material de desplegar todos los sistemas de forma simultánea en el computador portátil, dado que añadir una capa de red siempre aporta complejidad adicional al sistema y aumenta la latencia experimentada por el usuario.

3.5. Desarrollo de la herramienta

En esta sección se van a ver los detalles que se consideran más relevantes del desarrollo de la herramienta.

3.5.1. Back-end

Para acceder a información sobre elementos auxiliares de la aplicación y realizar operaciones CRUD sobre ellos, como las configuraciones de acceso a los diferentes sistemas de mensajería y streaming de datos, las simulaciones importadas o los tipos de consulta definidos, se ha utilizado la estructura de componentes de Spring Boot (entidad, repositorio, servicio y controlador) apoyado por JPA sobre una base de datos PostgreSQL. El esquema relacional de estas entidades se muestra en la Figura 3.2.

De esta forma, por cada entidad de la Figura 3.2 hay 4 clases Java: entidad, repositorio, servicio y controlador; que estandarizan y simplifican el acceso a la información almacenada en la base de datos, quedando una estructura similar a la que se muestra en la Figura 3.3.

Sobre esta base se ha desarrollado el resto del proyecto, del que hay que destacar las decisiones de modelado de los aspectos más complejos.

En primer lugar, hay que comentar el modelado del comportamiento de las clases que utilizan los sistemas de mensajería y streaming con una interfaz común, como se muestra en la Figura 3.4, para simplificar su gestión desde clases externas.

La misma idea se ha utilizado a la hora de modelar el formato de los mensajes y eventos generados en dichos sistemas, como se puede observar en la Figura 3.5, cada uno de ellos correspondiente a cada escenario de aplicación del proyecto, detallados en la Sección 4.1.

También se ha utilizado el mecanismo de herencia entre clases para modelar las relaciones entre los diferentes tipos de simulación (de nuevo, cada uno de ellos correspondiente a cada escenario de aplicación considerado en el proyecto), como se puede observar en la Figura 3.6. El código correspondiente se puede ver en el Listing B.1, destacando el uso de las posibilidades de sincronización de Java para pausar y volver a reproducir una simulación.

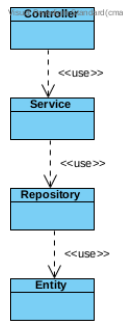


Figura 3.3: Diagrama de ejemplo de las relaciones entre controladores, servicios, repositorios y entidades de Spring Boot

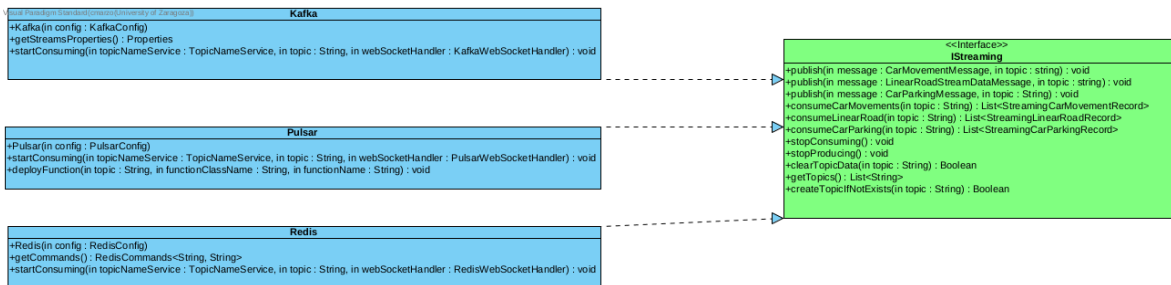


Figura 3.4: Diagrama de las clases de acceso a los sistemas de mensajería y streaming

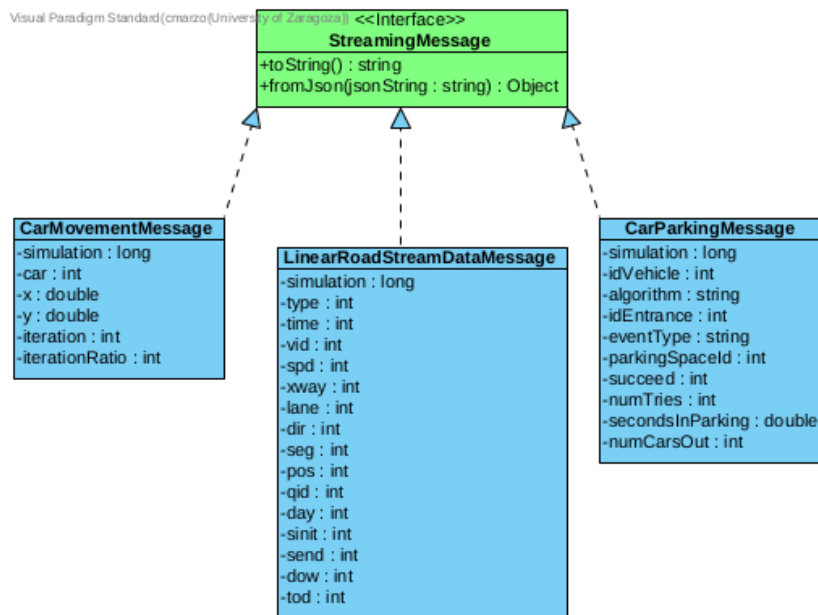


Figura 3.5: Diagrama de las clases que modelan los mensajes y eventos generados en la aplicación

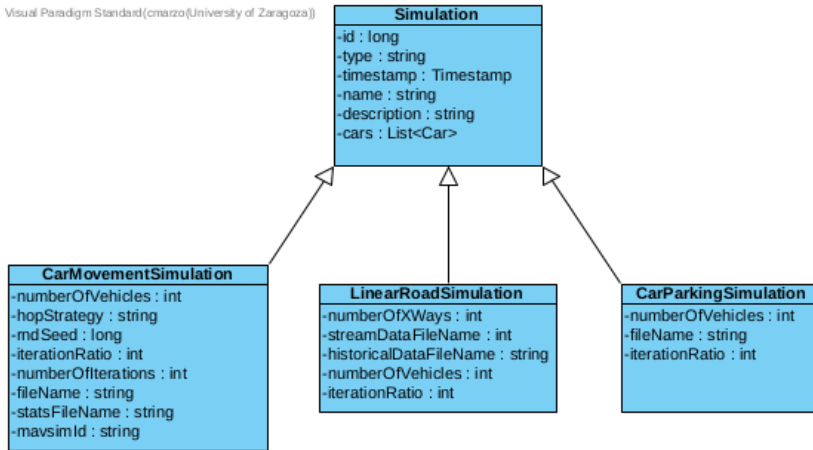


Figura 3.6: Diagrama de clases de simulaciones

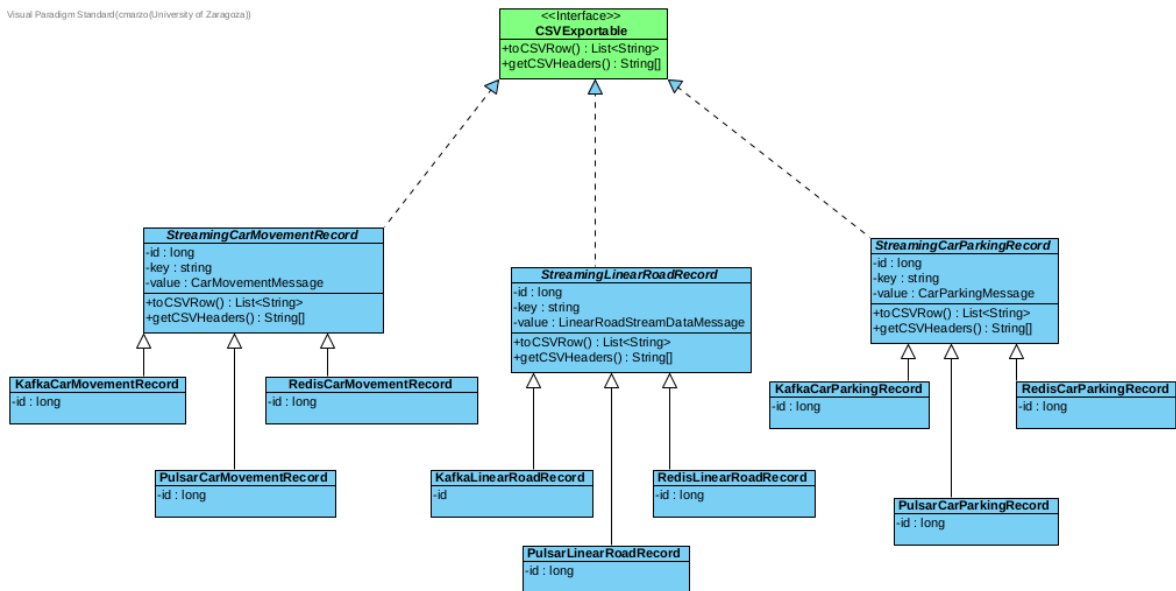


Figura 3.7: Diagrama de clases

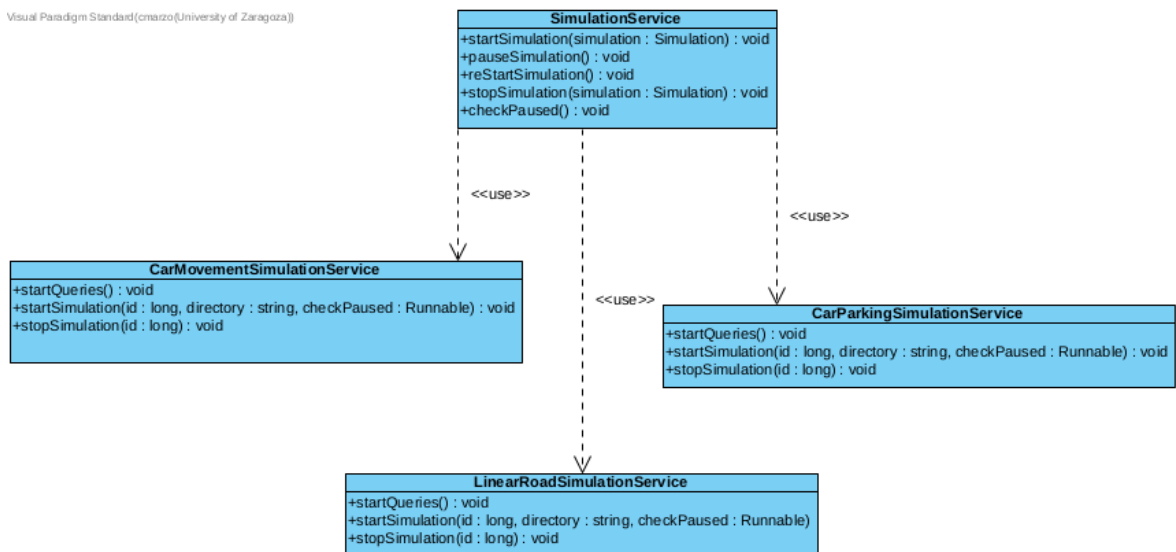


Figura 3.8: Diagrama de clases

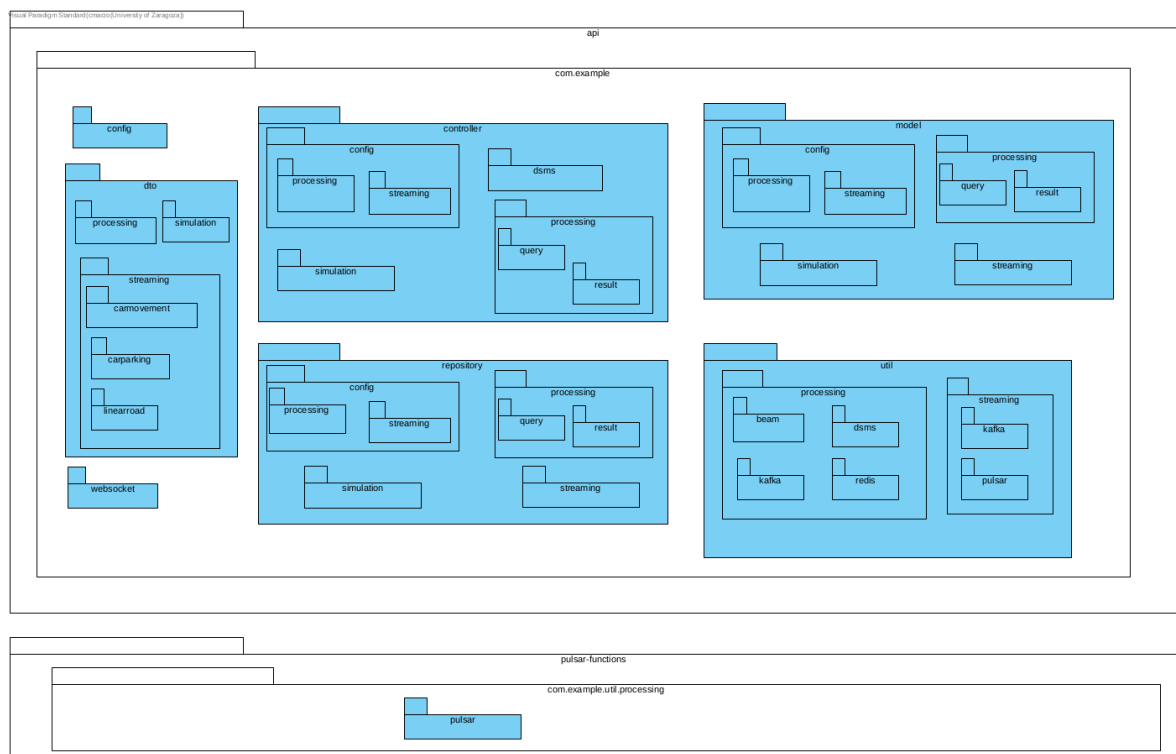


Figura 3.9: Diagrama de paquetes del componente back-end

3.5.2. Front-end

En el componente front-end, al utilizar el framework Next JS, la organización de los ficheros de código fuente depende el modo de organización de las rutas escogido (Pages Router o App Router).

Para poder crear una aplicación de escritorio a partir de la aplicación web creada con Next JS, es necesario generar una aplicación estática mediante procesos de SSG (Static Site Generation). Aunque App Router es compatible con SSG da muchos problemas al integrarlo con Electron y se ha utilizado Pages Router.

Por tanto, el directorio principal del componente front-end se llama **pages**, y en él se sitúan todos los ficheros que tienen correspondencia con los recursos de la aplicación accesibles mediante URIs. Además, los componentes React utilizados para dotar de contenidos a las vistas se encuentran en el directorio **components**, e imágenes e iconos se encuentran en el directorio **public**.

A lo largo de todas las vistas se repiten el mismo menú y el mismo panel lateral, encapsulados en dos componentes concretos que se utilizan desde todas ellas, para dar uniformidad a la interfaz de usuario.

Además, el panel lateral se puede encoger y expandir, y en esa variación de tamaño le acompaña el resto del contenido de la vista, gracias a una personalización CSS que se observa ver en el Listing B.2.

Por último, cabe decir que, aunque el grueso de la comunicación entre los dos componentes se realiza mediante la API REST ofrecida por el componente back-end, toda la transmisión de datos en tiempo real (bastante frecuente en este sistema en particular) se realiza mediante WebSocket.

Capítulo 4

Evaluación de técnicas de procesamiento de flujos de datos

En este capítulo se va a abordar el contenido principal de este Trabajo Fin de Grado, la evaluación de varias técnicas de procesamiento de flujos de datos en ciudades inteligentes.

Para ello, se han planteado tres escenarios de aplicación, explicados en la Sección 4.1, para comparar las técnicas detalladas en la Sección 4.2, mediante las consultas enumeradas en la Sección 4.3, con la presentación de resultados cualitativos en la Sección 4.4 y cuantitativos en la Sección 4.5.

4.1. Escenarios de aplicación considerados

Para llevar a cabo esta evaluación, se han considerado tres escenarios:

- La simulación de la movilidad de vehículos en una ciudad, dada por la comunicación de su posición en coordenadas geográficas GPS en tiempo real. Los datos de este escenario se han generado con la herramienta MAVSIM+, explicada en la Subsección 2.2.4, además de incluir el fichero jar del proyecto como librería para poder leer los ficheros de datos.
- La simulación de la gestión de los peajes en una autopista, dado por la comunicación de la posición de los vehículos respecto a segmentos del trayecto en tiempo real. Los datos de este escenario se han generado con el código disponible en <https://github.com/walmartlabs/LinearGenerator>, pese a que en la web del benchmark Linear Road, explicado en la Subsección 2.2.4 hay disponibles generadores de datos, dado que al intentar utilizarlos hubo varios problemas técnicos, que se detallan en la Subsección 5.1.

- La simulación de la gestión de un aparcamiento, dado por eventos de entrada y salida del aparcamiento, de reserva de plazas de aparcamiento, y de éxito o fallo al intentar aparcar (fallo si la plaza ya estaba ocupada). Los datos de este escenario se han generado con la herramienta SimulParking, explicada en la Subsección 2.2.4.

De este modo, se dispone de tres escenarios heterogéneos de prueba para comparar los sistemas y las técnicas utilizadas.

4.2. Técnicas evaluadas

Para comparar los tres sistemas de mensajería y streaming seleccionados, Apache Kafka, Apache Pulsar y Redis Streams; se van a utilizar los soportes nativos que ofrecen cada uno de ellos, es decir, Kafka Streams, Pulsar functions y Redis functions, respectivamente.

Además, también se va a utilizar el procesador de consultas basado en funciones de predicción explicado en la Subsección 2.2.4, utilizándolo directamente como librería, dado que se ha podido tener acceso a su código fuente. Además, está escrito en Java y ofrece varias clases de implementación de pruebas en las que se puede ver cómo hay que usarlo.

Aunque inicialmente se consideraron consultas de snapshot programadas con Apache Beam y ejecutadas en Apache Spark y Apache Flink, no se van a utilizar para comparar los sistemas porque no ofrecen soporte para flujos de datos continuos, y además al ejecutarlas con la misma infraestructura se añade la misma capa de procesamiento por encima. Estas consultas de snapshot se han mantenido en el proyecto por completitud, pero no se van a evaluar.

Lo mismo sucede con el framework de análisis de datos espacial que se había planteado en un principio, Apache Sedona, como añadido sobre Apache Spark y Apache Flink. Se ha mantenido el apartado correspondiente en la Sección 2.2 por completitud, ya que es un esfuerzo que se ha realizado en este proyecto, pero no se va a utilizar para responder consultas.

4.3. Consultas implementadas

Para comparar los cuatro sistemas se han propuesto varias consultas en cada escenario, que se van a ir explicando a lo largo de esta sección.

