Comparative study of open source database management systems to enable predictive maintenance of Autonomous Guided Vehicles

Gonzalo Burgos[1], J. Enrique Sierra-García[2], Bruno Baruque-Zanón[1]

[1] Dpt. of Digitalization, University of Burgos, Burgos, Spain
[2] Dpt. of Electromechanical Engineering, University of Burgos, Burgos, Spain
gbd1004@alu.ubu.es,jesierra@ubu.es,bbaruque@ubu.es

Abstract. A number of open source database systems have been researched and assessed in this work for the purpose of choosing the optimum technology to implement predictive maintenance systems for industrial Autonomous Guided Vehicles. An application-driven technique has been suggested as a way to achieve it. The use case and its specifications are first outlined and listed. The top five most popular time series database systems are then contrasted based on a variety of technical metrics including software support, community support, and different technical features. From this analysis the best two options are selected (InfluxDB and TimeScale DB). The performance of these two is then further examined, taking into account performance indicators like insert time, throughput, and resource consumption. Results show that TimeScale DB provides a higher performance but demands considerably more resources.

Keywords: database management systems, time series databases, performance analysis, AGVs, industrial applications

1 Introduction

Autonomous Guided Vehicles (AGVs) are complex robotic systems, able to move around a given environment, which are used to transport heavy loads in factories or warehouses, and are designed to improve efficiency and productivity in logistics and material transportation. Due to their advantages in security, flexibility and speed, this technology is becoming increasingly common [1].

Although they may improve productivity, misalignment in their configuration or operational errors might lead to reduced throughput and in extreme cases can cause a production line shutdown. For this reason, it is necessary to extract data from running systems to analyze the performance of machines and logistics applications. This data can be used to predict future system behavior, enable preventive and predictive maintenance operations and provide feedback to design for the continuous improvement of the machines. These predictions can be achieved with the use of time series analysis algorithms that enable to anticipate future conditions of the system [2].

Some data have a low update frequency such as the temperature and the battery voltage, but others change each few milliseconds such as the electric current, the speed and position of the vehicle, errors and status, etc. All this information provided by the AGVs must be related with the time when it is generated, thus it can be grouped in time series data [3]. Any type of database can be used to store AGV data, however, since time series are involved, it is preferable to use time series databases to optimize their performance.

There are different open source database management systems to perform this task available in the market, however there is some variability in performance: some are faster than others, some are more efficient, others consume more resources, etc., so it is advisable to properly select the technology for each application in the planning or design phase.While the field has seen the exploration of similar concepts in additional literature [4], [5], our primary aim is to undertake an exhaustive examination of the database system that directly aligns with the requirements of our particular use case, which is the implementation of the monitoring platform for an AGV fleet.

In this work, several database systems have been studied and evaluated to select the best technology to deploy predictive maintenance systems for industrial AGVs. To do this, an application-driven methodology has been proposed. First, the use case and its requirements are described and identified. Then the top five most popular time series database systems are compared considering the software support, the community support and different technical indicators. From this comparison the two best tools are selected. These two are then subjected to a further performance analysis, considering performance metrics such as the insert time, the throughput, and the consumed resources.

The remaining part of the paper is structured as follows. Section 2 describes the industry 4.0 use case. The methodology of the study is explained in Section 3. The results are discussed in Section 4. The paper ends with the conclusions and future work.

2 Use case

The architecture of the system under study is depicted in Figure 1. The vehicles send their data through a 5G/WiFi connection, encoded as a byte string. Then, the "AGV Coordinator" software node, receives and decodes the messages, and transform them to the JSON format to later forward them, properly ordered, to the "Receiver" through the UDP protocol. This "Receiver" then redirects this messages to the database system that will store them. In case there are no AGV available and data needs to be generated for testing purposes, the "Receiver" may receive information from the "Simulator" software node, that will simulate the data flow sent from a fleet of AGV.

Time plays an important role in this messages. For example, if one of those messages provides information about the battery level, it is important to know when that level was recorded. Some of those variables are sent every 10 to 20 milliseconds, so a high time precision is needed.

