UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Análisis y predicción de datos obtenidos del funcionamiento de un AGV

Presentado por Gonzalo Burgos de la Hera
en Universidad de Burgos — 19 de junio de 2023
Tutor: Bruno Baruque Zanón y Jesús Enrique Sierra García

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Gonzalo Burgos de la Hera, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 19 de junio de 2023

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

Los AGV, Autonomous Guided Vehicles por sus siglas en inglés, son complejos sistemas robóticos, capaces de moverse en un entorno concreto, cuyo uso es transportar cargas pesadas en fábricas o almacenes, y que están diseñados para mejorar la eficiencia y la productividad en la logística y el transporte de materiales. Debido a sus ventajas en seguridad, flexibilidad y velocidad, esta tecnología se está convirtiendo cada vez más importante [8].

Aunque estos sistemas pueden mejorar la productividad, desajustes en su configuración u otros errores operacionales pueden producir una reducción de su rendimiento, y, en casos extremos, causar una detención de la línea de producción. Por este motivo, es necesario extraer información de los sistemas en marcha para analizar el rendimiento de las máquinas y las aplicaciones logísticas. Esta información puede usarse para predecir comportamientos futuros del sistema, realizar mantenimiento predictivo y proveer retroalimentación con el fin de diseñar mejoras continuas de las máquinas. Estas predicciones pueden ser conseguidas con el uso de algoritmos de análisis de series temporales, que permitan anticipar futuras condiciones del sistema. [4]

Algunos de estos datos obtenidos del sistema tienen una baja frecuencia de actualización, como puede ser la temperatura y voltaje de la batería, pero otros cambian cada pocos milisegundos, como la corriente eléctrica, la velocidad, la posición del vehículo, errores y estado, etc. Toda esta información proveída por el AGV debe estar relacionada con el tiempo en el que fue generada, por lo que puede ser agrupada en series temporales [6]. Cualquier tipo de base de datos puede usarse para almacenar esta información generada por los AGV, sin embargo, ya que se trata de series temporales, es preferible utilizar bases de datos para series temporales para optimizar el rendimiento del sistema.

En este trabajo, por tanto, se propone un sistema capaz de almacenar la información recibida por dichos AGV, y, posteriormente, realizar las predicciones pertinentes sobre dicha información. Se propone también la creación de un simulador capaz de generar datos en caso de no tener ningún AGV disponible. Para ello, será necesario realizar una comparación de diferentes sistemas gestores de bases de datos, así como de los modelos de predicción que se consideren usar, para conseguir con ello el sistema más óptimo posible.

Objetivos del proyecto

Como con cualquier proyecto relacionado con temas industriales o de producción, los principales objetivos de este proyecto son claros: mejorar la eficiencia y reducir costes. Todo esto es posible gracias a la correcta realización de predicciones que permitan cosas como, por ejemplo, mantenimiento predictivo.

Conceptos teóricos

Como se ha mencionado en apartados anteriores, los datos recibidos por el AGV se agrupan en series temporales. Conviene por tanto explicar que son las series temporales, así como los modelos que se utilizarán para intentar predecirlas.

3.1. Series temporales

Una serie temporal es una colección de observaciones obtenidas mediante mediciones repetidas a lo largo del tiempo [13].

Normalmente, las series temporales presentan patrones que pueden utilizarse para realizar predicciones. Estos patrones son:

- **Tendencia**. La tendencia existe cuando hay un incremento o decremento del valor medido a largo plazo. Esta tendencia no tiene por qué ser lineal.
- **Estacionalidad**. El patrón de estacionalidad se presenta cuando una serie temporal se ve afectada por factores estacionales, como puede ser el día de la semana. Siempre tiene una frecuencia fija y conocida.
- **Ciclos**. Un ciclo ocurre cuando los datos muestran incrementos o decrementos a una frecuencia no fija.

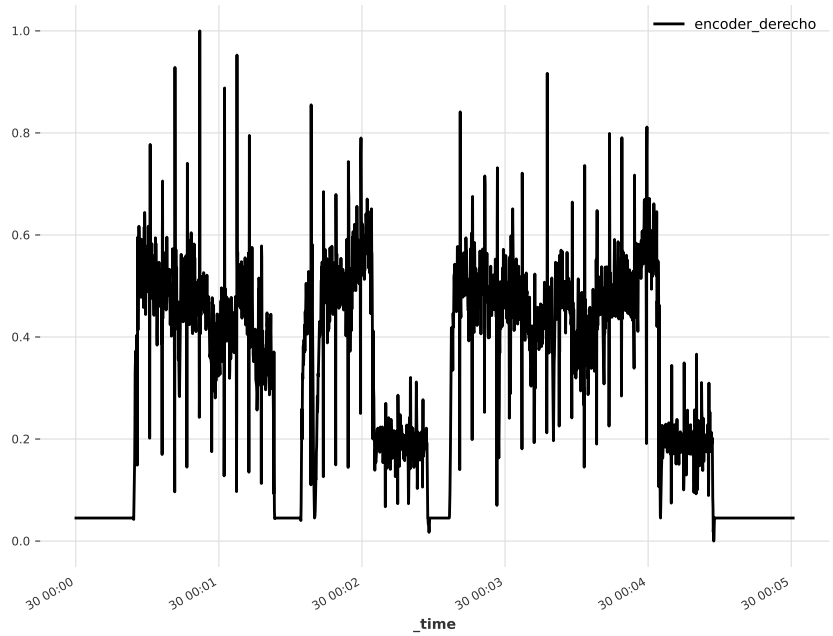


Figura 3.1: Ejemplo de serie temporal

Existen distintos tipos de clasificaciones para las series temporales, según varios puntos de vista [14], de las cuales destacan:

- **Continuas o discretas**. Las series temporales continuas son aquellas en las que la información se obtiene, valga la redundancia, de forma continua, normalmente por un dispositivo analógico, como podría ser los datos recibidos de un sismógrafo. Por otro lado, las series temporales discretas son aquellas en las que la información se obtiene en intervalos de tiempo concretos. Estos intervalos pueden ser equidistantes, o bien ser irregulares. Normalmente, las series temporales medidas por medios digitales son discretas. En nuestro caso, las series temporales enviadas por el AGV son discretas espaciadas en intervalos irregulares, pues el AGV manda dicha información cada varios milisegundos, de manera irregular.