4.3.1. Simulación de movilidad de vehículos en una ciudad

En este escenario, la información del flujo de datos consiste en las coordenadas geográficas de latitud y longitud de la ubicación del vehículo, el identificador del vehículo, la iteración en la que se ha generado la información (a modo de reloj), y la velocidad de reproducción (para poder reproducir la misma simulación a diferentes velocidades, al doble, a la mitad, etc., modificando el tiempo que transcurre entre cada iteración).

Con esta información se han planteado las siguientes consultas:

1. Obtener la posición de cada coche en cada momento de la simulación, análogo a `SELECT *` en SQL.
2. Obtener los vehículos cuya velocidad esté por encima de un determinado valor, análogo a `SELECT (distance)` en SQL.
3. Obtener la distancia recorrida hasta ese momento por cada vehículo, análogo a `SELECT sum(distance)` en SQL.
4. Obtener la velocidad media por km^2 , es decir, una consulta de agregación con consideraciones espaciales.
5. Obtener los vehículos que se encuentran a una distancia inferior a x metros en un intervalo de tiempo de y segundos, es decir, un join de ventana temporal.
6. Obtener las zonas más transitadas (hotspots).

Se ha procurado que el conjunto de consultas sea heterogéneo, abarcando desde operaciones muy básicas, pasando por agregaciones, sumas y medias, joins temporales y consultas avanzadas de análisis espacial.

4.3.2. Simulación de gestión de peajes en una autopista

En este escenario, la información del flujo de datos consiste en el segundo de la simulación (que dura tres horas) en el que se ha generado la información, el identificador del vehículo, la velocidad en millas por hora de dicho vehículo, y los identificadores de la autopista, carril, dirección, segmento y posición en los que se encuentra el vehículo, que conforman su ubicación.

En el benchmark original se proponen dos consultas continuas que se ha decidido implementar en el sistema:

1. Obtener el precio del peaje que un vehículo ha cruzado, en función de en la velocidad promedio de la autopista en esa zona, el número de vehículos que se encuentran en ese momento en esa zona de la autopista, y si hay algún accidente cercano.
2. Generar una alerta por cada vehículo que haya tenido un accidente, en función de la ausencia de variación en su posición.

Ambas consultas requieren un procesamiento complejo y el uso de ventanas temporales.

4.3.3. Simulación de gestión de un aparcamiento

En este escenario la información del flujo de datos es diferente a los de los otros dos escenarios, más similar a la traza del intercambio de información entre microservicios, caracterizado por la generación de diferentes tipos de evento con semánticas determinadas.

Los eventos pueden ser de tipo «ENTER», «BOOK», «PARK_SUCCESS», «VACATE», «FAIL_LEAVE», «PARK_FAIL» o «END_ENTRY». Con esta información se han propuesto dos consultas:

1. Obtener el tiempo que han tardado en aparcar los vehículos que han conseguido estacionarse con éxito.
2. Obtener el ratio de éxito en el aparcamiento ($\frac{\text{casos de éxito}}{\text{casos totales}}$)

Ambas consultas suponen el procesamiento del flujo de datos para ir actualizando las estadísticas de aparcamiento.

4.4. Evaluación cualitativa

Ante el reto de implementar las consultas definidas en la sección anterior (Sección 4.3), la primera cuestión que hay que plantearse es si los sistemas seleccionados ofrecen soluciones cómodas para poder realizar las consultas planteadas, objetivo de la Subsección 4.4.1. El siguiente paso es implementar las consultas con los medios ofrecidos por cada sistema. Los retos surgidos al hacer esto se detallan en la Subsección 4.4.2.

4.4.1. Programación de las consultas

Kafka Streams ofrece un DSL con abstracciones sobre los flujos de datos como `KStream`, o sobre tablas como `KTable`. Sobre estos elementos se ofrece una interfaz de programación declarativo con transformaciones sin estado, como *map* o *filter*, y transformaciones con estado como agregaciones, joins y ventanas. Sin duda es el sistema más completo y permite implementar todos los tipos de consultas planteados en este trabajo. Su documentación está disponible en <https://kafka.apache.org/documentation/streams/>.

Pulsar functions ofrece un SDK con una interfaz *Function* que especifica un método *process* para crear una Pulsar function a partir de una clase Java. En Java es posible utilizar otros dos métodos *initialize* y *close* para implementar lógica adicional al inicio y al final de la ejecución de la función, pero no se da soporte explícito a ninguna operación, por lo que hay que implementar toda la lógica de los operadores correspondientes mediante bibliotecas externas y estructuras de datos auxiliares internas. Su documentación está disponible en <https://pulsar.apache.org/docs/4.0.x/functions-overview/>.

En el caso de Redis functions, se ofrecen las funciones básicas del lenguaje Lua y una biblioteca redis para poder hacer llamadas de escritura/lectura en Redis, es decir, sobre Streams, Hash, etc. No hay ningún tipo de soporte explícito para ninguna operación de manipulación de datos, por lo que hay que implementar toda la lógica de los operadores correspondientes mediante funciones internas y utilizando el propio almacenamiento de Redis como estructura de datos auxiliar. Su documentación está disponible en <https://redis.io/docs/latest/develop/interact/programmability/functions-intro/>.

En el procesador de consultas basado en funciones de predicción se pueden utilizar dos tipos de restricciones al crear una consulta, una restricción de valor, y una restricción de join de ventana temporal. Al contrario que con Redis functions y Pulsar functions, no es posible implementar los operadores correspondientes basándose en él.

La Tabla 4.1 condensa la información redactada en este apartado, junto a información adicional sobre su facilidad de uso y su última fecha de actualización.

Características	Kafka Streams	Pulsar functions	Redis functions	Funciones de predicción
Lenguaje de programación	Java, Scala	Java, Python, Go	Lua	Java
Integración como biblioteca en un proyecto software	✓	✗	✗	✓
Ejecución en un servidor externo	✗	✓	✓	✗
Operador de agregación	✓	✗	✗	✗
Operador de join	✓	✗	✗	✗
Operador de ventana temporal	✓	✗	✗	✓
Operador de restricción de valor	✓	✗	✗	✓
Documentación disponible	✓	✓	✓	✓
Última fecha de actualización	Noviembre de 2024	Octubre de 2024	Octubre de 2024	Mayo de 2008
Facilidad de uso	Alto	Medio	Bajo	Bajo

Tabla 4.1: Evaluación cualitativa de los 4 sistemas de procesamiento de flujos de datos a utilizar

4.4.2. Implementación de las consultas

Dado que el procesador de consultas basado en funciones de predicción ofrece restricciones de valor y de join de ventana temporal, se va a utilizar en las consultas primera, segunda y quinta del primer escenario de aplicación, que son las que se ajustan a este tipo de operadores.

En la segunda consulta del primer escenario, *obtener los vehículos cuya velocidad esté por encima de un determinado valor*, es necesario calcular la velocidad de cada vehículo mediante la distancia entre su ubicación actual y su ubicación anterior, dado que el vehículo no informa de su velocidad.

Para realizar este cálculo, en Kafka Streams y Pulsar functions se han podido utilizar bibliotecas geográficas y geométricas específicas, *JTS* y *Proj4J*, dado que el procesamiento se programa en código Java. En cambio, en Redis functions esto no ha sido posible, por lo que el cálculo de la distancia se ha hecho de forma aproximada mediante la fórmula de *Haversine*¹, puesto que la ubicación de los vehículos viene dada por su latitud y longitud. Estas formas de calcular las distancias se siguen utilizando en el resto de consultas de este escenario, dado que todas ellas requieren el cálculo de distancias.

Aún así, al añadir estas bibliotecas al proyecto, el fichero .jar en el que se empaqueta todo el proyecto, incluyendo de forma fundamental las clases que implementan las Pulsar functions se ha quedado demasiado pesado, y su carga en Apache Pulsar resultó imposible. Para solucionar esto, se ha tenido que crear un módulo java específico para incluir únicamente las dependencias imprescindibles para que tuviera un tamaño asumible por la API de Apache Pulsar.

La cuarta consulta de este primer escenario, *Obtener la velocidad media por km²*, se ha implementado en Kafka Streams mediante el operador *aggregate*, y en Pulsar functions y Redis functions mediante el uso de estructuras de datos auxiliares para almacenar los datos intermedios de agregación.

De forma similar, la quinta consulta, un join de ventana temporal, se podría implementar también en Pulsar functions y Redis functions pese a no haber soporte específico para ello. No obstante, tras evaluar su complejidad, se ha decidido que supone demasiado esfuerzo respecto a los beneficios que puede aportar al trabajo. De esta forma, esta consulta se ha implementado solamente en Kafka Streams y el procesador de consultas basado en funciones de predicción.

Con la misma lógica, la sexta consulta solamente se ha implementado en Kafka Streams.

Respecto al segundo escenario, sucede lo mismo. Ambas consultas son bastante complejas y requieren agregaciones y el uso de ventanas temporales, por lo que solamente se han implementado en Kafka Streams.

La información expuesta en este apartado está resumida en la Tabla 4.2 para mayor claridad. El único sistema en el que ha podido implementar todas las consultas es Kafka Streams.

¹ $d = 2 \cdot R \cdot \arcsin \left(\sqrt{\sin^2 \left(\frac{\text{lat2} - \text{lat1}}{2} \right) + \cos(\text{lat1}) \cdot \cos(\text{lat2}) \cdot \sin^2 \left(\frac{\text{lon2} - \text{lon1}}{2} \right)} \right)$

Consultas	Kafka Streams	Pulsar functions	Redis functions	Funciones de predicción
SELECT *	✓	✓	✓	✓
SELECT (distance)	✓	✓	✓	✓
SELECT sum(distance)	✓	✓	✓	✗
avg(distance) por <i>km</i> ²	✓	✓	✓	✗
join de ventana temporal	✓	✗	✗	✓
hotspots	✓	✗	✗	✗
Coste (\$) de los peajes	✓	✗	✗	✗
Detección de accidentes	✓	✗	✗	✗
Coste (s) de aparcamiento	✓	✓	✓	✗
Ratio de éxito de aparcamiento	✓	✓	✓	✗

Tabla 4.2: Resumen de la viabilidad de la implementación de las consultas propuestas en Apache Kafka, Apache Pulsar, Redis y el procesador de consultas basado en funciones de predicción

4.5. Evaluación cuantitativa

Una vez implementadas todas las consultas, en esta sección se van a analizar los resultados de rendimiento que ofrecen los sistemas a comparar. En el Apartado 4.5.1 se va a analizar el impacto de la predictibilidad de la trayectoria de los datos en el ahorro de comunicaciones con el procesador de consultas con funciones de predicción, y en el Apartado 4.5.2 se va a analizar el rendimiento de los todos sistemas a la hora de ejecutar las consultas.

4.5.1. Procesador de consultas con funciones de predicción

La ventaja principal que tiene este procesador de consultas es que al implementar funciones de predicción, consigue evitar las comunicaciones que estén dentro del rango de valores predichos por la función. Se ha comprobado esto experimentalmente, con varias pruebas diferentes.

En primer lugar, se simularon trayectorias completamente constantes durante 10 iteraciones, muy sencillas de predecir, con distintas cantidades de vehículos en la simulación. La evolución del número de actualizaciones se puede ver en la Figura 4.1, y la evolución de la ratio de actualizaciones por segundo se puede observar en la Figura 4.2.

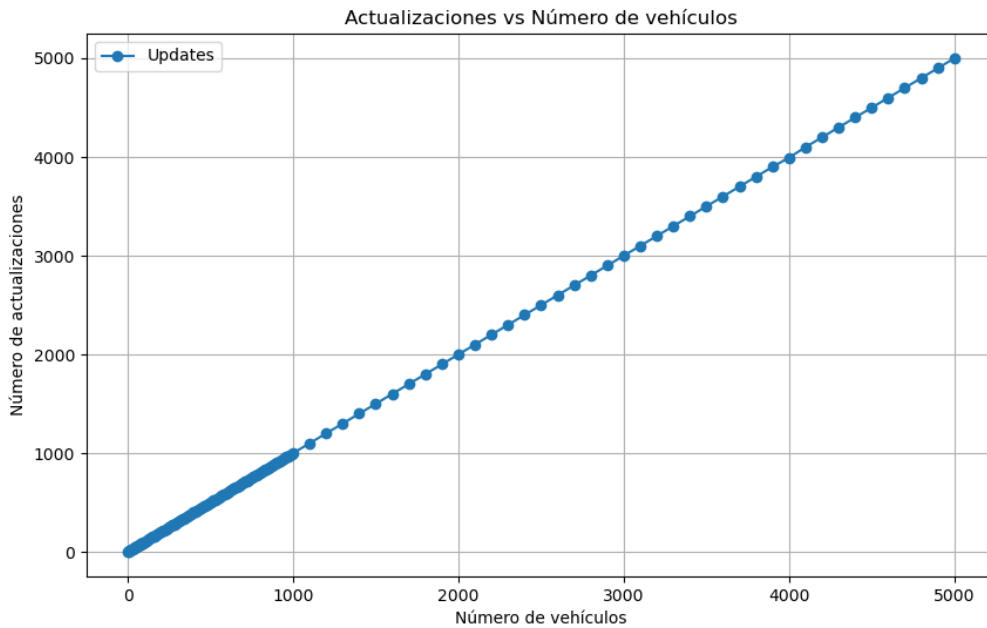


Figura 4.1: Evolución del número de actualizaciones respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias constantes

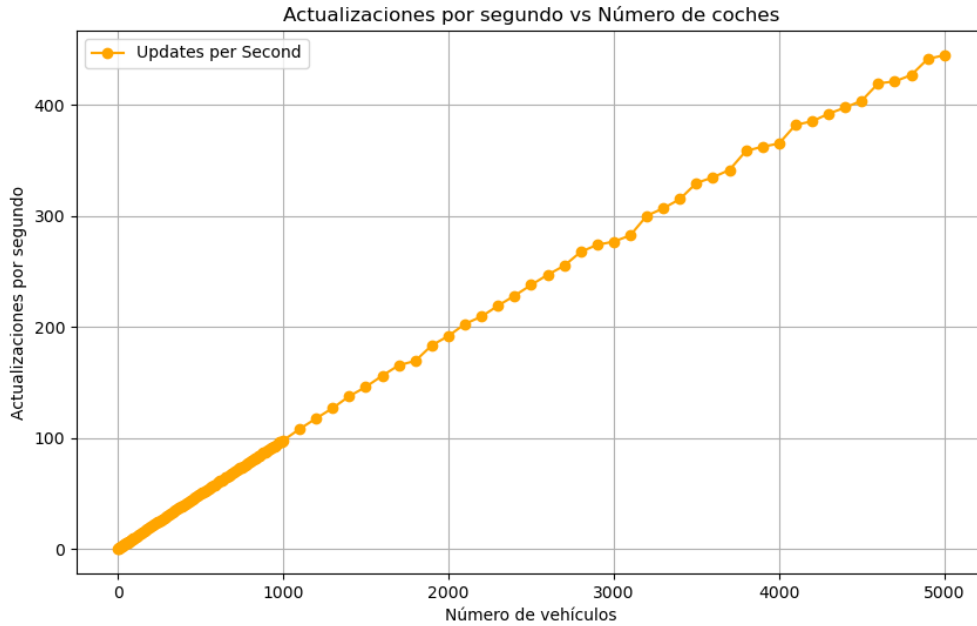


Figura 4.2: Evolución de la ratio de actualizaciones por segundo respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias constantes

Después, se simularon trayectorias aleatorias, muy difíciles de predecir (o prácticamente imposible) durante 10 iteraciones, con distintas cantidades de vehículos en la simulación. La evolución del número de actualizaciones se puede ver en la Figura 4.3, y la evolución de la ratio de actualizaciones por segundo se puede observar en la Figura 4.4.

El hecho de pasar de trayectorias constantes a erráticas supone pasar de 5.000 actualizaciones por 50.000 movimientos, es decir, un ahorro del 90 % en actualizaciones; a las 50.000, es decir, no ahorrar nada. En cuanto a la ratio de actualizaciones por segundo, se pasaría de unas 450 a unas 4.500, aproximadamente.

4.5.2. Rendimiento de los sistemas

En esta subsección se van a analizar los resultados de rendimiento obtenidos. En cada medición se ha recogido la latencia entre el inicio de la consulta y la emisión de la primera respuesta, y la media de las latencias entre las respuestas consecutivas.

Este apartado ha estado fuertemente condicionado por las características del entorno utilizado, descrito en la Sección 3.4. En un ordenador doméstico de 16 GB de RAM, se han ejecutado un total de 8 servidores a los que se ha sometido a pruebas de estrés. De hecho, se ha tenido que ejecutar el componente back-end en el ordenador

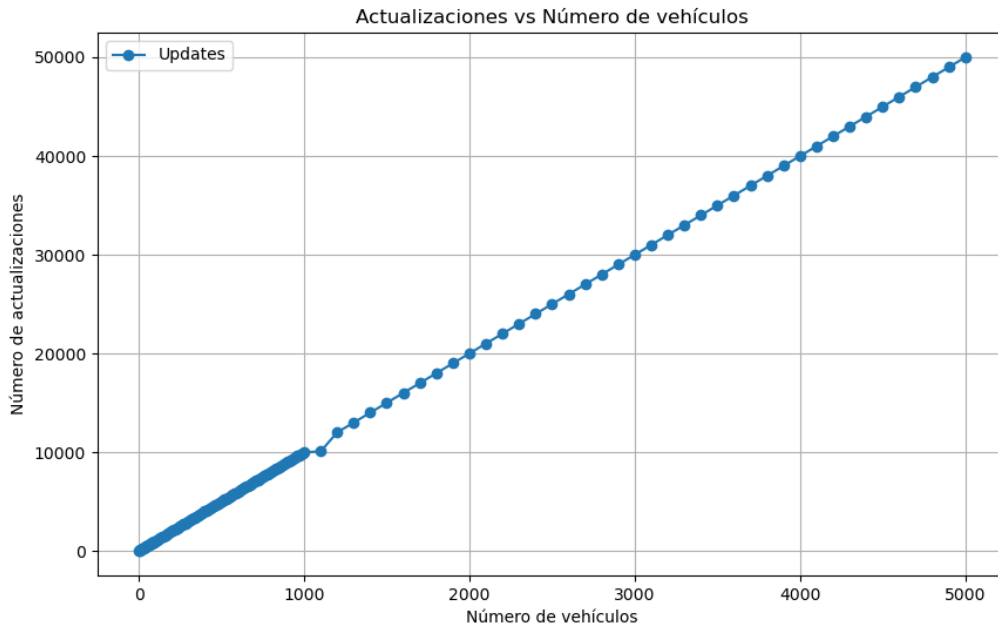


Figura 4.3: Evolución del número de actualizaciones respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias aleatorias

de sobremesa porque, de lo contrario, se experimentaban errores de red recurrentes a mitad de la prueba, disminuyendo la mayoría de la posible latencia de red existente en unas pruebas de estas características, debido a estar ejecutándose en la misma máquina que los servidores de los sistemas utilizados.