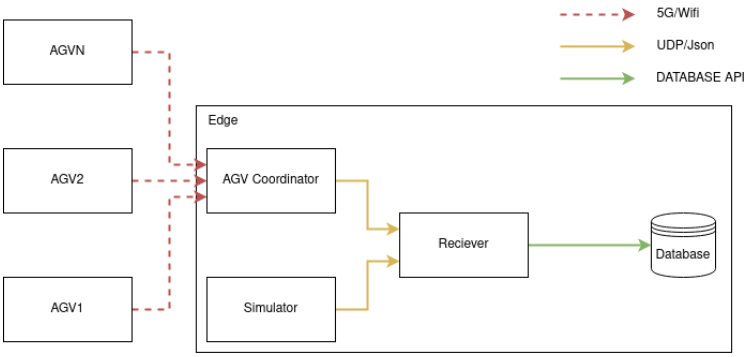


Fig. 1: AGV fleet and logging system architecture diagram

Our intention with this work is to find the database management system that best fits our use case. Table 1 contains the functional and non-functional requirements considered for our study.

Requirement	Description
FR1:	The database management system must support timestamped entries.
FR2:	Data should be able to be inserted at any moment.
FR3:	Data entries should be inserted with the timestamp corresponding to the time they were created in the AGVs.
FR4:	The database management system must support data entries with a time precision in the milliseconds range.
NFR1:	The database management system must be open source.
NFR2:	It must have good support for Linux and Python

Table 1: Functional (FR) and Non-Functional (NFR) Requirements

With the use case defined, the next section covers the methodology used to compare and choose the database management system that best suits such use case.

3 Methodology

3.1 Database management systems under study

The first thing considered is the database model to use, as this will have a key importance when deciding which database management system to use. Any of these models allow the management of time series data, like the data received from the AGVs. However, Time Series database management systems are specialized in this kind of work, making them the best suit for our needs.

For the initial screening, the most popular time series database management systems according to the DB-Engines ranking [6], have been selected. This ranking is used in previous related studies, like [4]. The time series database management systems chosen to be compared are:

1. InfluxDB (version 2.6.1) [7]. An open source time series database developed by InfluxData. Its main usage is the storage and retrieval of time series data, created in operations of monitoring IoT, sensor data, etc.
2. kdb+ (version 4.0) [8]. A relational time series database developed by KX, used mainly in high-frequency trading, to store and process data at high speed.
3. Prometheus (version 2.43.0) [9]. A time series database used for event monitoring and alerting, using an HTTP pull model.
4. Graphite (version 1.1.10) [10]. A tool that stores, monitors and graph numeric time series data.
5. TimescaleDB (version 2.10.1) [11]. An open source time series database build on top of PostgreSQL, improving the performance and analysis facilities for time series data.

Although this ranking only measures popularity and ranks the systems according to social attributes, it's really useful especially for open source solutions, as this usually means that it is supported by an active community with lots of collaborators involved in adding new functionality and correcting errors.

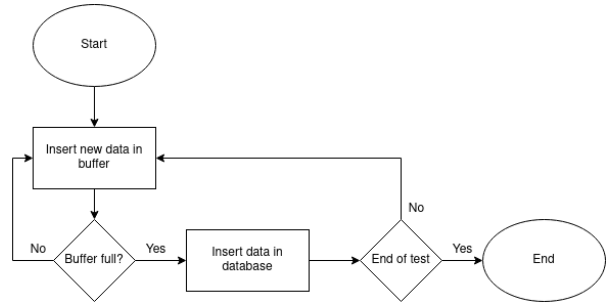
3.2 Comparison procedure

The comparison and filtering of the initially selected models has been performed following several sequential steps: software and community support; data model; technical specifications; and finally a performance test.

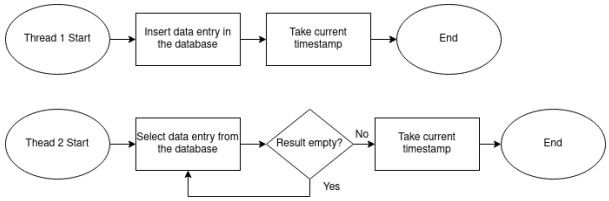
In the first two steps, the systems have been compared and those that do not meet the requirements specified in Table 1 have been discarded. Then, with the remaining systems, a performance test has been used to make the final decision.