- **Univariantes o multivariantes**. Aquellas series temporales que tengan solo una observación por cada momento del tiempo son series univariantes. Por contra, aquellas en las que se obtengan de manera simultánea mediciones de dos o más fenómenos son multivariantes. Esto

será importante a la hora de escoger que modelo utilizar para realizar predicciones, pues hay modelos que solo soportan series temporales univariantes.

- **Estacionarias o no estacionarias**. Una serie estacionaria [11] es aquella en la que sus propiedades no dependen del momento en el que se observan. Por ello, aquellas que presenten tendencias o estacionalidad no son estacionarias. Por otra parte, una serie de ruido blanco es estacionaria: no importa cuándo se observe, debería tener el mismo aspecto en cualquier momento. De manera general, las series estacionarias no tendrán patrones predecibles en el largo plazo, por lo que conviene convertir las series estacionarias en no estacionarias.

3.2. Predicción de series temporales

La predicción de series temporales es una actividad muy importante en muchos sectores: predicción de datos financieros, predicciones del clima, etc. Debido a esto, existe una gran cantidad de modelos para realizar dichas predicciones. Hay que tener en cuenta sin embargo que predecir datos futuros es una tarea especialmente complicada, y no siempre se obtiene una gran precisión.

Los siguientes modelos se tendrán en cuenta en este trabajo:

- **ARIMA.** Este modelo es una combinación de otros tres modelos:
- **VARIMA.**
- **NHitS.**
- **Transformer Model.**

Técnicas y herramientas

4.1. Metodologías

Las siguientes metodologías han sido utilizadas:

- **SCRUM**: es un marco de trabajo cuyas características principales son: desarrollo incremental en lugar de una ejecución completa, basar la calidad en el conocimiento de los integrantes del equipo en vez de en la calidad de los procesos y solapar las fases del proyecto.

  En SCRUM, el trabajo se divide en sprints, periodos de tiempo de entre 1 y 4 semanas. Al principio de cada sprint se planifica el trabajo a desarrollar, y se mantienen reuniones diarias que sirven para mantener la comunicación entre desarrolladores. Al final del sprint se tiene una última reunión en la que se analiza el trabajo realizado.

  En nuestro caso, los sprints han sido de entre 1 y 2 semanas, y no se han tenido las reuniones diarias.

- **Pomodoro** es un método diseñado para mejorar la productividad mediante una correcta gestión del tiempo. Se establece un tiempo de trabajo con una duración de entre 20 y 30 minutos, y un tiempo de descanso de 5 minutos. Después de cuatro ciclos de trabajo/descanso, se realiza un descanso más largo, de unos 15 a 30 minutos.

  Este método ha sido utilizado especialmente a la hora de escribir esta memoria y los anexos.

4.2. Herramientas de desarrollo

Desarrollo

El proyecto ha sido desarrollado íntegramente en el lenguaje de programación Python en su versión 3.10.6.

Además, todos los módulos que más tarde se explican han sido desarrollados como contenedores de Docker. Para ello se han utilizado:

- Docker, en su versión 24.0.2. Docker permite el despliegue de aplicaciones dentro de contenedores, similares a máquinas virtuales, de forma que cada contenedor está aislado del resto del sistema. Docker permite también la ejecución de dichas aplicaciones de forma independiente al sistema operativo, por lo que permite una gran compatibilidad.

- Docker-compose versión 2.16.0. Esta herramienta sirve para definir y ejecutar aplicaciones multi-contenedor.

Sistema Gestor de Base de Datos

El sistema gestor de bases de datos escogido ha sido InfluxDB. En la próxima sección se dan los motivos de su elección, así como una comparación con otros sistemas.

InfluxDB es un sistema gestor de bases de datos de series temporales. En InfluxDB, los datos se guardan en series. Una serie es un conjunto de puntos que comparten:

- Medida (Measurement): equivalente en SQL a una tabla.

- Conjunto de etiquetas (tag set): equivalente en SQL a una columna indexada. Solo pueden ser cadenas de texto. Sirven para almacenar metadatos.

- Conjunto de valores (field set): equivalente en SQL a una columna sin indexar. Guardan los datos.

Por último, estas series se guardan en “Buckets”, que son el equivalente en SQL a una base de datos.

Predicción

Como librería utilizada para la predicción de las series temporales se ha utilizado Darts [10]. Se consideraron otras alternativas, como Loud ML, herramienta desarrollada específicamente para InfluxDB, y Facebook Prophet. La primera fue descartada debido a que el proyecto está abandonado, y la segunda fue descartada debido a que Darts incluye muchos más modelos, incluido el mismo Prophet.

Se ha utilizado también la libreria Optuna para la optimización de los hiperparámetros de los modelos de predicción.

4.3. Entorno de desarrollo

Todo el trabajo ha sido realizado con el editor de texto de Microsoft Visual Studio Code. Esto ha sido así por varios motivos, principalmente mi comodidad con él, pues es el editor de texto que estoy acostumbrado a usar, y la gran cantidad de extensiones existentes que permitan una mejor experiencia de usuario.