En primer lugar, en el primer escenario de aplicación se ha comprobado el impacto de un aumento del número de vehículos simulados en la latencia entre respuestas consecutivas. Para esto, se han generado 10 simulaciones distintas para cada número de vehículos. En concreto, se han utilizado simulaciones de 5, 25, 50 y 75 vehículos. Este número puede parecer pequeño, pero es hasta donde se ha podido llegar con el entorno disponible. Al generar los datos se ha utilizado una característica de MAVSIM+, que consiste en que se puede seleccionar la semilla utilizada para generar los datos de las trayectorias. De este modo, la simulación de 5 vehículos con semilla 1 está relacionada con la simulación de 25 vehículos de semilla 1, y así sucesivamente. Tomando las 10 mediciones diferentes dadas por las 10 semillas distintas se pretende compensar el efecto de la aleatoriedad, para poder ofrecer información fiable. En la Figura 4.5 se pueden observar los resultados obtenidos en la segunda consulta, que es la consulta más compleja que se ha podido implementar con los 4 sistemas. En la leyenda de la gráfica se ha abreviado el procesador de consultas basado en funciones de predicción por *dsms*, para compactar la leyenda. Se puede observar un crecimiento en la latencia

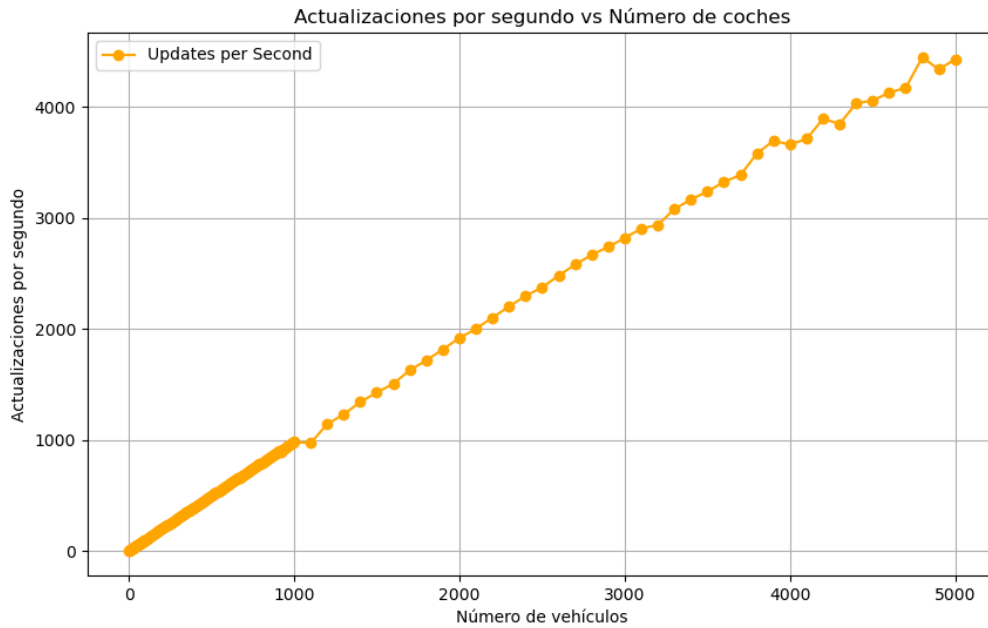


Figura 4.4: Evolución de la ratio de actualizaciones por segundo respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias aleatorias

en el procesador de consultas basado en funciones de predicción y Kafka Streams, el comportamiento esperable; aunque la disminución de la latencia de Pulsar functions y el descenso brusco de Redis functions se salen de la norma.

En segundo lugar, también en el primer escenario de aplicación, se ha probado el impacto de un aumento del número de consultas simultáneas en la latencia entre respuestas consecutivas, también en la segunda consulta propuesta. Para esto, utilizando las simulaciones creadas para la prueba anterior, se han ejecutado las 10 simulaciones de 5 vehículos con 5 instancias de la consulta ejecutándose simultáneamente en cada sistema, después se han vuelto a ejecutar con 25 consultas y, por último, con 50. De nuevo, las magnitudes de estos números pueden parecer pequeñas, pero ya con 50 consultas Kafka Streams ha dejado de funcionar y emitir resultados por la sobrecarga, y con 75 consultas el ordenador de sobremesa que ejecuta todos los servidores (incluido el del back-end de la herramienta desarrollada) se ha quedado sin recursos suficientes a mitad de la prueba. Los resultados obtenidos pueden observarse en la Figura 4.6. Redis functions y el procesador de consultas basado en funciones de predicción muestran el comportamiento esperado, un ascenso consistente, mientras que de forma llamativa Pulsar functions se mantiene bastante constante y Kafka Streams deja de funcionar y desaparece tras las 25 consultas.

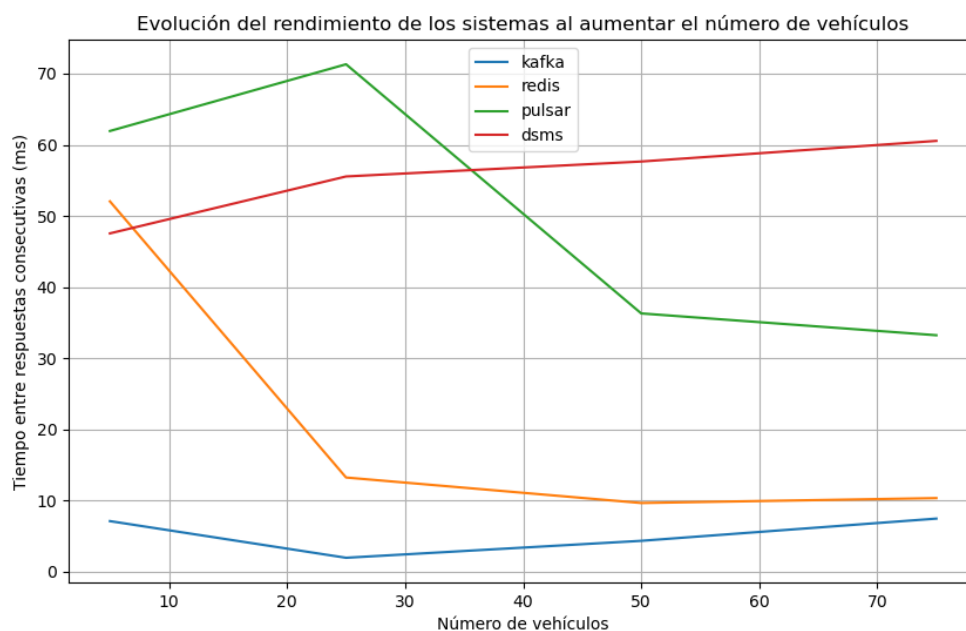


Figura 4.5: Evolución del rendimiento de los sistemas al aumentar el número de vehículos

En cuanto al segundo escenario de aplicación, el hecho de que las simulaciones duren 3 horas hace que los ficheros que almacenan dichos datos sean muy pesados (estamos hablando de GB) y, en el entorno disponible, esto hace que tan solo realizar la simulación ejecutando las dos consultas durante un periodo de tiempo prolongado hace que el ordenador en el que se ejecuta la herramienta se quede sin memoria RAM disponible e incluso la interfaz gráfica de su sistema operativo quede congelada. Por tanto, realizar pruebas de rendimiento en este escenario ha sido completamente inviable.

En cuanto al tercer escenario de aplicación, la generación de datos es mucho más lenta que en el primer escenario y las consultas son bastante más sencillas, además de que uno de los sistemas que se utiliza en el primer escenario (el procesador de consultas basado en funciones de predicción) no se ha podido utilizar en este. Por lo tanto, se ha priorizado evaluar el primer escenario, dado que los resultados de este podrían ser similares.

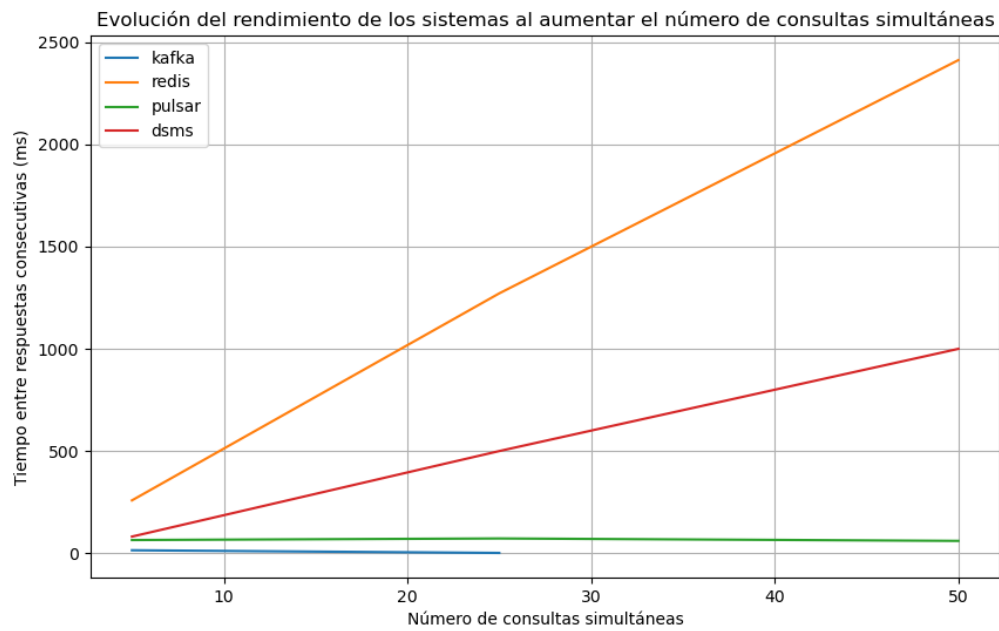


Figura 4.6: Evolución del rendimiento de los sistemas al aumentar el número de consultas simultáneas

Capítulo 5

Problemas encontrados y conclusiones del trabajo

Con este capítulo se finaliza la memoria de este proyecto, mencionando los problemas y errores más importantes que han surgido durante el desarrollo del trabajo (Sección 5.1), y estableciendo las conclusiones de este trabajo (Sección 5.2).

5.1. Problemas encontrados

Durante la implementación de este trabajo se han ido enfrentando bastantes retos, que se van a ir detallando en este capítulo, organizados según su problemática. Hay que destacar que la mayoría están relacionados con el componente back-end o un escenario de aplicación del sistema en general.

5.1.1. Base de datos

Para facilitar el acceso a las base de datos PostgreSQL y la persistencia de las entidades del componente back-end, se utiliza JPA como ORM. Sin embargo, tras muchos intentos, ha resultado imposible que JPA cree las tablas SQL correspondientes a las entidades por sí mismo. Por lo tanto, se ha tenido que crear un fichero con sentencias CREATE TABLE que se ha ido actualizando de forma manual, con los consecuentes despistes momentáneos de versiones que esto lleva a cometer.

Además, al utilizar el JPA que trae integrado Spring Boot, antes de iniciar el sistema se realiza la conexión a la base de datos. Si la base de datos no está disponible (ya sea no se haya encendido por despiste, o que la dirección IP de la máquina haya cambiado), salta una excepción que parece indicar que no termina de leer bien la configuración:

```
jpa: Caused by: org.hibernate.service.spi.ServiceException: Unable to
↳ create requested service
↳ [org.hibernate.engine.jdbc.env.spi.JdbcEnvironment] due to: Unable
↳ to determine Dialect without JDBC metadata (please set
↳ 'jakarta.persistence.jdbc.url' for common cases or
↳ 'hibernate.dialect' when a custom Dialect implementation must be
↳ provided)
```

Al conseguir tener disponible la base de datos en la dirección IP configurada, el error desaparece. Se destaca porque costó bastante averiguar qué estaba pasando la primera vez que ocurrió.

5.1.2. Conflictos de versiones Java

Inicialmente, el proyecto se había comenzado a programar en Java 22. Sin embargo, las bibliotecas de Apache Flink solo tienen compatibilidad hasta Java 17, por lo que se tuvo que portar el proyecto a Java 17, pero había módulos (MAVSIM+ y SimulParking) compilados en Java 22. Para resolver esto, como afortunadamente se disponía de su código fuente, solo hubo que recompilarlos. Hay que destacar que pese a haber trabajado en Java con anterioridad, nunca se había visto este tipo de errores, y costó averiguar qué estaba pasando.

5.1.3. Conflicto de dependencias internas de bibliotecas

Al añadir las bibliotecas de Apache Spark a las dependencias del proyecto, todo el componente back-end dejó de funcionar desde el arranque, sin dar ninguna explicación. La causa fue que colisionaron dos dependencias de logging utilizadas a la vez por varias dependencias, *log4j-to-slf4j* y *log4j-slf4j2-impl*. Para solucionar esto, se decidió excluir *log4j-to-slf4j* y hubo que ir viendo en qué paquetes se utilizaba para explícitamente en el fichero de configuración de Gradle decir que no lo utilizara.

5.1.4. Generación de datos para el benchmark Linear Road

Al intentar generar datos del benchmark Linear Road, con el software disponible en la página web de sus creadores, había que ejecutar unos scripts en Pearl que necesitaban la versión 7.0 como máximo de PostgreSQL debido a que la biblioteca de Pearl utilizada solo soporta la versión 2.0 de su protocolo de comunicación.

Se buscaron en internet los binarios de dicha versión de PostgreSQL, pero no se pudieron instalar en el computador que se ha utilizado, dado que los scripts de instalación no reconocen la arquitectura x86_64 de su CPU.

Se abrieron los ficheros de los scripts utilizados (bash) y se vio que el mensaje de error que aparecía estaba en un fragmento condicional en el que aparecía como opción válida la arquitectura i386. Se modificó cuidadosamente el script para que en las condiciones que reconocían i386 se entrara al tener un x86_64, pero el script se corrompió.

Finalmente, se encontró un repositorio en GitHub que solamente necesitaba Java para funcionar y generaba los datos adecuados para este benchmark, por lo que es la herramienta que ha generado los datos que se han utilizado en este caso de aplicación.

5.1.5. Paquete del componente back-end demasiado pesado

Al intentar cargar la primera Pulsar function en el servidor de Apache Pulsar, se excedían constantemente los tiempos límite de conexión, sin tener mayor explicación por parte del sistema.

Se revisó el tamaño del fichero .jar que se estaba intentando cargar y se probó a crear uno muy reducido (que solo contuviera esa clase) para poder diagnosticar el problema. Efectivamente, con el segundo fichero no hubo ningún error.

Cuando se metieron bibliotecas externas en las clases que implementan las Pulsar functions, hubo que convertir el proyecto Gradle en multimódulo para poder generar un fat jar (jar con el código de las Pulsar functions y todas sus dependencias) de tamaño limitado y de forma sencilla.

5.1.6. Generación del contenido estático del componente front-end

Para poder crear la aplicación de escritorio con Electron, hay que generar primero el contenido estático con Next JS. En la visualización de varias consultas continuas implementadas, se utilizan mapas, hechos con la biblioteca Leaflet. Esto ha generado muchos problemas, dado que Leaflet intenta acceder a un objeto *window* al cargarse, pero en la generación del contenido estático estos objetos no existen. Para solucionarlo, ha habido que importar dinámicamente la biblioteca React-Leaflet (utilizada por su facilidad de uso en React), importar de forma asíncrona Leaflet y esperar a que esté importado para empezar a renderizar el contenido.

5.2. Conclusiones y trabajo futuro

En esta sección voy a comentar mi propia conclusión personal (Subsección 5.2.1), la evaluación de los resultados finales del proyecto (Subsección 5.2.2), y varias posibilidades de ampliación en el futuro de este trabajo (Subsección 5.2.3).

5.2.1. Trabajo realizado y conclusión personal

Como conclusión personal, este ha sido un trabajo elegido con mucha ilusión, puesto que ya tenía experiencia previa con dos de los sistemas comparados, Apache Kafka y Redis, y además con el campo de aplicación del proyecto, las ciudades inteligentes, creo que se puede lograr un alto impacto de mejora en la calidad de vida de las personas.

Además, mi trayectoria profesional hasta la fecha ha estado muy ligada a las ciudades inteligentes y a los sistemas inteligentes de tráfico, tanto con prácticas en empresa, como con un proyecto realizado durante el instituto, como con este Trabajo Fin de Grado. Por ello, cuando vi este tema sugerido en la web de la escuela, casi sentí que llevaba mi nombre puesto.

Este trabajo ha sido mi primera toma de contacto con el mundo de la investigación académica, por lo que me llevo muchos aprendizajes y creo que escoger este trabajo fue una decisión muy positiva para mí.

Pese a tener ya una cierta experiencia técnica en estos sistemas y estar familiarizada con sus peculiaridades (al principio cuesta bastante entender la organización de topics, productores, consumidores, particiones, grupos de consumidores, suscripciones, etc.), ha sido un proyecto bastante complejo, en el que en ocasiones ha sido complicado percibir los avances realizados, y en el que se ha procurado prestar mucha atención al detalle.

Como consecuencia de su complejidad, se ha superado la cifra orientativa de 300 horas en base a la carga de créditos ECTS de la asignatura correspondiente al trabajo, habiendo utilizado un total de 460 horas en su realización. En la Tabla 5.1 se presenta el tiempo dedicado a cada tarea del proyecto y en la Figura 5.1 un cronograma de la distribución temporal que ha seguido cada bloque del proyecto.