In addition, the performance test has been separated in two complementary tests. The first test (Figure 2a) measures the CPU and RAM usage by the system when inserting data in bulk, as well as the time taken and the number of insertions per second performed. In total, 300000 entries are sent, consisting of the following fields: timestamp, vehicle id, battery, speed, x-coordinate position, y-coordinate position, temperature and voltage. To perform a more realistic simulation of the previously described AGV fleet system, the data has been sent in batches of 5000 entries that are stored in a buffer managed by the "Receiver", as this generally yields better performance than sending the entries one by one. To measure the use of CPU and RAM in the same way with all systems, the measurements have been extracted from what has been reported by the status of the Docker container in which the systems are executed [12].

In the second test (Figure 2b) the data latency is measured. That is, the time it takes for an entry to become available after insertion. For this purpose, a Python script, including two threads, has been created. One thread performs an insertion in the database and another one performs the retrieval of that same entry in a loop. At the moment the entry is obtained, the time at which it occurred is noted and subtracted from the time at which the data was inserted. This test is done 200 times and the results are averaged. Each test has been



(a) Insertion performance test procedure



(b) Latency test procedure

Fig. 2: Performance tests

executed 5 times to reduce variability. The results then are the average of all the executions.

3.3 Comparison perspectives

The software support aspects analyze:

- Organization: which organization or company is responsible for the development and maintenance of the system.
- Operative System: What operative systems are supported. For us, only Linux support matters.
- Python support: Our solution has been implemented in Python. For the reason, having good Python support and APIs is highly appreciated.
- Query language: which query language each database supports.
- ML Plugin: whether the system has any machine learning plugin that simplifies the forecasting of new data.

The community support indicates which system has the most active community and the characteristics compared include:

- Number of starts of the public GitHub repository: This will be used as a way of measuring its overall popularity.
- Pull requests: how many pull requests where submitted in the last month.
- Merged pull requests: of the previous pull request, how many of them have been merged.

- Issues: how many issues where submitted in the last month.
- Closed Issues: how many of those issues where closed.

The data model perspective contains a comparison of the following features:

- Data model: which data model the system implements.
- Schema: A Schema can be seen as a blueprint that defines how data is stored. This field compares whether the data organization is strict (fixed schema) or not (schema-free).
- Secondary indexes: whether the system support secondary indexes for improved query performance.
- Time precision: the minimum time unit the timestamp of an entry can have.

The technical information perspective comprises a set of technical information fields:

- Server-side scripts: whether the system is able to execute server-side scripts.
- Partitioning method: support for partitioning methods for future scalability.
- Replication: what methods of replication the system supports, if any.
- Consistency: whether the data written is assured to be consistent or not.
- ACID compliance: whether the system follows the ACID principles (Atomicity, Consistency, Isolation, Durability)
- Concurrency: whether the system supports concurrent accesses.
- Durability: whether the data persists in the database, even if the system fails.
- Data insertion approach: whether it is done by pushing data to the database through an insert query or by pulling data from an endpoint periodically.

Lastly, the performance benchmark contains the following metrics:

- Insert time: time that takes to do the insert test, measured in seconds.
- Throughput: number of insertions per second.
- CPU usage: CPU usage of the Docker container with the database during the first test, measured as a percentage.
- RAM usage: RAM usage of the Docker container with the database during the first test, measured in Megabytes.
- Latency: time of the second test, measured in milliseconds.

4 Experiments and results

4.1 Functional comparison

Only Open Source Software has been considered for this work. Although Kdb+ has a 32 bit free version, it has not be used and does not appear in the following comparisons. For the same reason, only the features of the Community Edition of InfluxDB has not been used, and features of the Enterprise Edition, which is not open source, has not been considered.

All of the systems support Linux and Python, the operative system and language we intend to use. Only Graphite does not have a defined query language,

although data querying is possible using "Functions" [13]. Only InlfuxDB supports a plugin for machine learning applications. Other systems, like TimescaleDB, provide documentation to do it externally in Python or R [14]. The Table 2 summarizes the software support of the systems compared. On the other hand, as the Table 3 shows, every project is highly active except for Graphite.