En cuanto a las extensiones utilizadas, caben destacar:

- Docker: utilizada para un manejo más simple de los contenedores de Docker.
- Python: extensión esencial para el desarrollo en Python, pues ofrece características como un Debugger, resaltado de la sintaxis, IntelliSense (autocompletado), etc.
- Todo Tree: utilizada para marcar elementos no terminados, de forma que luego sean fáciles de encontrar.
- LaTeX Workshop: permite compilar de forma automática al guardar los archivos de LaTeX, autocompletado, etc.
- LTeX - LanguageTool grammar/spell checkingLTex: como su nombre en inglés indica, esta extensión analiza errores gramaticales en los archivos de LaTeX.

4.4. Otras herramientas

Control de versiones

El proyecto se aloja en la plataforma GitHub, y la herramienta utilizada para el control de versiones es git. Inicialmente se empezó a usar ZenHub para llevar el control de los sprints. Sin embargo, debido a un problema de licencias a mitad del desarrollo se dejó de utilizar en favor de simplemente usar el apartado de Issues del repositorio para dicha labor.

Documentos

La memoria y los anexos han sido desarrollados en LaTeX, utilizando la distribución TeX Live[2] para la compilación de dichos documentos.

Aspectos relevantes del desarrollo del proyecto

El desarrollo de este proyecto puede decirse que ha estado dividido principalmente en dos partes. La primera fase estuvo relacionada con el diseño de la arquitectura, así como la elección del sistema gestor de bases de datos. En la segunda fase, se diseñó todo lo relativo a la predicción de datos y a la elección de las herramientas necesarias para dicha tarea.

5.1. Primera fase

Como ha sido mencionado en el párrafo anterior, en la primera fase se llevó a cabo el diseño de la arquitectura del sistema, así como el desarrollo de los servicios que forman dicho sistema. Además, se ha realizado una comparativa de diferentes sistemas gestores de bases de datos con el fin de elegir el que mejor se adapte a nuestros requisitos.

Diseño de la arquitectura

La arquitectura del sistema estudiado se representa en la figura 5.1. El sistema está formado por los siguientes sistemas software:

AGV Coordinator Es un servicio encargado de recibir la información enviada por los AGV a través de una conexión 5G/Wifi. Esta información está codificada como una cadena de bytes, que es decodificada y transformada a formato JSON. Estos mensajes son enviados al servicio "Receiver". Este servicio no ha sido desarrollado como parte de este trabajo.

Simulator Es, como su nombre en inglés indica, un servicio encargado de simular un AGV en caso de no disponer de AGV reales y sea necesario realizar pruebas.

Receiver Este servicio recibe mensajes a través del protocolo UDP bien del "AGV Coordinator" o bien del simulador, y se encarga de insertar dichos datos en la base de datos.

Database Como su nombre indica, este servicio será la base de datos encargada de almacenar todos los datos recibidos.

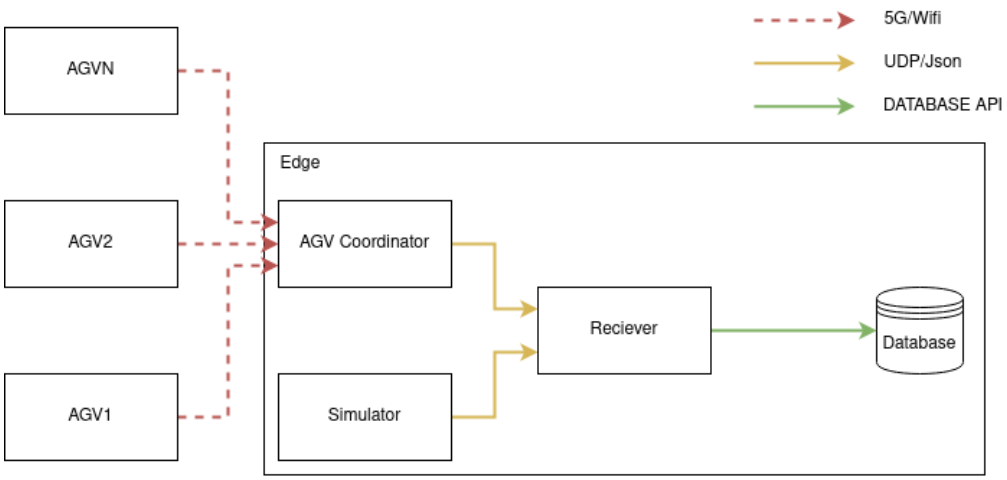


Figura 5.1: Diseño de la arquitectura

Comparativa de Sistemas Gestores de Bases de Datos

Lo primero a tener en cuenta a la hora de elegir un gestor de bases de datos es cuáles son nuestros requisitos relacionados con el almacenamiento de los datos. Como hemos comentado anteriormente, el tiempo juega un papel importante en los mensajes que enviamos, pues es importante saber cuando el valor de una variable, como puede ser el nivel de la batería, fue reportado. Algunas de estas variables son enviadas cada 10 o 20 milisegundos, por lo que una alta precisión temporal es necesaria.