Tarea	Horas
Reuniones de tutorización	12 h
Redacción de la propuesta	6 h
Diseño del sistema	11 h
Sistemas de mensajería y streaming	53 h
Interfaz	42 h
Simulación	94 h
Procesamiento	174 h
Pruebas de rendimiento	15 h
Memoria y documentación	53 h
Total	460 h

Tabla 5.1: Horas dedicadas al proyecto

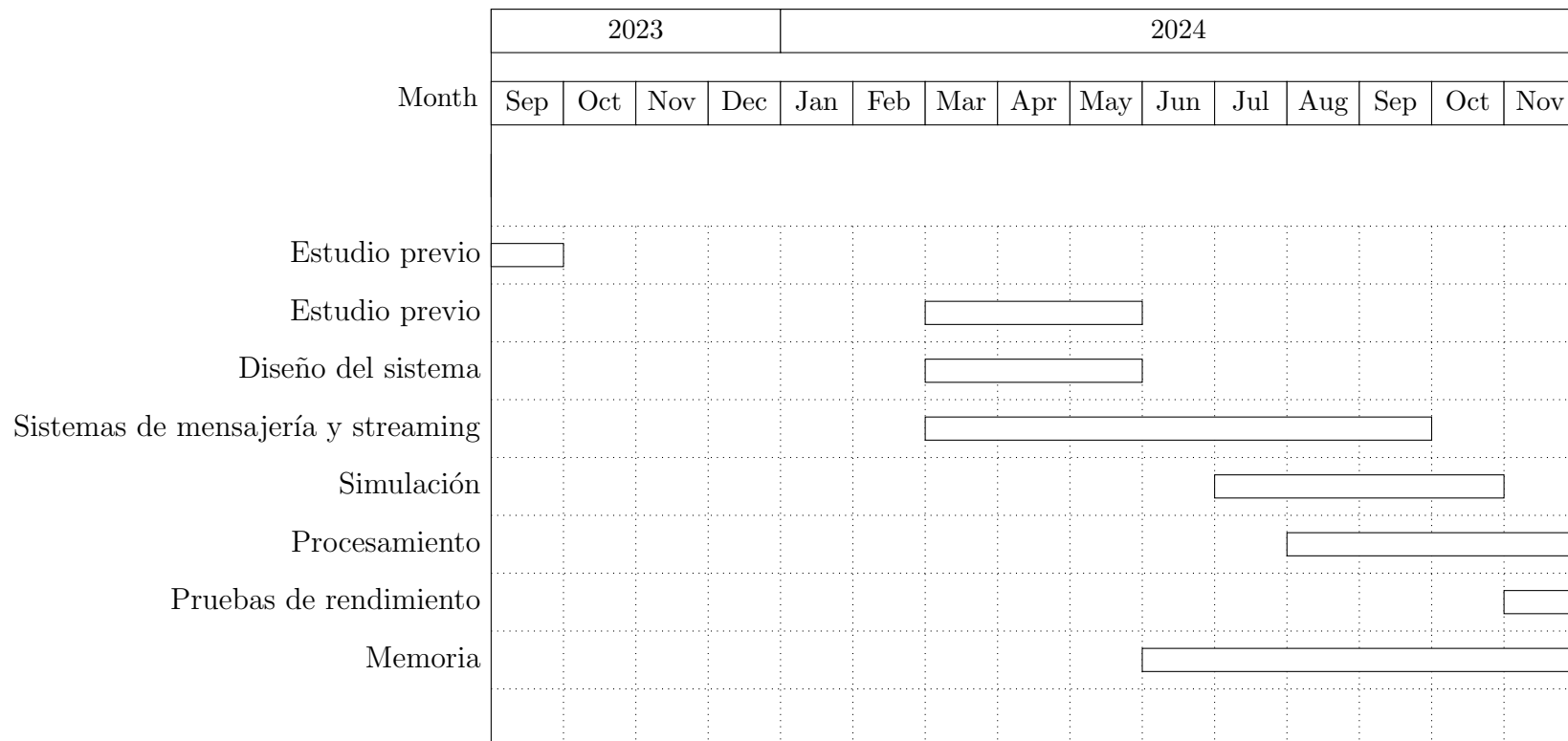


Figura 5.1: Diagrama de Gantt del proyecto

5.2.2. Conclusiones del proyecto

El objetivo del proyecto ha sido la evaluación de técnicas de gestión de flujos de datos en ciudades inteligentes, desarrollando una herramienta para simular escenarios y realizar consultas para permitir la comparación de las técnicas. Como resultados globales del proyecto se pueden destacar:

- Se han planteado tres escenarios de aplicación en ciudades inteligentes que explotan datos como apoyo a la toma de decisiones.
- Se ha logrado desarrollar un sistema web que permite simular la generación de flujos de datos y visualizarlos.
- El mismo sistema web permite iniciar consultas espaciales y espacio-temporales sobre los flujos de datos generados, y ver sus resultados.
- Se ha examinado el estado del arte de las tecnologías de manipulación y procesamiento de flujos de datos.
- Se ha logrado desarrollar un sistema web completo con dos componentes diferentes que se comunican entre sí en tiempo real, posibilitando así su distribución en diferentes máquinas físicas.

Estos resultados conllevan el cumplimiento de los objetivos planteados al inicio del trabajo.

5.2.3. Trabajo futuro

La herramienta implementada durante el proyecto podrá seguir siendo mejorada en un futuro. Algunas de las posibles tareas a realizar son:

1. Implementar el uso de técnicas de procesamiento de flujos de datos mediante lenguajes declarativos (como PulsarSQL).
2. En relación con la anterior, otra mejora podría ser permitir realizar cualquier consulta mediante una interfaz declarativa, en lugar de restringir las consultas a un conjunto concreto predefinido.
3. Al ser un sistema web, también podría resultar interesante implementar un control de acceso basado en usuarios para facilitar el uso de la herramienta por más de una persona a la vez, teniendo cada usuario sus propios ajustes de conexión a los sistemas externos utilizados, y sus propias simulaciones importadas.

4. Dada la volatilidad del entorno tecnológico actual y su alta tasa de cambio, es posible que en poco tiempo surgan nuevos sistemas de procesamiento de flujos de datos interesantes, que se podrían ir añadiendo a esta herramienta, para facilitar la evaluación de su rendimiento en contraste al rendimiento de los que ya están integrados.
5. También se podría mejorar el soporte de la herramienta desarrollada para realizar pruebas a gran escala de forma sencilla.

Sin duda, la herramienta desarrollada se puede mejorar (sería muy atrevido decir lo contrario), y estas son solo algunas ideas de los aspectos que podría ser más interesante añadir. Además, este trabajo inicial y el proyecto desarrollado puede ser útil en el contexto del proyecto NEAT-AMBIENCE (PID2020-113037RB-I00, financiado por MICIU/AEI/10.13039/501100011033) en el que se enmarca este trabajo, e incluso en el futuro para otras extensiones, mejoras y posibles trabajos derivados o relacionados desarrollados por investigadores de dicho proyecto.

Bibliografía

- [1] E. F. Codd, «A relational model of data for large shared data banks,» *Commun. ACM*, vol. 13, n.º 6, págs. 377-387, jun. de 1970, ISSN: 0001-0782. DOI: 10.1145/362384.362685. dirección: <https://dl.acm.org/doi/10.1145/362384.362685> (visitado 02-08-2024).
- [2] R. W. Taylor y R. L. Frank, «CODASYL Data-Base Management Systems,» en, *ACM Computing Surveys*, vol. 8, n.º 1, págs. 67-103, mar. de 1976, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/356662.356666. dirección: <https://dl.acm.org/doi/10.1145/356662.356666> (visitado 05-08-2024).
- [3] C. W. Bachman, «The programmer as navigator,» en, *Communications of the ACM*, vol. 16, n.º 11, págs. 653-658, nov. de 1973, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/355611.362534. dirección: <https://dl.acm.org/doi/10.1145/355611.362534> (visitado 05-08-2024).
- [4] *ISO 9075:1987 (SQL-86)*, en. dirección: <https://www.iso.org/standard/16661.html> (visitado 05-08-2024).
- [5] M. Atkinson, D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich y S. Zdonik, «The Object-Oriented Database System Manifesto,» en *Deductive and Object-Oriented Databases*, W. Kim, J.-M. Nicolas y S. Nishio, eds., Amsterdam: North-Holland, ene. de 1990, págs. 223-240, ISBN: 978-0-444-88433-6. DOI: 10.1016/B978-0-444-88433-6.50020-4. dirección: <https://www.sciencedirect.com/science/article/pii/B9780444884336500204> (visitado 05-08-2024).
- [6] M. Stonebraker, L. A. Rowe, B. G. Lindsay et al., «Third-generation database system manifesto,» en, *ACM SIGMOD Record*, vol. 19, n.º 3, págs. 31-44, sep. de 1990, ISSN: 0163-5808. DOI: 10.1145/101077.390001. dirección: <https://dl.acm.org/doi/10.1145/101077.390001> (visitado 05-08-2024).
- [7] H. Darwen y C. J. Date, «The third manifesto,» *SIGMOD Rec.*, vol. 24, n.º 1, págs. 39-49, mar. de 1995, ISSN: 0163-5808. DOI: 10.1145/202660.202667. dirección: <https://dl.acm.org/doi/10.1145/202660.202667> (visitado 05-08-2024).
- [8] C. Date y H. Darwen, «Databases, Types, and the Relational Model: The Third Manifesto (3rd edition) chapter 11,» en, en *Databases, Types, and the Relational Model: The Third Manifesto (3rd edition)*, Reading, Mass.: Addison-Wesley, 2006. dirección: <https://www.dcs.warwick.ac.uk/~hugh/TTM/DBE-Chapter11.pdf>.
- [9] *ISO/IEC 9075-1:1999 (SQL-99)*, en. dirección: <https://www.iso.org/standard/26196.html> (visitado 05-08-2024).

- [10] S. Chaudhuri y U. Dayal, «An overview of data warehousing and OLAP technology,» *SIGMOD Rec.*, vol. 26, n.º 1, págs. 65-74, mar. de 1997, ISSN: 0163-5808. DOI: 10.1145/248603.248616. dirección: <https://dl.acm.org/doi/10.1145/248603.248616> (visitado 10-08-2024).
- [11] C. Strauch, U.-L. S. Sites y W. Kriha, «NoSQL databases,» *Lecture Notes, Stuttgart Media University*, vol. 20, n.º 24, pág. 79, 2011.
- [12] E. A. Brewer, «Towards robust distributed systems (abstract),» en *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, ép. PODC '00, New York, NY, USA: Association for Computing Machinery, jul. de 2000, pág. 7, ISBN: 978-1-58113-183-3. DOI: 10.1145/343477.343502. dirección: <https://dl.acm.org/doi/10.1145/343477.343502> (visitado 10-08-2024).
- [13] M. Aslett, *How will the database incumbents respond to NoSQL and NewSQL*, 2011. dirección: <https://cs.brown.edu/courses/cs227/archives/2012/papers/newsq/aslett-newsq.pdf>.
- [14] M. Stonebraker, «New opportunities for New SQL,» en, *Communications of the ACM*, vol. 55, n.º 11, págs. 10-11, nov. de 2012, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/2366316.2366319. dirección: <https://dl.acm.org/doi/10.1145/2366316.2366319> (visitado 10-08-2024).
- [15] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil y D. Barton, «Big data: the management revolution,» *Harvard business review*, vol. 90, n.º 10, págs. 60-68, 2012, Publisher: Cambridge.
- [16] D. Laney et al., «3D data management: Controlling data volume, velocity and variety,» *META group research note*, vol. 6, n.º 70, pág. 1, 2001, Publisher: Stanford.
- [17] J. Dean y S. Ghemawat, «MapReduce: simplified data processing on large clusters,» en, *Communications of the ACM*, vol. 51, n.º 1, págs. 107-113, ene. de 2008, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1327452.1327492. dirección: <https://dl.acm.org/doi/10.1145/1327452.1327492> (visitado 24-07-2024).
- [18] A. Fernández, C. J. Carmona, M. J. d. Jesus y F. Herrera, «A View on Fuzzy Systems for Big Data: Progress and Opportunities,» en, *International Journal of Computational Intelligence Systems*, vol. 9, n.º Supplement 1, págs. 69-80, abr. de 2016, Publisher: Atlantis Press, ISSN: 1875-6883. DOI: 10.1080/18756891.2016.1180820. dirección: <https://www.atlantis-press.com/journals/ijcis/25868762> (visitado 03-07-2024).
- [19] E. W. Dijkstra, *E.W. Dijkstra Archive: My recollections of operating system design (EWD1303)*, 2000. dirección: <https://www.cs.utexas.edu/~EWD/transcriptions/EWD13xx/EWD1303.html> (visitado 10-08-2024).
- [20] S. Geisler, «Data Stream Management Systems,» en, *DROPS-IDN/v2/document/10.4230/DFU.Vol5.10452.275*, 2013, Publisher: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. DOI: 10.4230/DFU.Vol5.10452.275. dirección: <https://drops.dagstuhl.de/entities/document/10.4230/DFU.Vol5.10452.275> (visitado 28-07-2024).

- [21] Y.-N. Law, H. Wang y C. Zaniolo, «Query languages and data models for database sequences and data streams,» en *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, págs. 492-503.
- [22] M. Stonebraker, U. Çetintemel y S. Zdonik, «The 8 requirements of real-time stream processing,» *ACM SIGMOD Record*, vol. 34, n.º 4, págs. 42-47, dic. de 2005, ISSN: 0163-5808. DOI: 10.1145/1107499.1107504. dirección: <https://dl.acm.org/doi/10.1145/1107499.1107504> (visitado 18-03-2024).
- [23] R. H. Arpaci-Dusseau y A. C. Arpaci-Dusseau, *Operating systems: Three easy pieces*, 1.10. Arpaci-Dusseau Books, nov. de 2023, vol. CPU Scheduling. dirección: <https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf>.
- [24] K. Stolze, «SQL/MM spatial: The standard to manage spatial data in a relational database system,» en, Gesellschaft für Informatik e.V., 2003, págs. 247-264, ISBN: 978-3-88579-355-7. dirección: <https://dl.gi.de/handle/20.500.12116/30056> (visitado 23-07-2024).
- [25] A. Arasu, M. Cherniack, E. Galvez et al., *Linear road — Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*. dirección: <https://dl.acm.org/doi/10.5555/1316689.1316732> (visitado 18-08-2024).
- [26] S. Ilarri, O. Wolfson, E. Mena, A. Illarramendi y P. Sistla, «A query processor for prediction-based monitoring of data streams,» en, en *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, Saint Petersburg Russia: ACM, mar. de 2009, págs. 415-426, ISBN: 978-1-60558-422-5. DOI: 10.1145/1516360.1516409. dirección: <https://dl.acm.org/doi/10.1145/1516360.1516409> (visitado 11-02-2024).
- [27] Ó. Cardiel Enguita y S. Ilarri Artigas, *Sistema de procesamiento de data streams*, spa. Zaragoza: Universidad de Zaragoza, 2014. dirección: <https://zaguan.unizar.es/record/31564>.
- [28] S. Ilarri, R. Trillo-Lado, Á. Arraez y A. Piedrafita, «Simulating Scenarios to Evaluate Data Filtering Techniques for Mobile Users,» en, en *Advances in Mobile Computing and Multimedia Intelligence*, P. Delir Haghighi, I. Khalil y G. Kotsis, eds., Cham: Springer Nature Switzerland, 2022, págs. 87-101, ISBN: 978-3-031-20436-4. DOI: 10.1007/978-3-031-20436-4_9.
- [29] S. Ilarri y R. Trillo-Lado, «Simulating Scenarios for the Evaluation of Data Management Approaches for Searching Parking Spaces,» en *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, ISSN: 2766-8576, mar. de 2024, págs. 382-384. DOI: 10.1109/PerComWorkshops59983.2024.10503310. dirección: <https://ieeexplore.ieee.org/abstract/document/10503310> (visitado 02-08-2024).

Lista de Figuras

2.1. El modelo de programación MapReduce. Fuente: [18]	7
3.1. Diagrama de la arquitectura del sistema	27
3.2. Esquema relacional de las entidades en PostgreSQL. Generado con DataGrip	31
3.3. Diagrama de ejemplo de las relaciones entre controladores, servicios, repositorios y entidades de Spring Boot	32
3.4. Diagrama de las clases de acceso a los sistemas de mensajería y streaming	32
3.5. Diagrama de las clases que modelan los mensajes y eventos generados en la aplicación	32
3.6. Diagrama de clases de simulaciones	33
3.7. Diagrama de clases	33
3.8. Diagrama de clases	33
3.9. Diagrama de paquetes del componente back-end	34
4.1. Evolución del número de actualizaciones respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias constantes	45
4.2. Evolución de la ratio de actualizaciones por segundo respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias constantes	46
4.3. Evolución del número de actualizaciones respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias aleatorias	47
4.4. Evolución de la ratio de actualizaciones por segundo respecto al aumento en el número de vehículos en el procesador de consultas con funciones de predicción con trayectorias aleatorias	48
4.5. Evolución del rendimiento de los sistemas al aumentar el número de vehículos	49

4.6. Evolución del rendimiento de los sistemas al aumentar el número de consultas simultáneas	50
5.1. Diagrama de Gantt del proyecto	56
D.1. Home screen in English	82
D.2. Home screen in Spanish	83
D.3. Position of the buttons to swipe languages	83
D.4. Location of the toggle button on lateral panel	84
D.5. Location of the untoggle button on lateral panel	84
D.6. Position of the streaming servers area of the lateral panel	85
D.7. Drop-down menu to add a new streaming server	85
D.8. First part of the Apache Kafka settings dialog	86
D.9. Second part of the Apache Kafka settings dialog	87
D.10. Location of the unfolding buttons on streaming servers panel section . .	88
D.11. Topics folder location after unfold	88
D.12. Topic list location after unfold	89
D.13. Location of <i>View data</i> button	90
D.14. Page for visualizing topic data on streaming systems	91
D.15. Graphical aspect of the drop-down menu to sort and filter row data at grids	91
D.16. Position of <i>Simulation list</i> button on the menu	91
D.17. Page showing a list of the saved simulations	92
D.18. Information about a simulation	92
D.19. Interface feedback on selected simulation	93
D.20. Selected simulation name set on lateral panel	93
D.21. Edit simulation dialog	94
D.22. Position of the option to import simulations on menu	94
D.23. Initial import simulation page	94
D.24. Import vehicle mobility simulation page	95
D.25. Import vehicle mobility simulation page with data format details	95
D.26. Import Linear Road simulation page	96
D.27. Import parking lot simulation page	96
D.28. Position of the option to export simulations on menu	96
D.29. Position of the button to view continuous queries results on menu . . .	97
D.30. View continuous queries page	97
D.31. How to select a continuous query to view its results	97
D.32. Speed limit query visualization	98

D.33.Average speed per km^2 query visualization	98
D.34.Vehicles close to each other query visualization	98
D.35.Distance traveled by vehicle query visualization	99
D.36.Parking times query visualization	99
D.37.Position of logging access on menu	99
D.38.Logging visualization page	100
D.39.Position of the <i>Settings</i> button on the menu	100
D.40.Settings page	100
D.41.Apache Kafka settings page	101

Lista de Tablas

2.1. Comparativa de características de Apache Kafka, Apache Pulsar y Redis Streams	12
2.2. Comparativa de características de Apache Beam, Apache Flink, Apache Hadoop, Apache Samza, Apache Spark y Apache Storm	17
2.3. Comparativa de características de Apache Sedona, GeoMesa, Simba y LocationSpark	20
4.1. Evaluación cualitativa de los 4 sistemas de procesamiento de flujos de datos a utilizar	42
4.2. Resumen de la viabilidad de la implementación de las consultas propuestas en Apache Kafka, Apache Pulsar, Redis y el procesador de consultas basado en funciones de predicción	44
5.1. Horas dedicadas al proyecto	55
A.1. Requisitos no funcionales de la aplicación	71
A.2. Requisitos funcionales de la aplicación	72

Anexos

Anexos A

Análisis de requisitos

En este anexo se presenta el análisis de requisitos de la aplicación desarrollada, incluyendo los requisitos funcionales y los no funcionales.