System	Organization	OS	Python	Query lang.	ML Plugins
InfluxDB	InfluxData	Linux/OS x	Yes	Flux/InfluxQL	LoudML
Prometheus	-	Linux/Windows	Yes	PromQL	No
Graphite	-	Linux/Unix	Yes	No	No
TimescaleDB	Timescale	Linux/Windows/OS x	Yes	SQL	No

Table 2: Software support

Systems	Github Stars	Pull requests	Merged pull requests	Issues	Closed Issues
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

Table 3: Community Support

Regarding the data model, both InfluxDB and Prometheus use a multidimensional mode. This model may be seen as a multidimensional key-value model: data points consist of a field that describes the data stored ("metric name" for Prometheus, and "measurement" for InfluxDB) and a set of key-value pairs associated with a timestamp. The main difference is that InfluxDB's data points consist of the measurement, a tag set and a field set, instead of only a key-value set. Tags store metadata as string values, are optional and are indexed, while fields store the actual data, are not indexed and are associated with a timestamp. Graphite automatically aggregates all data sent in windows of 1 second or more. This behaviour is not desired for our needs, as it is required to store all data sent. This comparison is shown in Table 4.

System	Model	Schema	Typing	Secondary Index	Time precision
InfluxDB	Multidimensional	Schema-free	Numeric and string	No	Nanosecond
Prometheus	Multidimensional	Yes	Numeric	No	Millisecond
Graphite	Key-Value	Yes	Numeric	No	Second
TimescaleDB	Relational	Yes	SQL types	Yes	Nanosecond

Table 4: Data model comparison

Consistency in InfluxDB is eventual, meaning that, according to the InfluxDB documentation, it prioritizes read and write performance over strong consistency. It is ensured, however, that data is eventually consistent[15].
In typical databases, the data is inserted through some type of query from the outside (the data is "pushed" to the database). Prometheus, on the contrary,

listens to an endpoint in which the data is published and "pulls" it at a fixed time interval. This means that an insertion will only happen when Prometheus polls data from the specified endpoint, thus, data may not be inserted at any time. A "Push" method is available, but it is not recommended, and it is not possible to specify timestamps.[16]. Only InfluxDB and TimescaleDB meet all the requirements, as Graphite aggregates all the data in 1-second windows, making it impossible to obtain concrete data, and the way the information is inserted in Prometheus makes it incompatible with our requirements, as it is required to be able to insert data at any time. For this reason, only these two systems have been compared in the performance tests.

System	InfluxDB	Prometheus	Graphite	TimescaleDB
Server scripts	No	No	No	Yes
Partitioning	No	Sharding	No	Yes
Replication	No	Yes	No	Yes
Consistency	Eventual	No	No	Innmediate
ACID compliance	No	No	No	Yes
Concurrency	Yes	Yes	Yes	Yes
Durability	Yes	Yes	Yes	Yes
User rights	Rights via user accounts	No	No	Access rights SQL-standard
Insert method	Push	Pull	Push	Push

Table 5: Technical information

4.2 Performance evaluation

The test where performed in a computer with a AMD Ryzen 5 3600 CPU and 32 GB of RAM. Because this model of CPU has 12 threads, CPU usage may be as high as 1200% (CPU Usage = Threads $*$ 100). The results of this performance evaluation are shown in Table 6. These results include the the system resources used in the bulk insertion of 300000 entries and the latency tests (last row). Latency is measured, only in that case, as the average time for one entry.

System	InfluxDB	TimescaleDB
Insert time (s)	24.13	**1.16**
Throughput (I/s)	12432.66	**258620.69**
CPU Usage (%)	**15.05**	55.32
RAM Usage (MB)	**219.85**	373.73
Latency (ms)	3.37	**0.22**

Table 6: Performance test results

According to the performance test (Table 6), InfluxDB is slower in every test than TimescaleDB, but the second uses more system resources. If scaling is necessary in the future, InfluxDB may be more convenient, as less system resource usage usually means less costs. However, TimescaleDB is more flexible, because

it extends the functionality of PostgreSQL. Thus both of these systems could be perfectly used for our use case. However, if enough computational resources are available, TimescaleDB is recommended because of its better performance and flexibility.

Here, we mainly focus on testing the performance of data insertion and availability, because at the moment we need a database that is able to store all the data sent by the fleet as fast as possible. In future work, the retrieval of that data should be measured and compared. Another thing worth to mention is that InfluxDB 3.0 version is, at the time of publication, expected to be released soon. This new version is supposed to greatly improve performance [17], so the results of this test could be outdated.