Requisito	Descripción
RF1:	El sistema gestor de bases de datos debe soportar entradas con timestamp
RF2:	Los datos tienen que poder insertarse en cualquier momento
RF3:	El timestamp de las entradas de la base de datos ha de corresponder con el momento en el que dichos datos fueron creados en el AGV
RF4:	El sistema gestor de bases de datos debe poder soportar entradas con una precisión temporal en el rango de los milisegundos
RNF1:	El sistema gestor de base de datos debe ser de código abierto
RNF2:	EL sistema gestor de base de datos debe tener un buen soporte de Linux y Python

Tabla 5.1: Requisitos Funcionales (RF) y no Funcionales (RNF) de la base de datos

Metodología de la comparación

Sistemas de gestión de bases de datos bajo estudio Lo primero a tener en cuenta es el modelo de la base de datos que se va a utilizar, ya que tendrá una gran importancia a la hora de decidir qué sistema de gestión de bases de datos se va a utilizar. Los siguientes modelos [15] han sido tenidos en cuenta:

- Bases de datos relacionales: en estos sistemas, la información se almacena en relaciones. Una relación, definidas también como tablas, es una colección de tuplas, o filas. Las relaciones se definen por su nombre y un número fijo de atributos, o columnas, con tipos de datos fijos. Estos sistemas respetan las propiedades ACID (Atomicity, Consistency, Isolation y Durability), y tienen operaciones básicas definidas, como la selección, proyección y unión.
- Bases de datos de documentos: la principal característica de estas bases de datos es la organización de los datos de forma libre, sin seguir ningún esquema. Esto significa, por contrario que en las bases de datos relacionales, las entradas no poseen una estructura uniforme, las columnas pueden tener más de un valor e incluso pueden almacenar estructuras anidadas.
- Bases de datos de Clave-valor: son las bases de datos más simples y solo almacenan pares clave-valor. Normalmente no son factibles

para aplicaciones complejas, pero normalmente presentan un gran rendimiento debido a su simplicidad.

- Motores de búsqueda: su uso principal es la búsqueda de datos, y son típicamente NoSQL, es decir, que no siguen un modelo relacional.
- Bases de datos de series temporales: estas bases de datos están optimizadas para almacenar series temporales [13]. Aunque son típicamente NoSQL, bases de datos de series temporales relacionales existen.

Cualquiera de estos modelos permiten el manejo de información de series temporales, como la información que recibimos de los AGV. Sin embargo, las bases de datos de series temporales están especializadas en este tipo de trabajos, por lo que las hace perfectas para nuestras necesidades.

Como selección inicial, se han escogido los cinco sistemas gestores de bases de datos de series temporales más temporales según el ranking DB-Engines [7]. Este ranking es utilizado en otros estudios, como [9]. Los sistemas gestores de bases de datos elegidos para comparar son:

- **InfluxDB 2.6.1** Un sistema gestor de bases de datos de series temporales desarrollado por InfluxData. Su uso principal es el almacenamiento y obtención de series temporales, creadas en operaciones de monitoreo de IoT, información de sensores, etc.
- **kdb+ 4.0** Una base de datos de series temporales relacional desarrollado por KX, usado principalmente en negociación bursátil de alta frecuencia, para almacenar y para procesar datos a una alta velocidad.
- **Prometheus 2.43.0** Una base de datos de series temporales usada para el monitoreo de eventos y alarmas, que utiliza un modelo HTTP.
- **Graphite 1.1.10** Una herramienta que almacena, monitoriza y grafica series temporales numéricas.
- **TimescaleDB 2.10.1** Una base de datos de código abierto, desarrollada como complemento a PostgreSQL con el fin de mejorar el rendimiento y análisis para series temporales.

Aunque este ranking solo mide la popularidad y ordena los sistemas en función de atributos sociales, es especialmente útil para soluciones de código abierto, pues esto generalmente significa que está soportada por una comunidad activa con muchos colaboradores involucrados en añadir nueva funcionalidad y corregir errores.

Procedimiento de la comparación La comparación y el filtrado de los modelos seleccionados ha sido realizados en tres pasos secuenciales:

1. Información general, soporte software y apoyo de la comunidad.
2. Modelo de datos y especificaciones técnicas.
3. Prueba de rendimiento.

En los primeros dos pasos, los sistemas comparados que no cumplan los requisitos especificados en la tabla 5.1 han sido descartados. Después, con los sistemas restantes, se ha realizado una prueba de rendimiento con el fin de tomar la decisión final. A su vez, esta prueba se divide en otras dos pruebas.