A.1. Requisitos no funcionales

Tabla A.1: Requisitos no funcionales de la aplicación

Código	Descripción
RNF 1	El sistema debe ser multiplataforma.
RNF 2	El sistema debe ser una aplicación de escritorio, es decir, funcionar en los sistemas operativos Windows, Mac OS y Unix.
RNF 3	El sistema debe permitir evaluar técnicas de gestión de flujos de datos.
RNF 4	El sistema debe soportar internacionalización, siendo posible utilizar su interfaz en inglés y en español.
RNF 5	Las simulaciones importadas deben tener persistencia.
RNF 6	El sistema debe permitir simular la recepción de flujos de datos procedentes de sensores a través de Apache Kafka, Apache Pulsar y Redis Streams.
RNF 7	El sistema debe permitir implementar el procesamiento de datos en las plataformas Apache Spark y Apache Flink.
RNF 8	El sistema debe ofrecer una API REST.

A.2. Requisitos funcionales

Tabla A.2: Requisitos funcionales de la aplicación

Código	Descripción
RF 1	La aplicación permite importar y exportar simulaciones de tráfico realizadas con MAVSIM+ para incorporarlas a los sistemas de procesamiento.
RF 2	La aplicación permite importar y exportar ficheros de extensión .csv que contengan información de simulaciones de tráfico realizadas con MAVSIM+ para incorporarlas a los sistemas de procesamiento.
RF 3	Los datos generados por la simulación realizada con MAVSIM+ se podrán comunicar a los sistemas de procesamiento para ser tratados como información procedente de sensores.
RF 4	La aplicación permite importar y exportar trazas GPS de movimientos de sensores para incorporarlas a los sistemas de procesamiento.
RF 5	Los datos generados por la simulación de las trazas GPS se podrán comunicar a los sistemas de procesamiento para ser tratados como información procedente de sensores.
RF 6	La aplicación permite importar y exportar simulaciones de tráfico en autopistas siguiendo el formato del benchmark LinearRoad para incorporarlas a los sistemas de procesamiento.
RF 7	Los datos generados por la simulación de tráfico en autopistas LinearRoad se podrán comunicar a los sistemas de procesamiento para ser tratados como información procedente de sensores.
RF 8	La aplicación permite importar y exportar simulaciones de aparcamiento realizadas con SimulParking para incorporarlas a los sistemas de procesamiento.
RF 9	Los datos generados por la simulación realizada con SimulParking se podrán comunicar a los sistemas de procesamiento para ser tratados como información procedente de sensores.
RF 10	La aplicación permite crear consultas de snapshot sobre los datos de las simulaciones.
RF 11	La aplicación dispone de consultas continuas predefinidas.
RF 12	La aplicación permite visualizar en tiempo real el resultado de las consultas continuas conforme se vayan produciendo.
RF 13	La aplicación permite acceder a los resultados generados por una consulta de snapshot en cualquier momento, salvo que el usuario los haya eliminado.
RF 14	La aplicación permite almacenar y modificar configuraciones de servidores de los sistemas de mensajería y streaming de datos.
RF 15	La aplicación permite consultar en tiempo real los flujos de datos existentes en los sistemas cuyo acceso haya sido configurado.
RF 16	La aplicación permite eliminar los mensajes de los flujos de datos existentes en los sistemas cuyo acceso haya sido configurado.

Anexos B

Fragmentos de código destacados

En este anexo se presentan varios fragmentos de código referenciados desde otros puntos de la memoria, que se reúnen aquí para mayor comodidad.

Listing B.1: Código fuente en Java que permite iniciar, pausar, reiniciar y parar la reproducción de una simulación

```
1 public class SimulationService {
2
3     private volatile boolean paused = false;
4
5     public void startSimulation (Simulation simulation) {
6         if (!paused) {
7             switch (simulation.getType()) {
8                 case "car-movement": {
9                     TopicName topic = topicNameService
10                        .findByValueClassName("car-movement");
11                     if (topic != null) {
12                         createTopicIfNotExists(topic);
13                     }
14                     checkPaused();
15                     carMovementSimulationService.startQueries();
16                     checkPaused();
17                     carMovementSimulationService
18                        .startSimulation(simulation.getId(),
19                                       UPLOAD_DIR, this::checkPaused);
20                     break;
21                 }
22                 case "linear-road": {
23                     TopicName topic = topicNameService
24                        .findByValueClassName("linear-road");
25                     if (topic != null) {
26                         createTopicIfNotExists(topic);
27                     }
28                     checkPaused();
29                     linearRoadSimulationService.startQueries();
30                     checkPaused();
31                     linearRoadSimulationService
32                        .startSimulation(simulation.getId(),
33                                       UPLOAD_DIR, this::checkPaused);
34                     break;
35                 }
36                 case "car-parking": {
```

```

37         TopicName topic = topicNameService
38             .findByValueClassName("car-parking");
39         if (topic != null) {
40             createTopicIfNotExists(topic);
41         }
42         checkPaused();
43         carParkingSimulationService.startQueries();
44         checkPaused();
45         carParkingSimulationService
46             .startSimulation(simulation.getId(),
47                 UPLOAD_DIR, this::checkPaused);
48         break;
49     }
50     default: {
51         break;
52     }
53 }
54 } else {
55     reStartSimulation();
56 }
57 }
58
59 public void pauseSimulation () {
60     synchronized (this) {
61         paused = true;
62     }
63 }
64
65 public void reStartSimulation () {
66     synchronized (this) {
67         paused = false;
68         this.notifyAll();
69     }
70 }
71
72 public void stopSimulation (Simulation simulation) {
73     synchronized (this) {
74         paused = false;
75     }
76     switch (simulation.getType()) {
77         case "car-movement" -> carMovementSimulationService.
78             stopSimulation(simulation.getId());
79         case "linear-road" -> linearRoadSimulationService
80             .stopSimulation(simulation.getId());
81         case "car-parking" -> carParkingSimulationService
82             .stopSimulation(simulation.getId());
83     }
84 }
85
86 public void checkPaused() {
87     synchronized (this) {
88         while (paused) {
89             try {
90                 this.wait();
91             } catch (InterruptedException e) {
92                 Thread.currentThread().interrupt();
93                 return;
94             }

```

```

95     }
96   }
97 }
98 }

```

Listing B.2: Código fuente en JavaScript que permite que el contenido de la página se expanda o encoja al encoger o expandir el panel lateral

```

1  const drawerWidth = 300;
2
3  const ContentWrapper = styled('div', {
4    shouldForwardProp: (prop) => prop !== 'open',
5  })(({ theme, open }) => ({
6    flexGrow: 1,
7    padding: theme.spacing(3),
8    transition: theme.transitions.create('margin', {
9      easing: theme.transitions.easing.sharp,
10     duration: theme.transitions.duration.leavingScreen,
11   }),
12   marginLeft: `calc(${theme.spacing(6)} + 1px)`,
13   ...(open && {
14     transition: theme.transitions.create('margin', {
15       easing: theme.transitions.easing.easeOut,
16       duration: theme.transitions.duration.enteringScreen,
17     }),
18     marginLeft: `${drawerWidth}px`,
19   }),
20 }));
21
22 export default ContentWrapper;

```


Anexos C

Installation guide

This appendix provides the the necessary instructions to deploy the setup used to develop and test the system created in this project. The instructions are given in English, as the interface created supports both English and Spanish speakers.

First, download all the source code of the project. This guide assumes that the steps are followed on a computer with Docker installed, which will make the setup easier. If you do not have docker installed or are unfamiliar with it, visit <https://www.docker.com/> for additional information. If you are unsure whether Docker is installed or not, open a terminal and type:

```
docker version
```

This guide provides instructions for starting the servers of all the external systems required (in Section C.1), starting the back-end component (in Section C.2), and starting the front-end component (in C.3).

C.1. External systems (database, data streaming systems)

On a computer with Docker installed, navigate to the directory **backend/docker**. You will find three files there: *processing.docker-compose.yaml*, *postgres.docker-compose.yaml* and *streaming.docker-compose.yaml*. These files contain the configuration settings for all the external servers required.

Now to set up a PostgreSQL database server, open a terminal and type

```
docker compose up -f postgres.docker-compose.yaml -d
```

That should be enough to have a PostgreSQL database server running.

In order to run Apache Kafka, Apache Pulsar and Redis, open the *streaming.docker-compose.yaml* file and make sure to replace all occurrences of the string **192.168.0.101** with the IP address of your own machine on the network you're using. This step is essential. There should be four occurrences of this address, three in *pulsar-broker* and one in *kafka*. Once you have made the changes and saved the file, go back to the terminal and type

```
docker compose up -f streaming.docker-compose.yaml -d
```

This will start six new Docker containers.

If you want to process snapshot queries, type the following in the terminal:

```
docker compose up -f processing.docker-compose.yaml -d
```

No changes are required in the processing file.

Now, all the necessary external systems should be running.

C.2. Back-end

To run the back-end component, two options are available: running it as a Docker container (described in Subsection C.2.1) or running it with Java (described in Subsection C.2.2).

C.2.1. Docker

First of all, it is very important that connections can be established with PostgreSQL database. Navigate to the **backend/api/build/libs** and modify the content of the file **BOOT-INF/classes/application.properties** to fit your network (IP address) configuration.

Alternatively, you can modify the content of the same file located in the **backend/api/src/main/resources/application.properties** and then rebuild the .jar file after making sure you have Gradle installed and are using Java 17, typing this in a terminal on **backend** directory:

```
./gradlew :api:bootJar
```

To run the back-end of the application in Docker, navigate to the **backend/api** directory and type the following in the terminal:

```
docker build -t backend:latest .
```

Then, run the container with:

```
docker run --name <name> -p 8080:8080 backend
```

C.2.2. Native (java + gradle)

If you prefer not to use Docker, make sure you have Gradle installed and are using Java 17.

First of all, modify the content of

backend/api/src/main/resources/application.properties file to fit your network (PostgreSQL database IP address) configuration. Then, navigate to the **backend** directory and type the following in the terminal:

```
./gradlew :api:bootRun
```

C.3. Front-end

To run the front-end component, two options are available: via Docker or via npm. The first option is addressed in Subsection C.3.1 and the second in Subsection C.3.2.

On top of the interface web application server, you can run a desktop application with electron. This is addressed in Subsection C.3.3.

It is very important that before any further step, you open the file **frontend/.env** and modify the proposed IP address of the back-end application if needed.

C.3.1. Docker (web application)

To run the front-end of the application in Docker, navigate to the **frontend** directory and type the following in the terminal:

```
docker build -t frontend:latest .
```

Then, run the container with:

```
docker run --name <name> -p 3000:3000 frontend
```

By doing so, the interface of the application should be available at port 3000.

C.3.2. Native (web application)

If you prefer not to use Docker, navigate to the **frontend** directory and type the following in the terminal:

```
npm install
```

Then, build the application with:

```
npm run build
```

Finally, start the application with:

```
npm run start-next
```

By doing so, the web interface of the application should be available at port 3000.

C.3.3. Native (desktop application)

To launch the desktop application, navigate to the **frontend** directory and install electron by typing the following in the terminal:

```
npm install electron
```

Then, start the desktop application with:

```
electron .
```

By doing so, a new window should appear on your computer.

Anexos D

User's manual

This appendix provides a user's manual about the application developed in this project. All the characteristics of the application will be explained, with an introduction on the system (Section D.1), how to start the application (Section D.2), the home screen (Section D.3), the lateral panel (Section D.4), how to visualize streaming data (Section D.5), simulations (Section D.6), continuous queries (Section D.7), logging (Section D.8) and configuration (Section D.9).

D.1. Introduction

The application created in this project simplifies working with data streaming systems by providing an intuitive interface for data loading, visualization, and querying.

It has two main functionalities: load data into data streaming systems and enable its visualization, and make continuous queries over that data and offer a proper visualization.

D.2. Starting the application

On Appendix C there are described all the steps necessary to run the application.

D.3. Home screen

When the application is open, home screen will appear, including a menu bar, a lateral panel and a home page. Its appearance is shown on Figure D.1. If the system's language is set to Spanish, the home screen will appear as shown in Figure D.2.

If you want to change the language of the interface, just click on the proper language name beside their flag, as shown in Figure D.3.

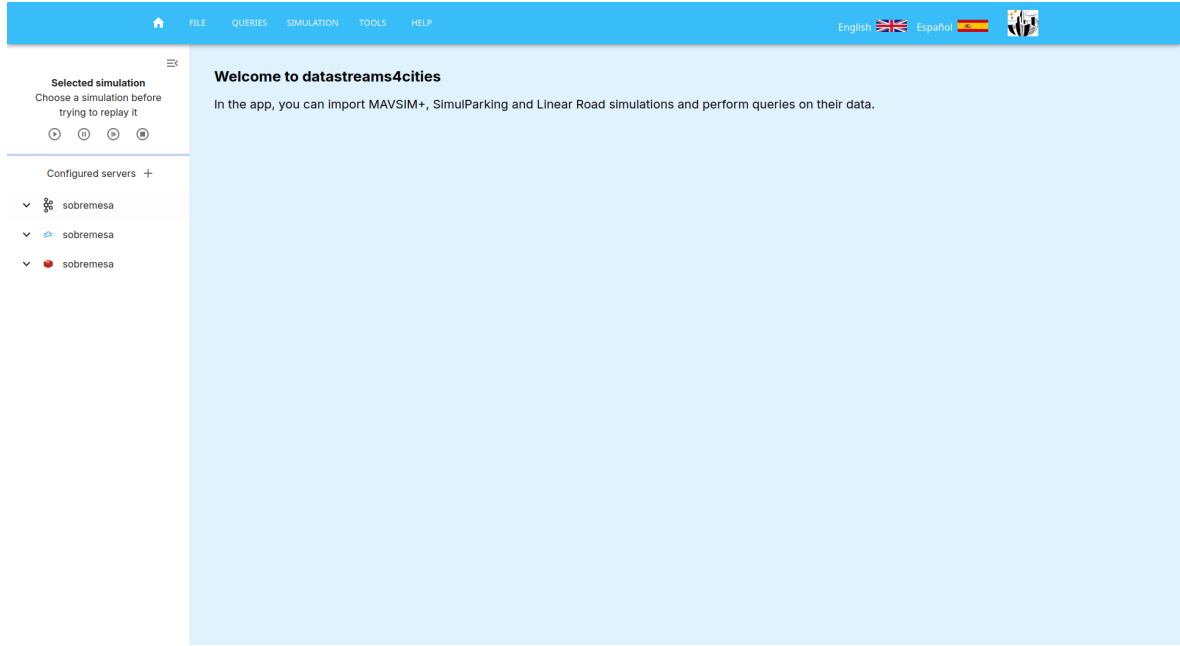


Figura D.1: Home screen in English

D.4. Lateral panel

At the lateral panel, there are two different sections. The first one is a little playing menu for enabling users to play simulations from any part of the application. As the application has just been started, no simulation is selected yet. If you want to select a simulation, follow the steps on Subsection D.6.1. The second section offers a broad view over the streaming systems whose access has been set up. More information is provided in Subsection D.4.1.

Furthermore, the lateral panel can also be toggled, by clicking on the button on the top right side, as shown in Figure D.4. Similarly, to untoggle the panel click on an analog button, as shown in Figure D.5.

D.4.1. Streaming servers

This subsection explains the area of the lateral panel shown in Figure D.6. If you have not configured any server, no servers will appear.

If you want to add a new server, click on the add button and you will find a drop-down menu, as shown in Figure D.7.

If *Apache Kafka* button is clicked, the dialog shown in D.8 will appear. There are some fields with predefined values, but it is advisable to modify their values to fit your setup. Once you have filled all the fields, scroll down and there will appear two buttons below, as shown in Figure D.9. Click on *OK* button to confirm the configuration and

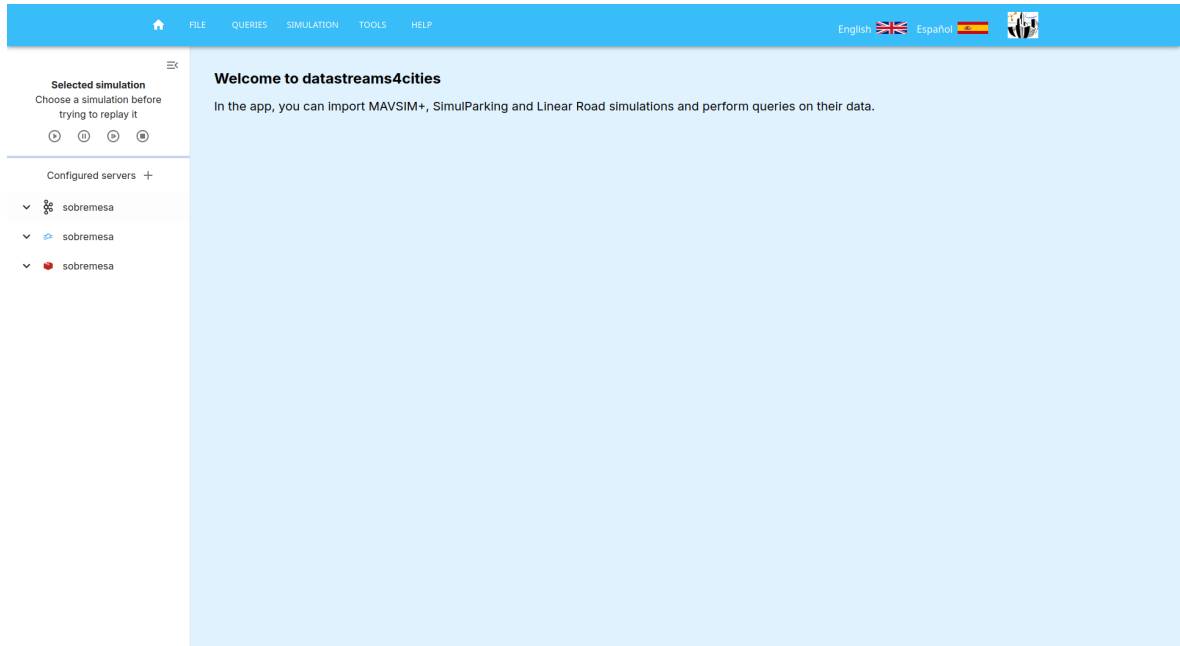


Figura D.2: Home screen in Spanish



Figura D.3: Position of the buttons to swipe languages

it will appear on the lateral panel.