5 Conclusions and Future Work

In this study, we compared several database systems for storing data obtained from an AGV fleet in a generic industrial environment setting. Our goal was to identify which database system performed best in terms of system resources and data processing speed for managing time series data.

The results of the tests indicate that there is not a clear best system overall. While TimescaleDB is 95.19% faster, it consumes 267.57% more of CPU time, and uses 69.99% more RAM. Therefore, the decision must be taken between data availability speed and system usage when designing the final system.

Several directions for future research have been identified. One direction is to evaluate the performance of these database systems under different AGV scenarios, such as varying load capacities, different types of movements, and varying levels of complexity in layouts or AGV fleet. Another area for future research is to investigate the integration of these database systems with other emerging technologies such as machine learning and artificial intelligence. This could involve exploring how these technologies may be used to optimize data operations by AGVs, as well as how this data may be leveraged to improve decision-making in industrial operations.

Acknowledgement

This work was partially supported by the European Commission, under European Project 5G-Induce, grant number 101016941.

References

[1] F Espinosa, C Santos, and JE Sierra-García. “Transporte multi-AGV de una carga: estado del arte y propuesta centralizada”. In: *Revista Iberoamericana de Automática e Informática industrial* 18.1 (2020), pp. 82–91.

[2] Bruno Baruque et al. "Geothermal heat exchanger energy prediction based on time series and monitoring sensors optimization". In: *Energy* 171 (2019), pp. 49–60. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2018.12.207. URL: https://www.sciencedirect.com/science/article/pii/S0360544218325817.

[3] Fatoumata Dama and Christine Sinoquet. "Analysis and modeling to forecast in time series: a systematic review". In: *CoRR* abs/2104.00164 (2021). arXiv: 2104.00164. URL: https://arxiv.org/abs/2104.00164.

[4] Piotr Grzesik and Dariusz Mrozek. "Comparative Analysis of Time Series Databases in the Context of Edge Computing for Low Power Sensor Networks". In: *Computational Science – ICCS 2020*. Ed. by Valeria V. Krzhizhanovskaya et al. Cham: Springer International Publishing, 2020, pp. 371–383. ISBN: 978-3-030-50426-7.

[5] Alexey Struckov et al. "Evaluation of modern tools and techniques for storing time-series data". In: *Procedia Computer Science* 156 (2019). 8th International Young Scientists Conference on Computational Science, YSC2019, 24-28 June 2019, Heraklion, Greece, pp. 19–28. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.08.125. URL: https://www.sciencedirect.com/science/article/pii/S1877050919310439.

[6] DB-Engines. *DB-Engines Ranking of Time Series DBMS*. [Internet; read on 22-march-2023]. Mar. 2023. URL: https://db-engines.com/en/ranking/time+series+dbms.

[7] InfluxData. *InfluxDB: Open Source Time Series Database | InfluxData*. Dec. 2021. URL: https://www.influxdata.com/.

[8] KX. *KX: Make Data Driven Decisions 100X Faster At 1/10th The Cost*. Mar. 2023. URL: https://kx.com/.

[9] Prometheus. *Prometheus - Monitoring system & time series database*. URL: https://prometheus.io/.

[10] *Graphite*. URL: https://graphiteapp.org/.

[11] *Time-series data simplified*. URL: https://www.timescale.com/.

[12] Docker Inc. *Docker*. [Internet; read on 5-may-2023]. URL: https://www.docker.com.

[13] *Functions — Graphite 1.2.0 documentation*. URL: https://graphite.readthedocs.io/en/latest/functions.html.

[14] *Timescale Documentation | Time-series forecasting*. URL: https://docs.timescale.com/tutorials/latest/time-series-forecast/.

[15] InfluxData. *InfluxDB design principles*. [Internet; read on 22-march-2023]. URL: https://docs.influxdata.com/influxdb/cloud/reference/key-concepts/design-principles/.

[16] Prometheus community. *Prometheus Pushgateway*. [Internet; read on 26-april-2023]. URL: https://github.com/prometheus/pushgateway#readme.

[17] InfluxData. *InfluxData Unveils Future of Time Series Analytics with InfluxDB 3.0 Product Suite*. [Internet; read on 22-march-2023]. URL: https://www.influxdata.com/blog/influxdata-announces-influxdb-3-0/.