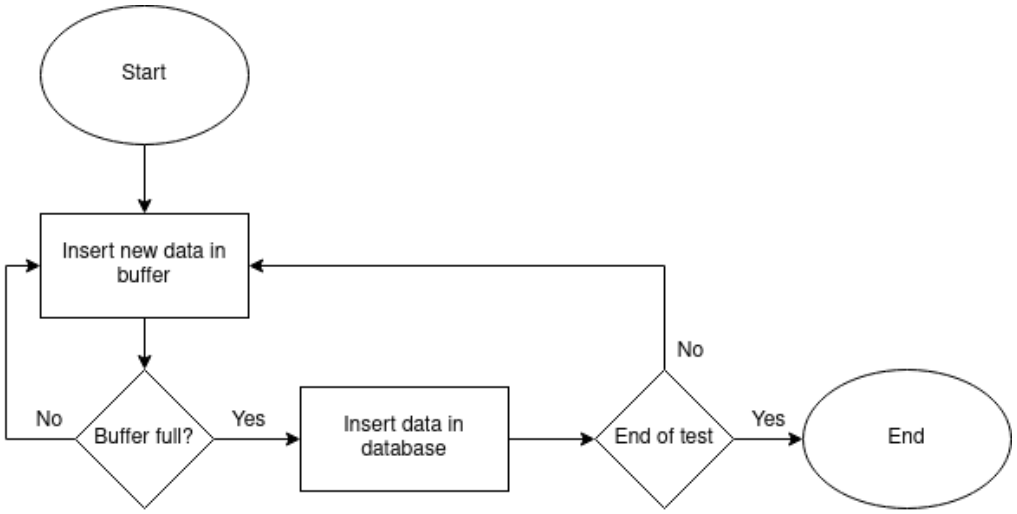


Figura 5.2: Procedimiento de la prueba de inserción

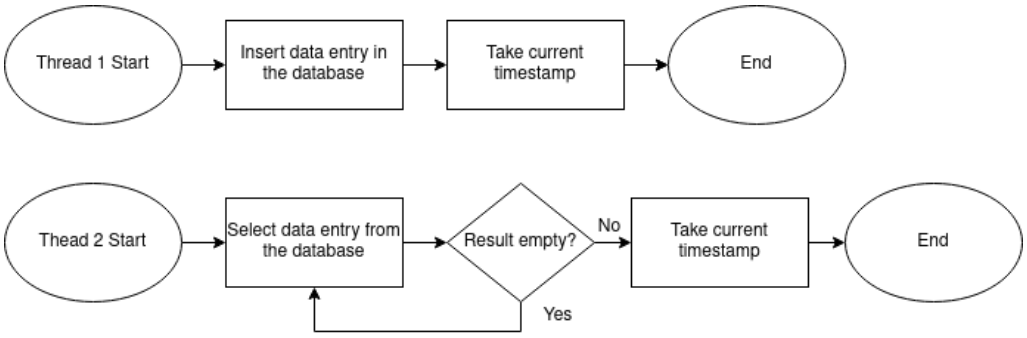


Figura 5.3: Procedimiento de la prueba de latencia

El primer test, (Figura 5.2) mide el uso de CPU y RAM del sistema cuando se insertan datos de forma masiva, así como el tiempo tomado y el número de inserciones por segundo realizadas. En total, 300.000 entradas son enviadas, formadas por los siguientes campos: timestamp, id del vehículo, batería, velocidad, posición en la coordenada x, posición en la coordenada y, temperatura y voltaje. Para realizar la prueba de manera más realista, los datos se insertan en la base de datos en tandas de 5.000. Para ello se almacenan primero en un buffer controlado por el servicio "Receiver", ya que de esta manera se obtiene un mejor rendimiento que si se insertasen de uno en uno. Para medir el uso de CPU y de RAM de la misma forma con todos los diferentes gestores, las medidas son tomadas de lo que reporta el estado del contenedor de Docker en el que se ejecutan dichos sistemas.

En el segundo test (Figura 5.3) se mide la latencia de inserción. Esto es, el tiempo que tarda en estar disponible una entrada después de su inserción. Para esto, un script de Python, formado por dos hilos, ha sido creado. El primero de esos hilos realiza la inserción en la base de datos y el otro intenta obtener dicha entrada en un bucle. En el momento en el que dicha entrada se obtiene, se anota dicho tiempo y se resta del momento en le que se insertó la entrada. Este test se realiza 200 veces y se hace una media con los resultados.

Cada test se realiza cinco veces para reducir la variabilidad, tomando la media de dichas ejecuciones como valor final.

Métricas de la comparación A continuación se detallan las métricas utilizadas para realizar la comparación.

Las métricas de información general analizan:

- Organización: que organización o compañía es responsable del desarrollo y mantenimiento del sistema.

- Año de lanzamiento: en que año se lanzó inicialmente.
- Última versión: en que año se lanzó la última versión.
- Licencia: que tipo de licencia tiene el software: código abierto (OSS), o licencia comercial.

El indicador de rendimiento del soporte de software analiza:

- Sistema Operativo: qué sistemas operativos se soportan.
- Soporte para Python: como el sistema está implementado en Python, nos interesa que el sistema tenga buen soporte de este lenguaje.
- Lenguaje de consultas: que lenguaje de consultas soporta el sistema.
- Plugins para aprendizaje automático: si el sistema soporta plugins que simplifiquen la predicción de nuevos datos.

La comparación del soporte de la comunidad contiene:

- Número de estrellas del repositorio de GitHub: como forma de medir su popularidad.
- Pull requests: número de pull requests enviadas en el último mes.
- Pull requests aceptadas: número de pull requests aceptadas en el último mes.
- Issues: número de issues creados en el último mes.
- Issues cerrados: número de issues cerrados en el último mes.

Estos cuatro últimos campos se usarán para comparar que comunidad es más activa.

El indicador de rendimiento del modelo de datos compara:

- Modelo de datos: que modelo concreto implementa cada sistema.
- Esquema: un esquema puede verse como una plantilla que define como se almacena la información. Este campo compara si la organización de los datos es estricta (esquema fijo) o no (esquema libre).

- Índices secundarios: si el sistema soporta índices secundarios para un mejor rendimiento de consultas o no.
- Precisión temporal: unidad mínima de tiempo que puede tener una entrada.

El indicador de rendimiento de información técnica se compone de los siguientes campos:

- Scripts de servidor: si el sistema es capaz de ejecutar scripts en el servidor o no.
- Método de partición: si se soportan o no métodos de partición para una mayor escalabilidad.
- Replicación: que métodos de replicación soporta.
- Consistencia: si la información escrita es consistente o no.
- Conformidad con ACID: si el sistema sigue los principios ACID o no.
- Concurrencia: si el sistema soporta accesos concurrentes o no.
- Durabilidad: si la información es persistente, incluso si falla.
- Método de inserción: si la información se introduce mediante una consulta de inserción o extrayendo los datos de un endpoint de forma periódica.

Por último, el análisis de rendimiento compara:

- Tiempo de inserción: tiempo que se tarda en hacer la prueba de inserción en segundos.
- Tasa de transferencia: número de inserciones por segundo.
- Uso de la CPU: uso de la CPU del contenedor de Docker en el que se ejecuta base de datos durante la primera prueba.
- Uso de RAM: uso de RAM del contenedor de Docker en el que se ejecuta la base de datos durante la primera prueba.
- Latencia: tiempo que tarda en ejecutarse la segunda prueba en milisegundos.