By doing the same with *Apache Pulsar* and *Redis Streams* options, similar dialogs will be displayed.

It is also possible to explore the topics available in each server from the lateral panel. To do so, click on the unfolding icon near to the server you want to explore, as shown in Figure D.10. It will appear an extra line with a folder icon, as shown in Figure D.11. If the new unfold icon is clicked, a list of the available topics will appear, as in Figure D.12.

D.5. Streaming data

To visualize the data contained in a topic, toggle the lateral panel, and unfold the server you want to explore until you get the list of its topics. Then, left click on the topic you want to examine and click on the first option, as shown in Figure D.13. You

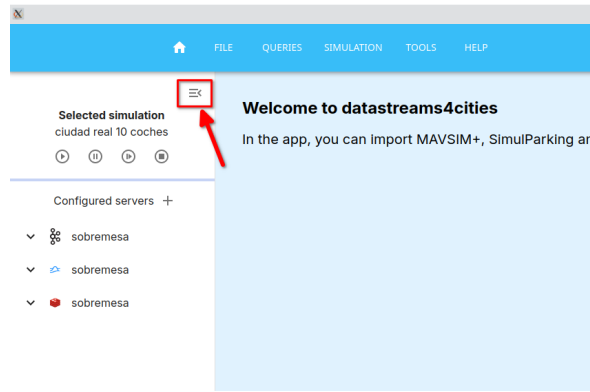


Figura D.4: Location of the toggle button on lateral panel

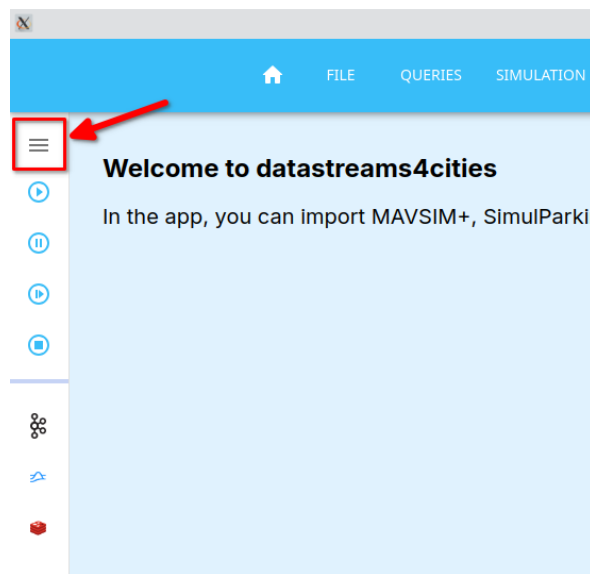


Figura D.5: Location of the untoggle button on lateral panel

might as well just double click on the name of the topic and the result will be the same.

Now, your screen should be similar to what is shown in Figure D.14. If you have not played any simulation, there will not be any data available. To do so, go to Section D.6. In this page, you will be able to view in real-time all the data stored in the topic.

At this page by default there will be only 5 rows showed, but it is possible to see up until 25 rows together. To switch among data rows, just click on the arrows on the right bottom of the grid. The width of the columns can also be modified, by dragging the dividers on the headers with the mouse. You might as well filter and sort the rows by the content of some of the columns. To do so, hover around the desired column and click on the three dots appearing, as shown in Figure D.15. It is also possible to download a .csv file with all the data in the topic by clicking the blue button. You can also switch between topics by clicking the desired tab.

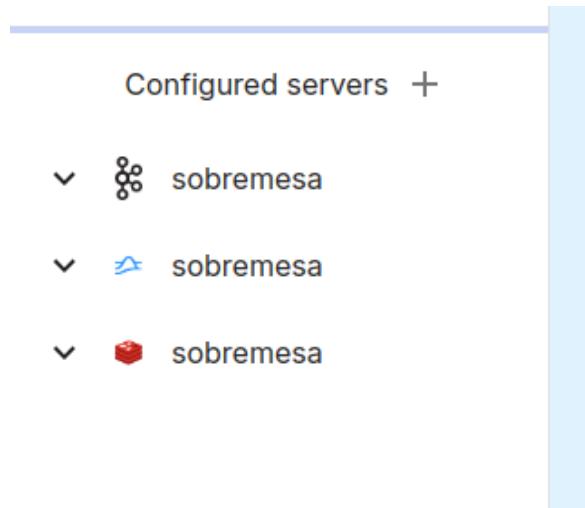


Figura D.6: Position of the streaming servers area of the lateral panel

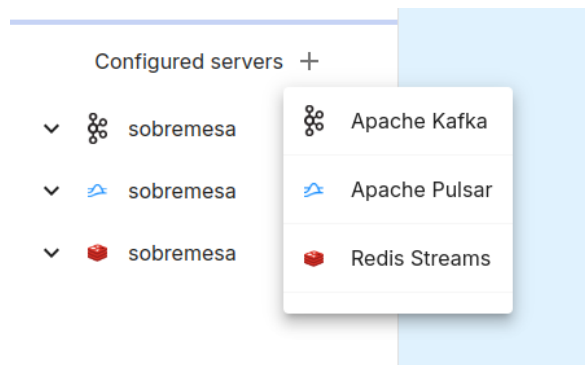


Figura D.7: Drop-down menu to add a new streaming server

D.6. Simulations

In this section, it will be explained how to select a simulation (Subsection D.6.1), and import (Subsection D.6.2) or export (Subsection D.6.3) a simulation.

D.6.1. Select a simulation

To select a simulation to play, go to **Simulation/Simulation list** on menu and click, as shown in Figure D.16.

After clicking, you should be in a new page with the appearance shown on Figure D.17.

If you have never imported a simulation in the system, it will appear instead a text saying there are not stored simulations. Go to Subsection D.6.2 for information on importing simulations.

To select a simulation, click on the unfold icon at the left side of the desired simulation, and a card content will be displayed, like in Figure D.18. Now, click on

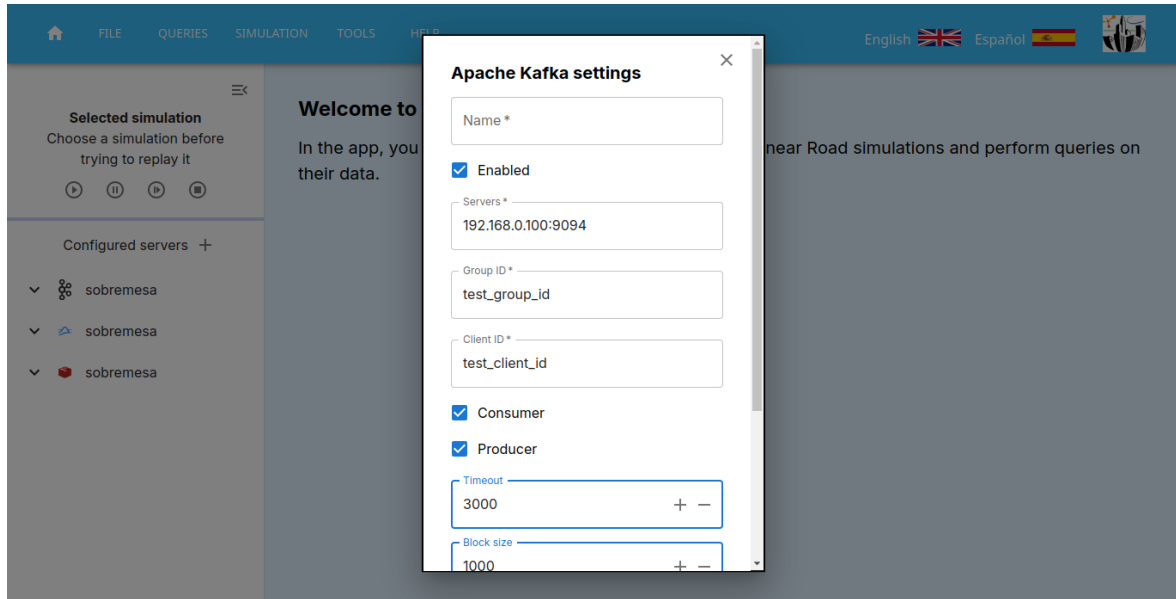


Figura D.8: First part of the Apache Kafka settings dialog

Select simulation button. From now on, at any page of the application, when you click the play button on the lateral panel, this simulation will be started. In fact, you a blue outline will appear around the selected simulation (as shown in Figure D.19), and the name of the simulation will be displayed on the lateral panel, as shown in Figure D.20.

It is also possible to modify basic information about the simulation, by clicking on *Edit* button, and filling the dialog, as shown in Figure D.21; as well as delete the simulation from the system.

D.6.2. Import simulations

To import a simulation in the system, go to **Simulations/Import** on menu, as shown in Figure D.22.

The outcome of this action will be the one shown in Figure D.23. By clicking on the drop-down, you can choose the type of simulation you want to import, among the three scenarios considered in this work.

If you choose the first one, you will see the content of the page shown in Figure D.24. If any doubt appears about the format of the data files, unfold the darker blue section to see further detail, as shown in Figure D.25. Click on *Select file* (although in the figures it says *Seleccionar archivo* in Spanish, as the system used to take the captures is in Spanish) and a native pop-up will be displayed for you to choose your file. Once this is done, click on *Upload files* blue bottom and the simulation will be imported into the system. You will also be redirected to the list of simulations.

Group ID *
test_group_id

Client ID *
test_client_id

☒ Consumer

☒ Producer

Timeout
3000 + -

Block size
1000 + -

Number of partitions
3 + -

Replication factor
1 + -

OK CANCEL

Figura D.9: Second part of the Apache Kafka settings dialog

If you choose to import a Linear Road simulation, the page will be similar to Figure D.26, whilst if a parking lot simulation is to be imported, the page will be similar to Figure D.27.

D.6.3. Export simulations

To export a simulation in the system, go to **Simulations/Export** on menu, as shown in Figure D.28. In a similar way to how to select a simulation to start it (Subsection D.6.1), unfold the card of the simulation and a blue outline will appear. Then, click *Download file* button and a native pop-up will appear asking where you want to store it.

D.7. Continuous queries

To view the results of the continuous queries running at the moment on the system, click on **Queries/View active queries** on menu, as shown in Figure D.29.

You will see something similar Figure D.30. To see the results of any of the simulations, as in other parts of the application, click the right placed button at the

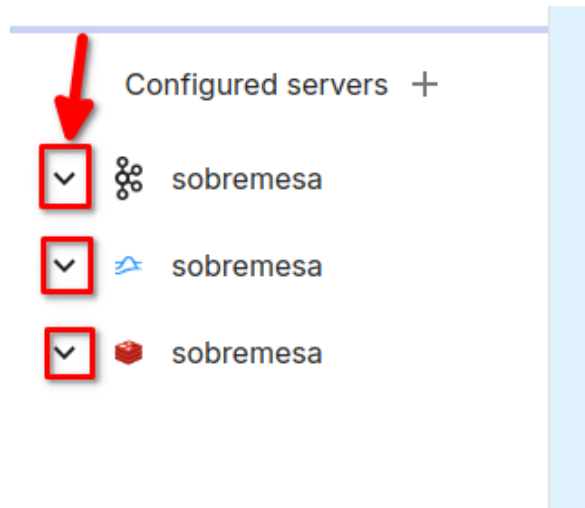


Figura D.10: Location of the unfolding buttons on streaming servers panel section

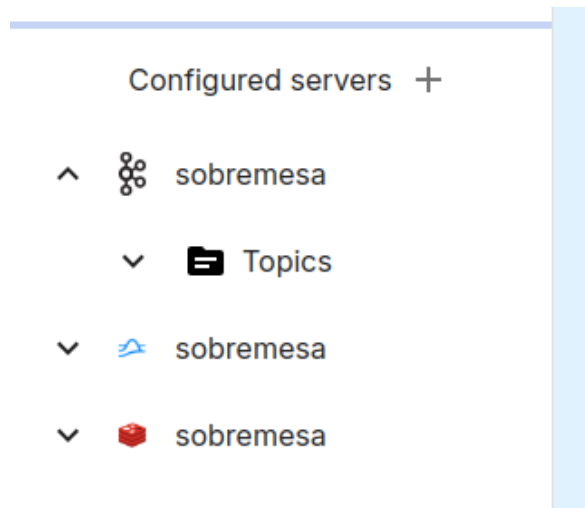


Figura D.11: Topics folder location after unfold

side and a card will be displayed, as in Figure D.31. Then, click the *View results* button and the visualization will be loaded.

There are several types of visualization: maps, as shown in Figure D.32, Figure D.33 or Figure D.34, tables as shown in D.35

D.8. Logging

You can also view the logs of the system, by going to **Tools/Log console**, as shown in Figure D.37. The layout will be something similar to the one shown in Figure D.38, with a red background for ERROR level logs, a yellow background for WARN level logs, and alternation of white and grey background for INFO level logs, in order to make them easier to read.

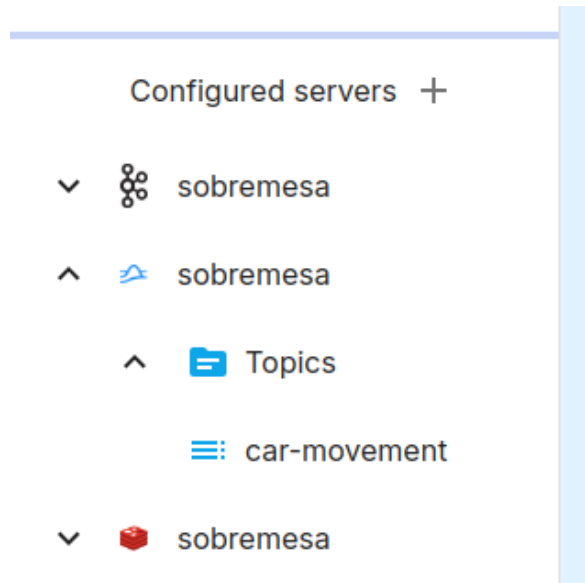


Figura D.12: Topic list location after unfold

D.9. Configuration

If you want to change any setting, go to **Tools/Settings** on the menu, as shown in Figure D.39. The content of the page will change to match the one shown in Figure D.40. The text field that appears under *Visualization* tag controls how many decimals will be shown from any real number of the system.

If you want to change any external systems settings, click on the arrow next to it and a new page will appear, as the one shown in Figure D.41. From there, you can edit the configuration (by unfolding a hidden card with the icon on the left), delete it, or create a new one.

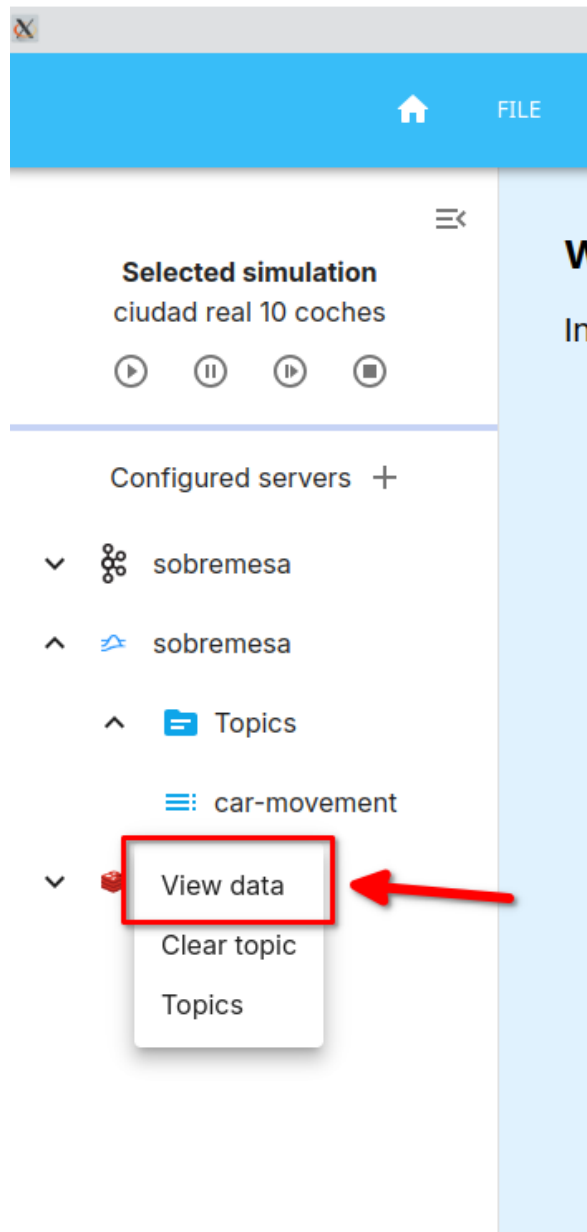


Figura D.13: Location of *View data* button

Topics found:

LINEAR-ROAD CAR-MOVEMENT CAR-PARKING CAR-MOVEMENT

Data from car-movement [EXPORT DATA IN CSV](#)

simulation	car	iteration	x	y
15	6	286	-3.9337066	38.983952099999996
15	5	286	-3.9345521000000003	38.983553099999995
15	3	286	-3.9234375000000004	38.992284749999996
15	4	286	-3.9236400910790192	38.98271677690332
15	2	286	-3.9033127261483775	38.990748388462926
15	0	287	-3.9312373999999997	38.99084895
15	4	287	-3.923521609376096	38.98275237322312
15	3	287	-3.9234375000000004	38.992284749999996
15	9	287	-3.9196155651258913	38.991732261822165
15	7	287	-3.9373202128050675	38.98521718486637

Rows per page: 10 1-10 of 305

Figura D.14: Page for visualizing topic data on streaming systems

iteration	x	y
286		38.98395209
286		38.98355309
286		38.99228474
286		38.98271677
286	-3.9033127261483775	38.99074838

Figura D.15: Graphical aspect of the drop-down menu to sort and filter row data at grids