Experimentos y resultados

Systems	Organization	Launch year	Latest version	License
InfluxDB	InfluxData	2013	2023	OSS
db+	Kx Systems	2000	2020	Comercial
Prometheus	-	2015	2023	OSS
Graphite	-	2006	2022	OSS
TimescaleDB	Timescale	2017	2023	OSS

Tabla 5.2: Comparativa de información general

Información general (Tabla 5.2) Solo software de código abierto será considerado en este trabajo. Aunque kdb+ tiene una versión de 32 bits, no se usará y no volverá a aparecer en las siguientes comparaciones. Por esta razón también, solo las características de la "Comunity Edition" de InfluxDB serán utilizadas en dichas comparaciones, y características de la "Enterprise Edition", que no es de códigp abierto, no se tendrán en cuenta.

System	OS	Python	Query language	ML Plugins
InfluxDB	Linux OS x	Sí	Flux and InfluxQL	Loud ML
Prometheus	Linux Windows	Sí	PromQL	No
Graphite	Linux Unix	Sí	No	No
TimescaleDB	Linux OS X Windows	Sí	SQL	No

Tabla 5.3: Soporte software

Soporte software (Tabla 5.3) Todos los sistemas soportan Linux y Python, el sistema operativo y lenguaje utilizados. Solo Graphite no tiene un lenguaje de consultas definido, aunque se pueden realizar utilizando lo que llaman Funciones [1]. Sólo InfluxDB soporta plugins para aprendizaje automático. Otros sistemas como TimescaleDB proveen documentación para realizarlo de forma externa en lenguajes como Python o R [3].

Sistemas	Estrellas GitHub	Pull requests	Pull requests aceptadas	Issues	Issues cerrados
InfluxDB	25.2k	26	22	37	13
Prometheus	47.4k	75	53	42	20
Graphite	5.6k	0	0	0	0
TimescaleDB	14.7k	105	83	67	46

Tabla 5.4: Soporte de la comunidad

Soporte de la comunidad (Tabla 5.4) Como muestra la tabla, todos los proyectos son muy activos a excepción de Graphite.

Sistema	Modelo	Esquema	Tipado	Índice secundario	Precisión temporal
InfluxDB	Multidimensional	Libre	Numéricos y strings	No	Nanosegundos
Prometheus	Multidimensional	Sí	Numéricos	No	Milisegundos
Graphite	Key-Value	Sí	Numéricos	No	Segundos
TimescaleDB	Relacional	Sí	Tipos SQL	Sí	Nanosegundos

Tabla 5.5: Comparativa del modelo de datos

Comparación del modelo de datos (Tabla 5.5) Tanto InfluxDB como Prometheus utilizan un modelo multidimensional. Este modelo puede verse como un modelo clave-valor multidimensional: las entradas de datos están formados por un campo que describe la información almacenada ("nombre de la métrica" para Prometheus y "medida" para InfluxDB) y un set de pares clave-valor asociados con un timestamp. La principal diferencia es que las entradas en el modelo de InfluxDB están formados por la medida, un set de etiquetas y un set de valores, en vez de solo un set de pares clave-valor. Estas etiquetas guardan metadatos en forma de cadenas de caracteres, son opcionales y están indexados, mientras que el set de valores guardan la información, no están indexados y están asociados con un timestamp.

Graphite agrega los datos de manera automática en ventanas de un segundo o más. Este comportamiento no es el deseado para nuestras necesidades, ya que se requiere almacenar todos los datos enviados.

Sistema	InfluxDB	Prometheus	Graphite	TimescaleDB
Scripts del servidor	No	No	No	Sí
Particionamiento	No	Sharding	No	Sí
Replicación	No	Sí	No	Sí
Consistencia	Eventual	No	No	Innmediata
ACID	No	No	No	Sí
Concurrencia	Sí	Sí	Sí	Sí
Durabilidad	Sí	Sí	Sí	Sí
Permisos de usuario	Permisos vía cuentas	No	No	Derechos estandar SQL
Método inserción	Push	Pull	Push	Push

Tabla 5.6: Comparativa de información técnica

Comparativa información técnica (Tabla 5.6) En InfluxDB, la consistencia es eventual. Según la documentación, se prioriza el rendimiento de lectura y escritura antes que una fuerte consistencia. Se asegura, sin embargo, que la información es eventualmente consistente [12].

En bases de datos típicas, la información se inserta a través de algún tipo de consulta desde fuera. Por otro lado, Prometheus escucha a un endpoint en el que se publican los datos y se obtienen en intervalos fijos de tiempo. Esto significa que la inserción solo ocurrirá cuando Prometheus escuche a dicho endpoint, por lo que la información no se puede insertar en cualquier momento. Un método más típico existe, pero no es recomendado y no es posible especificar timestamps [5].

Solo InfluxDB y TimescaleDB cumplen todos los requisitos, ya que Graphite agrega los datos en ventanas de 1 segundo, haciendo imposible obtener datos de un momento concreto, y la forma de inserción de Prometheus le hace incompatible con nuestras necesidades, ya que es necesario poder insertar datos en cualquier momento. Por esto, solo estos dos sistemas se compararán en la prueba de rendimiento.

Análisis del rendimiento La prueba se realizó con un procesador AMD Ryzen 5 3600 y 32 GB de RAM. Ya que este modelo de CPU tiene 12 hilos, el uso de la CPU puede ser tan alto como 1200 % (Uso CPU = Hilos * 100). Los resultados de esta prueba se muestran en la tabla 5.7.

System	InfluxDB	TimescaleDB
Tiempo inserción (s)	24.13	**1.16**
Tasa inserción (I/s)	12432.66	**258620.69**
Uso CPU (%)	**15.05**	55.32
Uso RAM (MB)	**219.85**	373.73
Latencia (ms)	3.37	**0.22**

Tabla 5.7: Resultados de la prueba de rendimiento

Como se puede observar, InfluxDB es más lento en todos las pruebas que TimescaleDB, pero este último utiliza más recursos del sistema. Si en un futuro es necesario escalar InfluxDB puede ser mejor opción, ya que un menor uso de recursos suele significar un menor coste. Sin embargo, TimescaleDB es más flexible, ya que al estar basado en PostgreSQL tiene todas sus características. Cualquiera de estos dos sistemas puede ser perfectamente usado según los requisitos marcados.

Elección

Al final, el sistema gestor de bases de datos escogido ha sido InfluxDB. Aunque sea más lento que TimescaleDB, tiene una velocidad lo suficientemente buena, y al consumir menos recursos la hace una opción más barata en caso de que esta solución se aplique comercialmente.

Otro motivo de peso para elegir InfluxDB es que viene por defecto con una interfaz web llamada Chronograf (Figura 5.4) en la que se puede manejar la base de datos, crear gráficas, establecer alarmas, etc.

Figura 5.4: Interfaz Chronograf

Desarrollo de los servicios

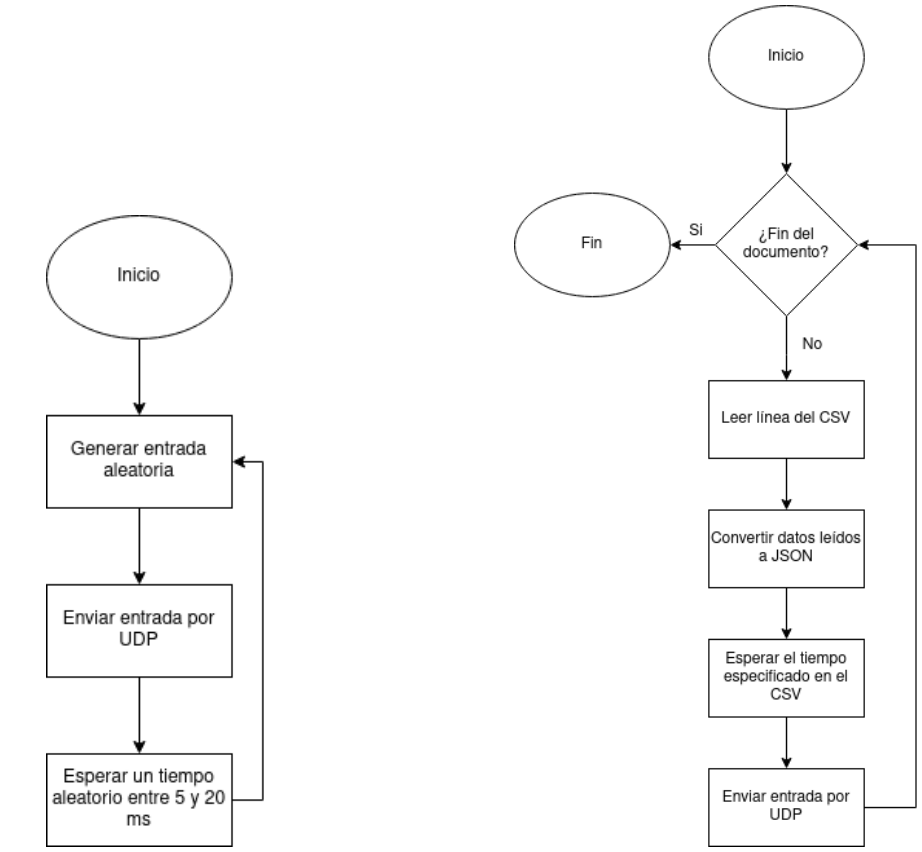
Una vez diseñada la arquitectura y el sistema gestor de bases de datos escogido, se procedió a desarrollar los sistemas definidos.

Simulator

Inicialmente, no disponía de datos reales del AGV, por lo que la primera versión (Figura 5.5a) simplemente generaba datos aleatorios con campos aleatorios, y los enviaba al nodo "Receiver" por UDP utilizando el puerto 5004.

Después, se intentó desarrollar un simulador capaz de, valga la redundancia, simular el comportamiento de un AGV. Sin embargo, una vez dispuse de datos reales del AGV, esta idea se descartó, pues simular dicha información de forma precisa iba a ser demasiado complejo, y se escapa del objetivo de este proyecto. Por tanto, se decidió simular el comportamiento del AGV leyendo los datos de un CSV (Figura 5.5b) obtenido a partir de uno real.

Por último, en la versión final, se unificaron los dos procedimientos, de forma que el comportamiento del simulador se decide según lo especificado en un archivo de configuración.



(a) Generación aleatoria

(b) Generación por CSV

(c) Versión final

Figura 5.5: Diagramas de flujo del simulador

Receiver

El funcionamiento del nodo "Reciever" es muy simple (Figura 5.6): escucha el puerto UDP 5004, y cuando recibe información, la inserta en la base de datos. Inicialmente, el servicio intentará conectarse a la base de datos. Si esta conexión falla un número determinado de veces, el servicio fallará informando de que la conexión no ha podido realizarse.