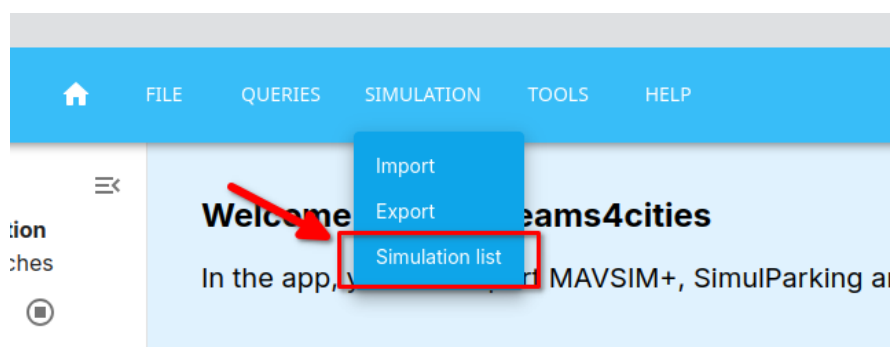


Figura D.16: Position of *Simulation list* button on the menu

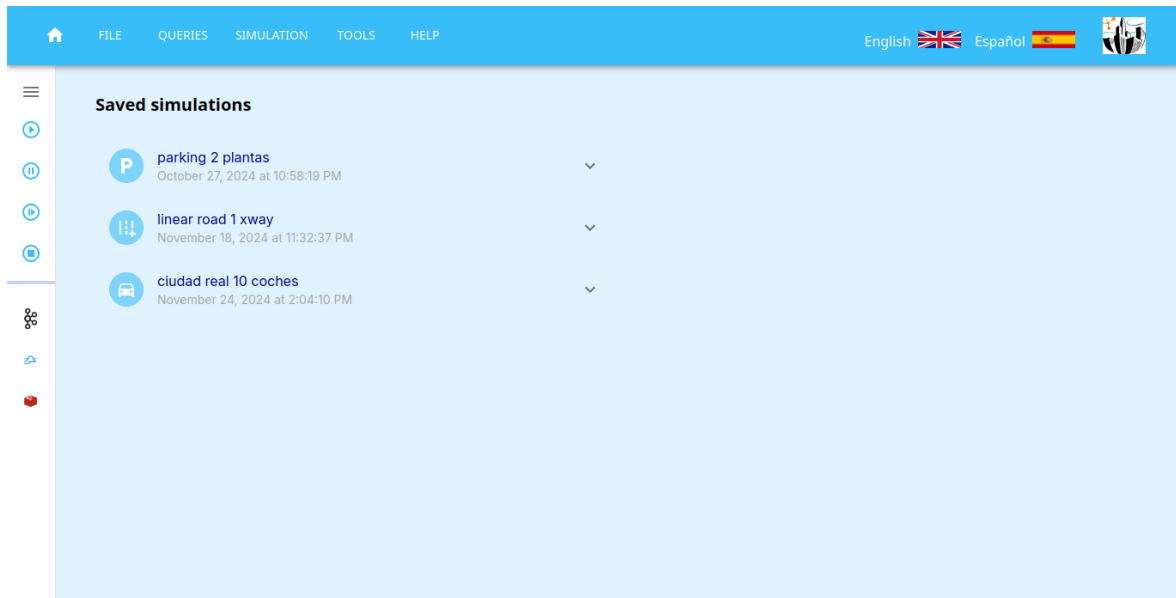


Figura D.17: Page showing a list of the saved simulations

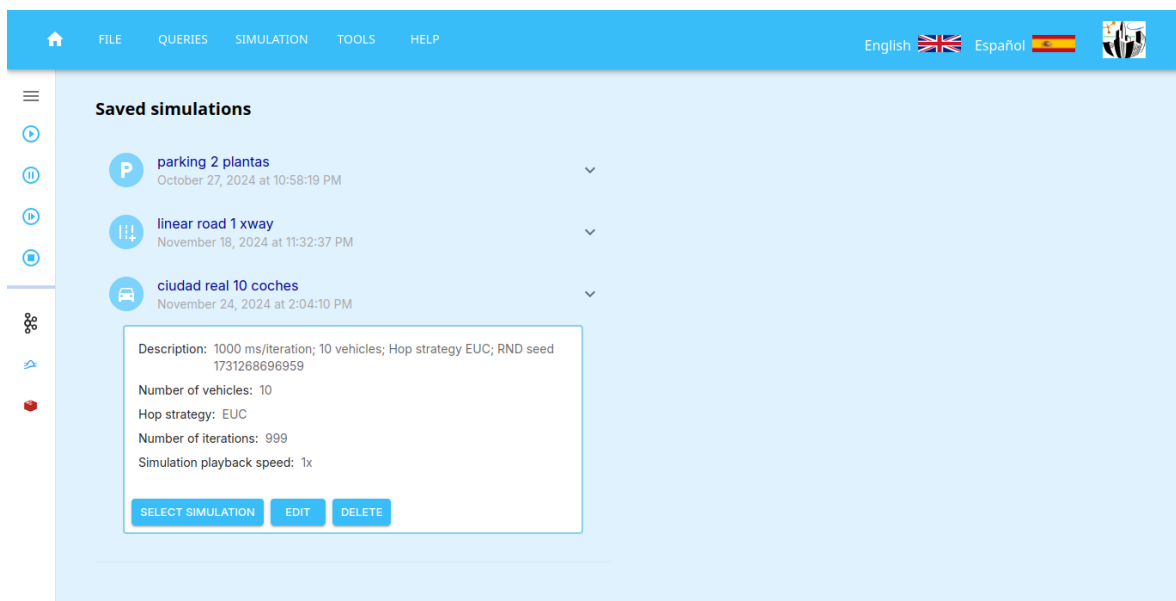


Figura D.18: Information about a simulation

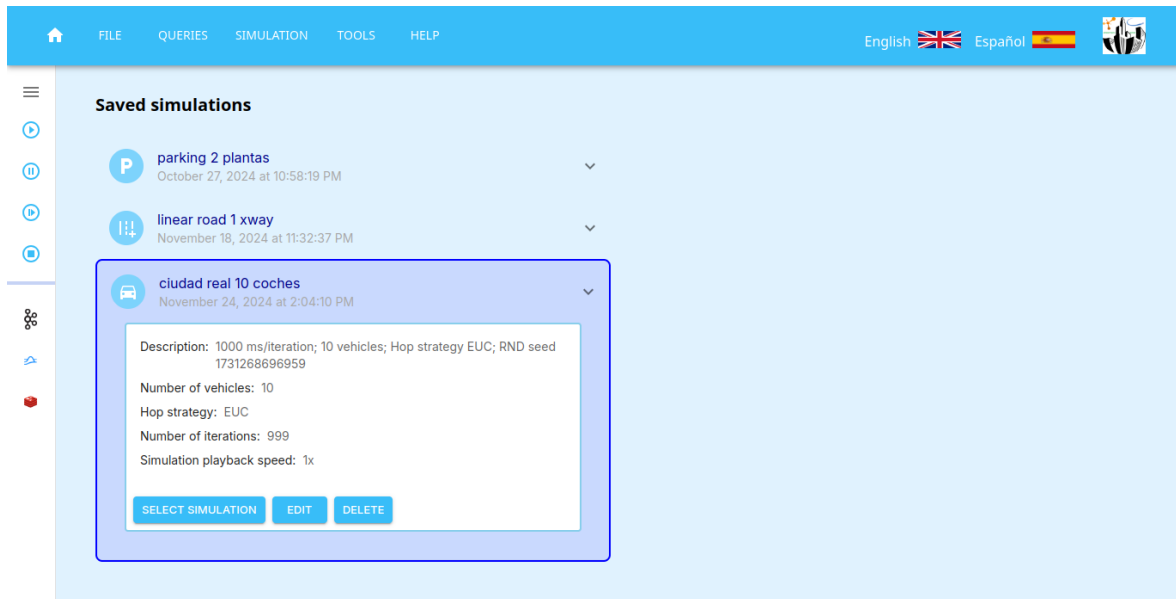


Figura D.19: Interface feedback on selected simulation

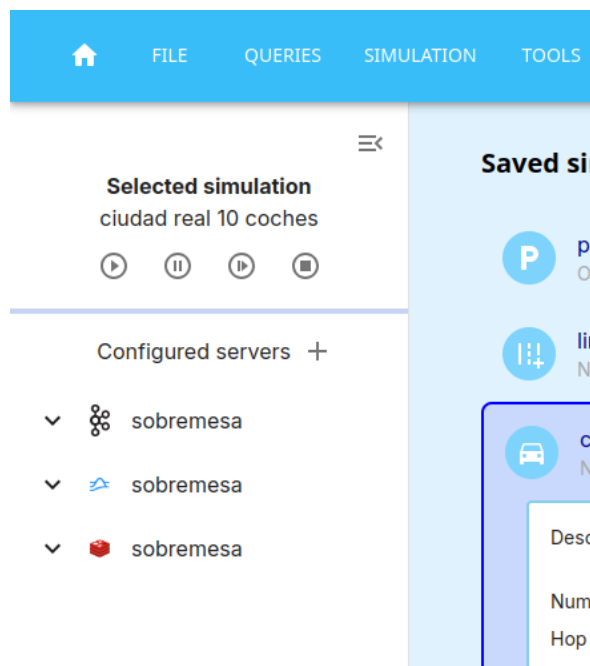
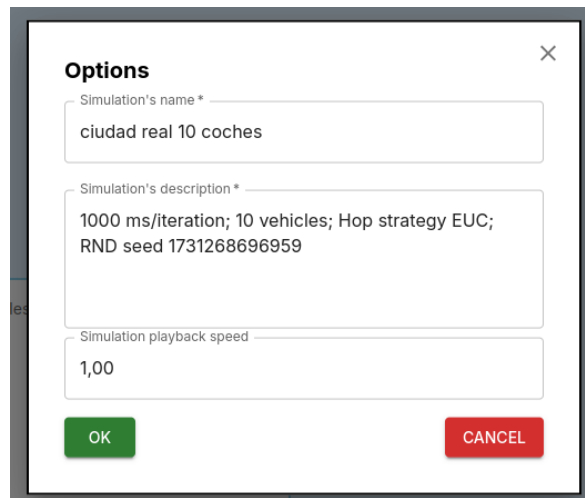


Figura D.20: Selected simulation name set on lateral panel



Options

Simulation's name *

ciudad real 10 coches

Simulation's description *

1000 ms/iteration; 10 vehicles; Hop strategy EUC;
RND seed 1731268696959

Simulation playback speed

1,00

OK CANCEL

Figura D.21: Edit simulation dialog

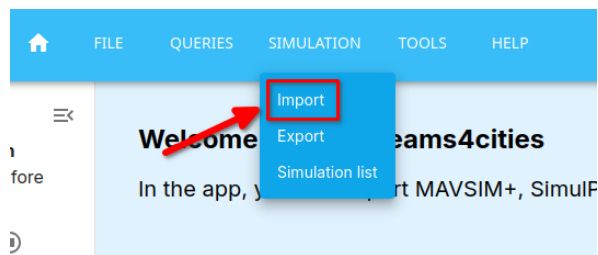
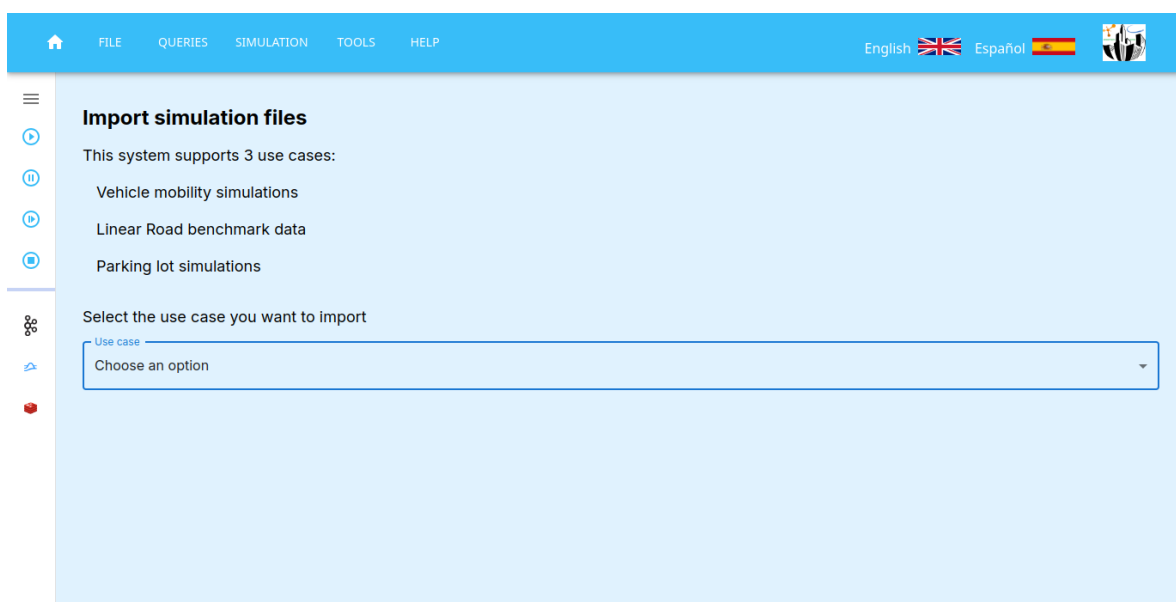


Figura D.22: Position of the option to import simulations on menu



The image shows the 'Import simulation files' page of the software. The page has a blue header bar with the same menu items as Figure D.22. Below the header, the page title is 'Import simulation files'. The main content area is light blue and contains the following text:

This system supports 3 use cases:

- Vehicle mobility simulations
- Linear Road benchmark data
- Parking lot simulations

Select the use case you want to import

Use case

Choose an option

Figura D.23: Initial import simulation page

FILE

QUERIES

SIMULATION

TOOLS

HELP

English

Español

Import simulation files

This system supports 3 use cases:

Vehicle mobility simulations

Linear Road benchmark data

Parking lot simulations

Select the use case you want to import

Use case

Vehicle mobility simulations

Accepted file types:

MAVSIM+ simulation trace file (.rpl)

GPS trace file (.gpx)

Comma separated value file (*.csv)

If the *.csv format is chosen, the file must include a header with the following fields:

Please select the file you want to import

Seleccionar archivo

Ningún archivo seleccionado

If you want to compare the processed travel distance result with the calculation performed by MAVSIM+, you must also import the file "MAVSIM_individual_vehicle_stats.csv", which you will find in the application's log directory.

Seleccionar archivo

Ningún archivo seleccionado

UPLOAD FILES

Figura D.24: Import vehicle mobility simulation page

FILE

QUERIES

SIMULATION

TOOLS

HELP

English

Español

Import simulation files

This system supports 3 use cases:

Vehicle mobility simulations

Linear Road benchmark data

Parking lot simulations

Select the use case you want to import

Use case

Vehicle mobility simulations

Accepted file types:

MAVSIM+ simulation trace file (.rpl)

GPS trace file (.gpx)

Comma separated value file (*.csv)

If the *.csv format is chosen, the file must include a header with the following fields:

Field	Description
car	vehicle identifier in the simulation
iteration	simulation iteration number in which the movement was recorded
x	vehicle's x-coordinate position
y	vehicle's y-coordinate position

Please select the file you want to import

Seleccionar archivo

Ningún archivo seleccionado

Figura D.25: Import vehicle mobility simulation page with data format details

95

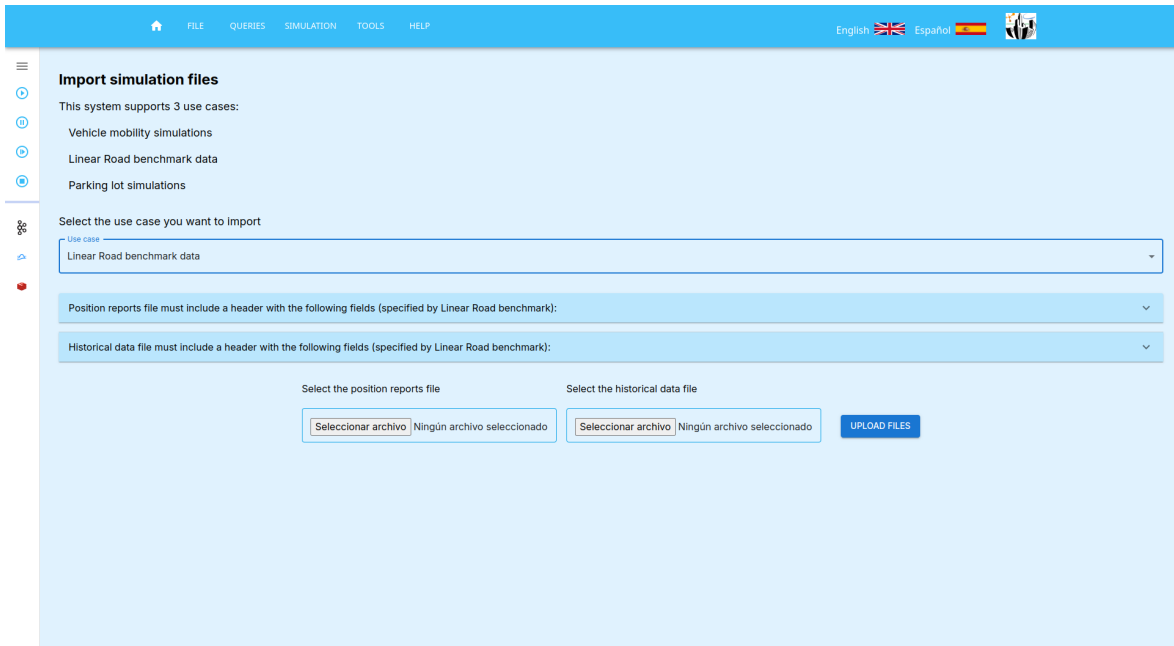


Figura D.26: Import Linear Road simulation page

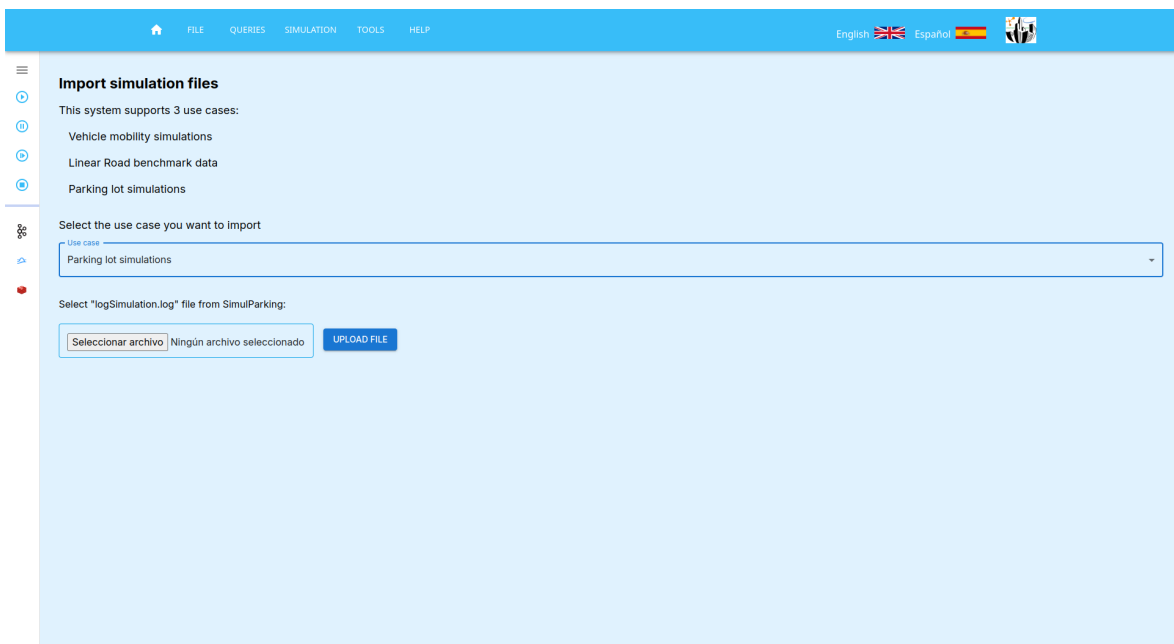


Figura D.27: Import parking lot simulation page

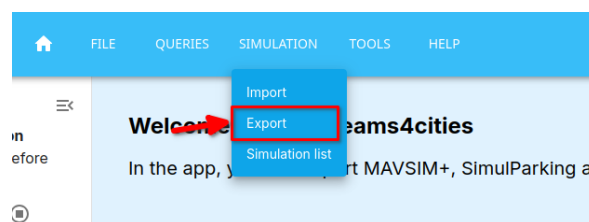


Figura D.28: Position of the option to export simulations on menu

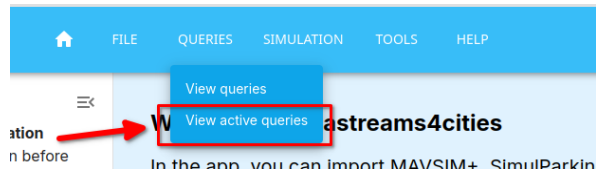


Figura D.29: Position of the button to view continuous queries results on menu

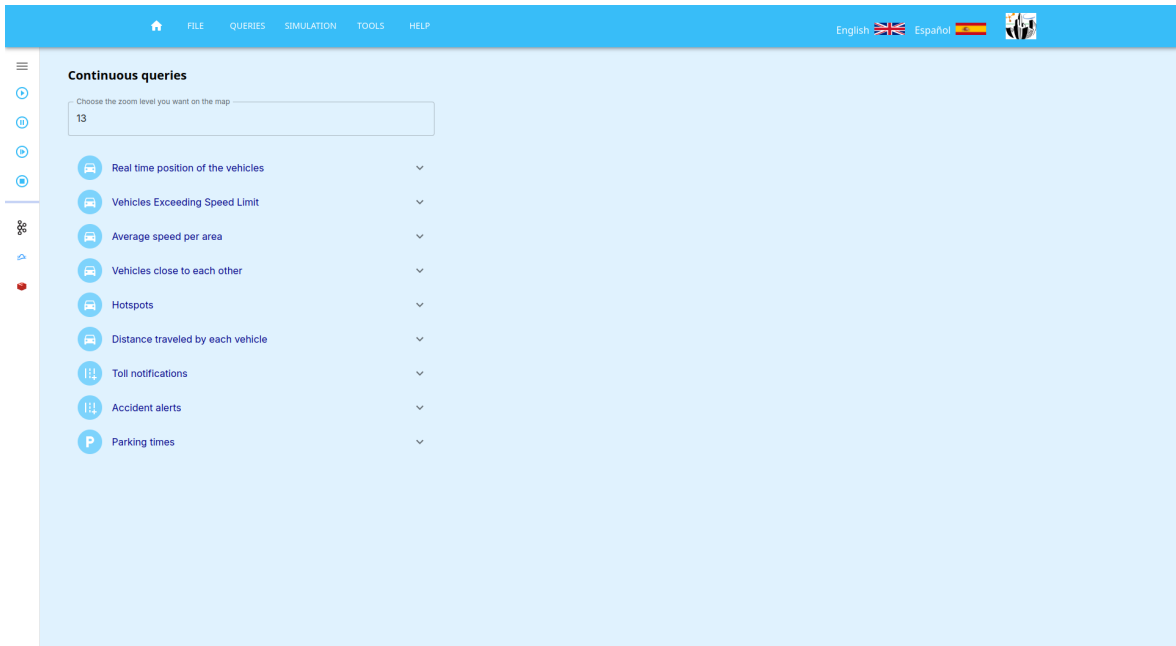


Figura D.30: View continuous queries page

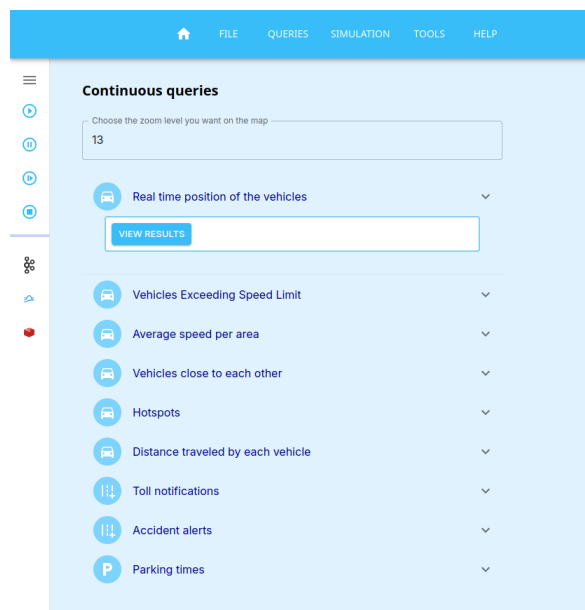


Figura D.31: How to select a continuous query to view its results

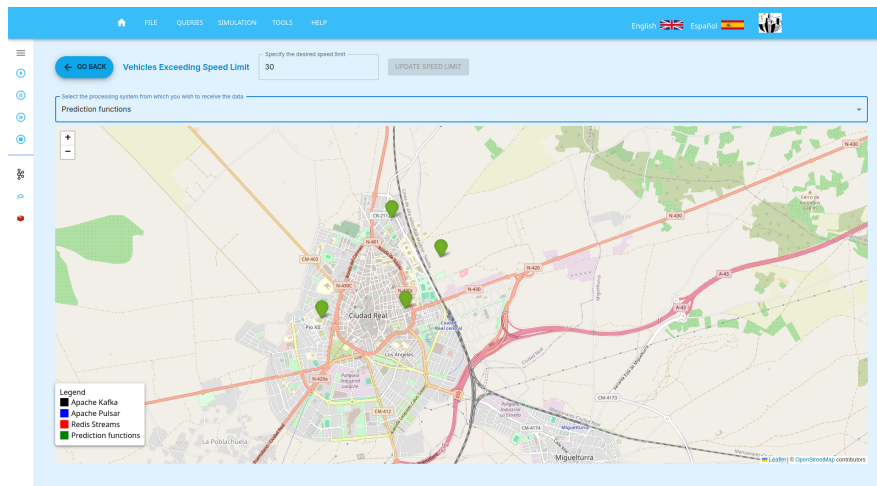


Figura D.32: Speed limit query visualization

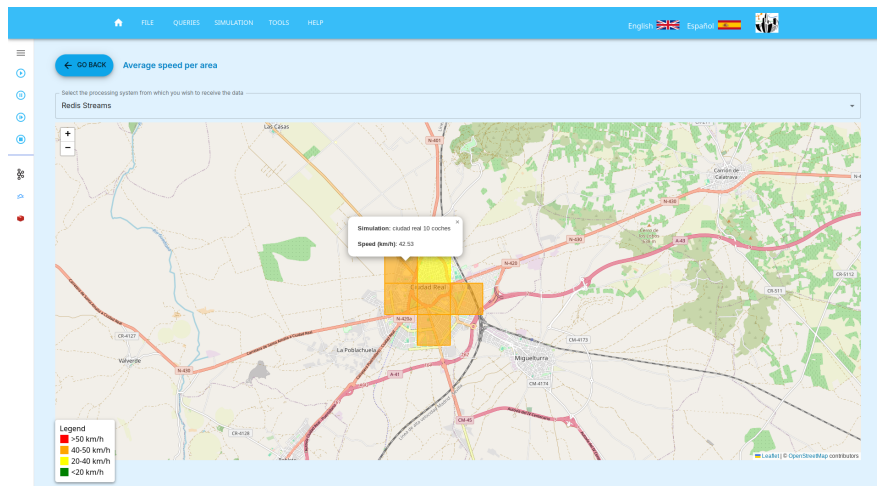


Figura D.33: Average speed per km^2 query visualization

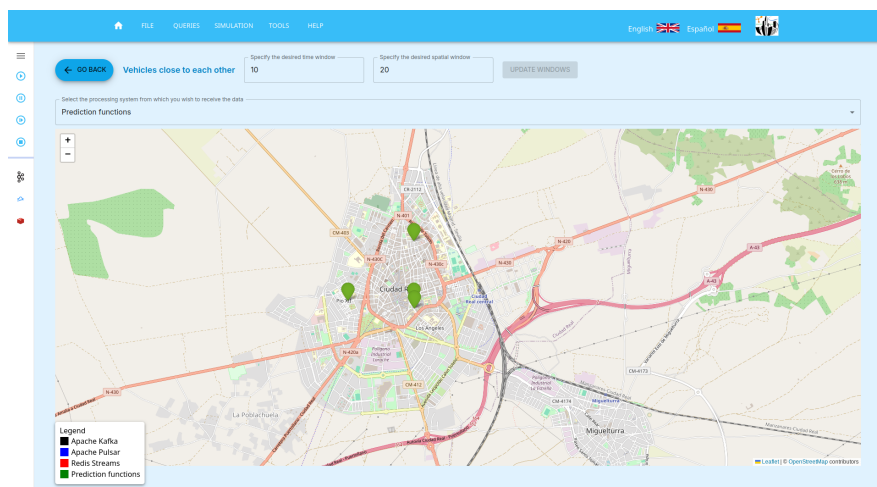


Figura D.34: Vehicles close to each other query visualization

Simulation	Distance (m)	Distance (m) calculated by MAVSIM+	Vehicle
biarritz-75-coches_2024...	776.49	14583.33	0
biarritz-75-coches_2024...	866.05	14722.22	1
biarritz-75-coches_2024...	756.7	3075.67	2
biarritz-75-coches_2024...	711.87	12500	3
biarritz-75-coches_2024...	783.71	3344.93	4

Figura D.35: Distance traveled by vehicle query visualization

Simulation	Vehicle	Parking time (s)	Parking spot
parking 2 plantas	0	1.8	0
parking 2 plantas	2	4.05	7
parking 2 plantas	1	3.6	5
parking 2 plantas	3	4.297	1
parking 2 plantas	4	4.5	6

Figura D.36: Parking times query visualization

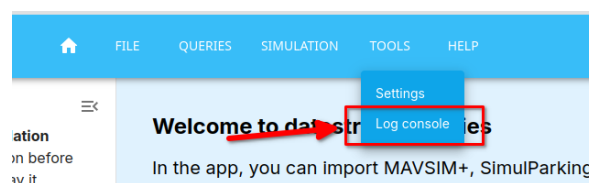


Figura D.37: Position of logging access on menu

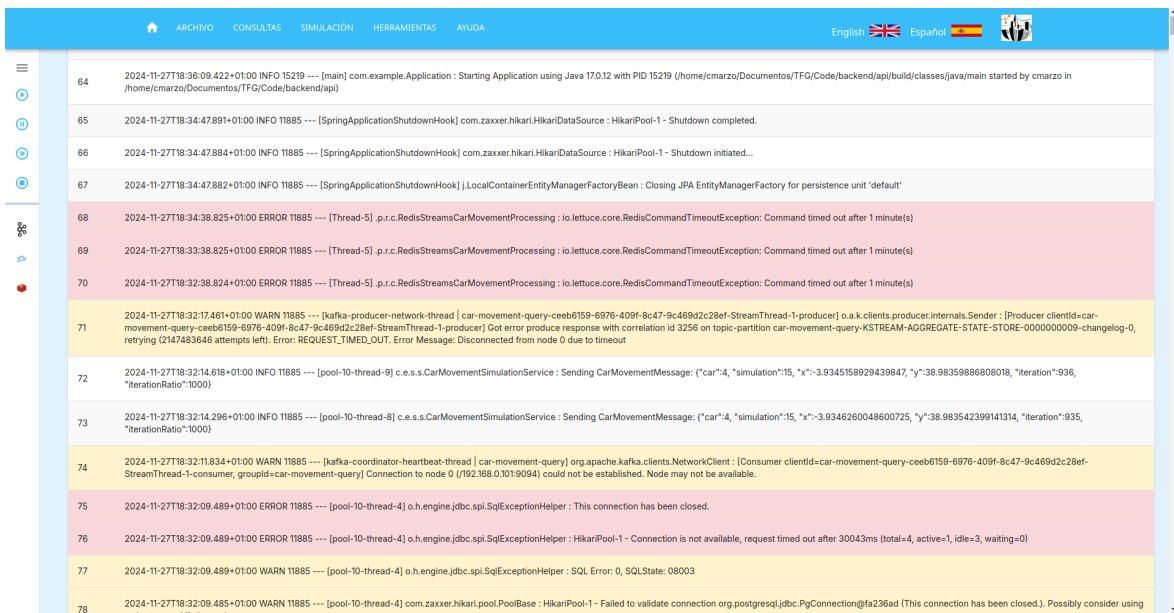


Figura D.38: Logging visualization page

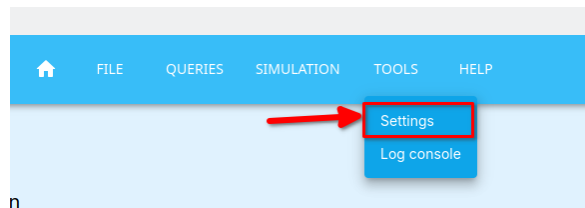


Figura D.39: Position of the *Settings* button on the menu

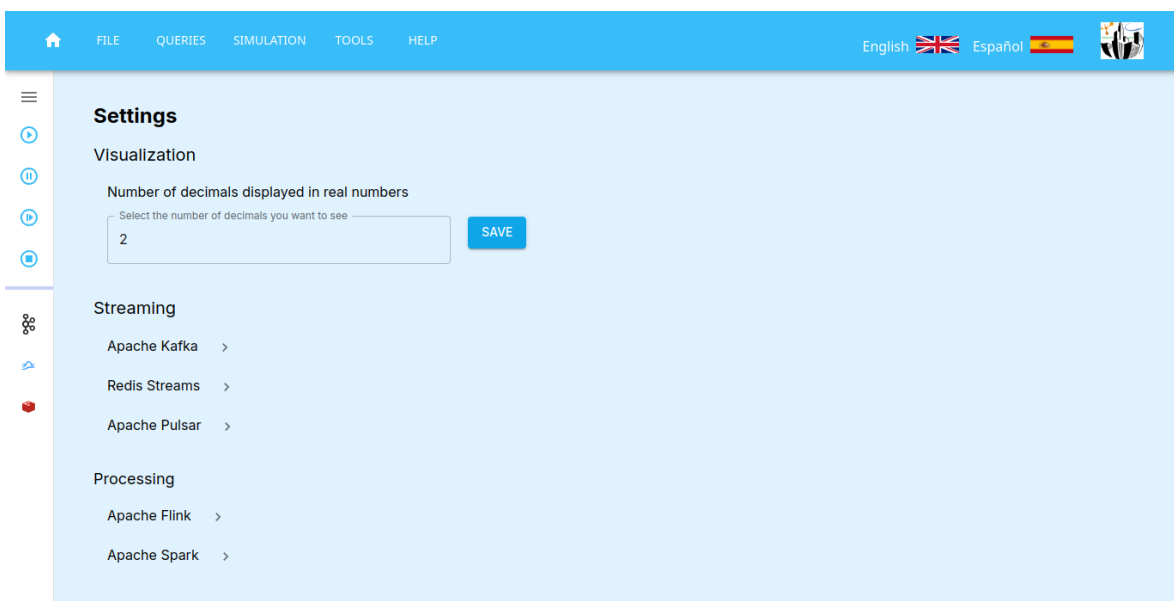


Figura D.40: Settings page

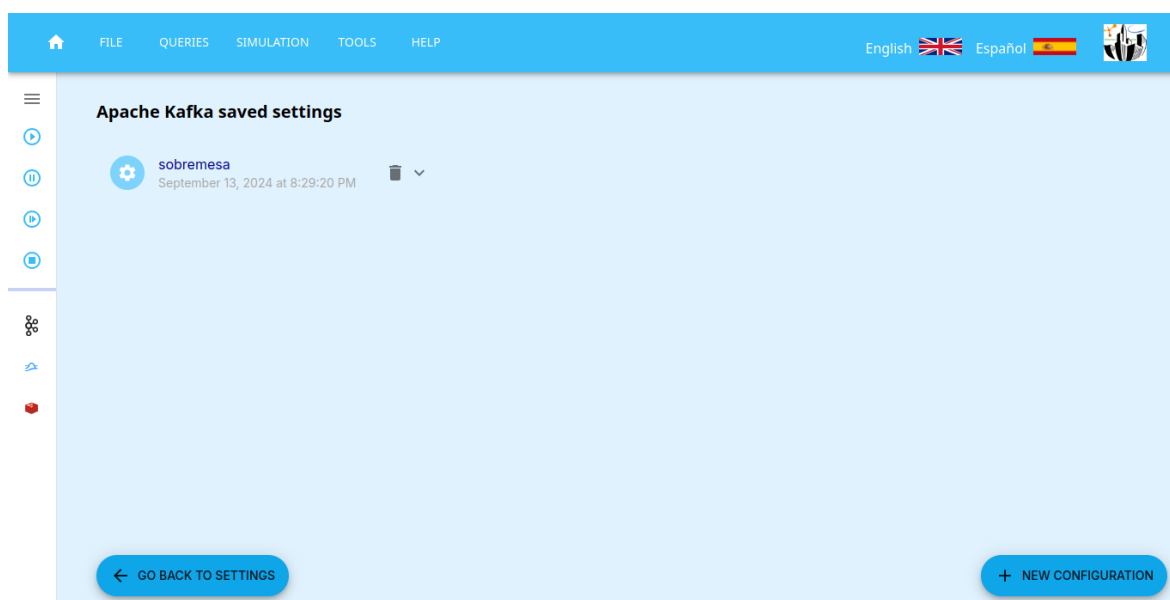


Figura D.41: Apache Kafka settings page