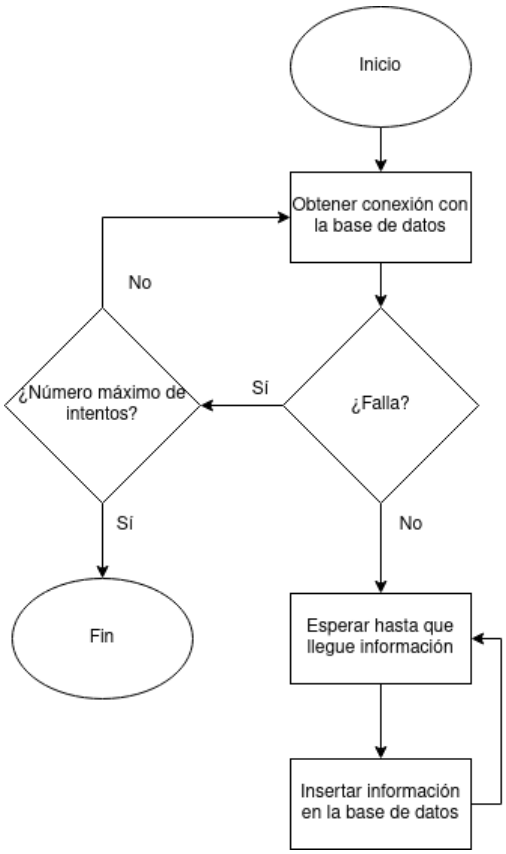


Figura 5.6: Diagrama de flujo del servicio "Reciever"

5.2. Segunda fase

La segunda fase se centró en el desarrollo y diseño del servicio de predicción de nuevos datos. Se ha realizado un análisis comparativo de diferentes modelos predictivos, así como un análisis de los datos recibidos por el AGV para hacer la mejor elección.

Modificación de la arquitectura

Ya que se ha pretendido desarrollar una arquitectura lo más modular posible, la predicción de la información se realiza desde un nuevo servicio. Por ende, la arquitectura modificada queda de la siguiente manera:

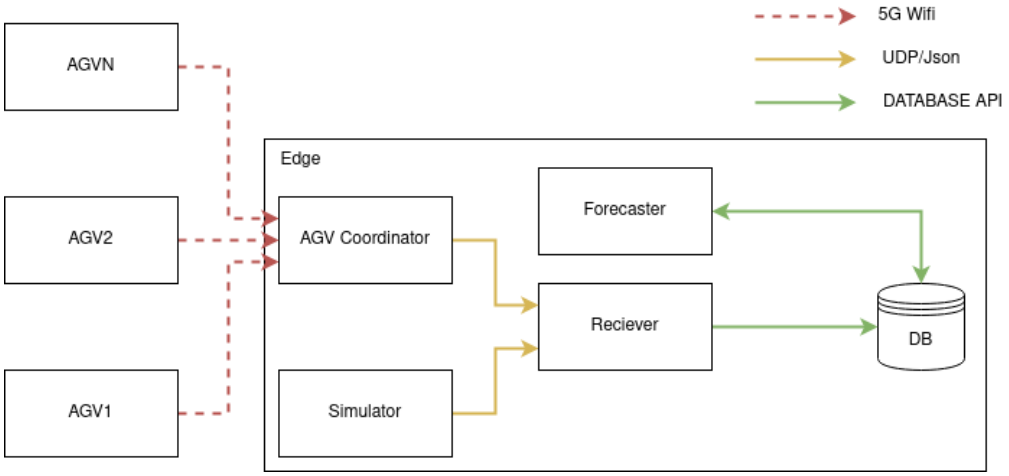


Figura 5.7: Arquitectura final

Este nuevo servicio se conecta directamente a la base de datos. Con esta conexión, obtiene la información necesaria para hacer las predicciones, y una vez hechas se insertan en la base de datos para poder ser visualizadas en Chronograf (Figura 5.4).

Comparativa de modelos de predicción

Desarrollo del servicio

5.3. Puesta en marcha

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Functions — Graphite 1.2.0 documentation.

[2] Tex live - tex users group.

[3] Timescale Documentation | Time-series forecasting.

[4] Bruno Baruque, Santiago Porras, Esteban Jove, and José Luis Calvo-Rolle. Geothermal heat exchanger energy prediction based on time series and monitoring sensors optimization. *Energy*, 171:49–60, 2019.

[5] Prometheus community. Prometheus pushgateway. [Internet; read on 26-april-2023].

[6] Fatoumata Dama and Christine Sinoquet. Analysis and modeling to forecast in time series: a systematic review. *CoRR*, abs/2104.00164, 2021.

[7] DB-Engines. Db-engines ranking of time series dbms, March 2023. [Internet; read on 22-march-2023].

[8] F Espinosa, C Santos, and JE Sierra-García. Transporte multi-agv de una carga: estado del arte y propuesta centralizada. *Revista Iberoamericana de Automática e Informática industrial*, 18(1):82–91, 2020.

[9] Piotr Grzesik and Dariusz Mrozek. Comparative analysis of time series databases in the context of edge computing for low power sensor networks. In Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 371–383, Cham, 2020. Springer International Publishing.

[10] Julien Herzen, Francesco LÃ¤ssig, Samuele Giuliano Piazzetta, Thomas Neuer, LÃ©o Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan KoÅ›cisz, Dennis Bader, FrÃ©dÃ©rick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and GaÃ«l Grosch. Darts: User-friendly modern machine learning for time series. *Journal of Machine Learning Research*, 23(124):1–6, 2022.

[11] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Melbourne, Australia, 2nd edition, 2018. OTexts.com/fpp2.

[12] InfluxData. Influxdb design principles. [Internet; read on 22-march-2023].

[13] InfluxData. What is time series data? [Internet; read on 22-march-2023].

[14] Genshiro Kitagawa. *Introduction to time series modeling*. CRC press, 2010.

[15] Andreas Meier and Michael Kaufmann. *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. Springer Vieweg, 2019.