Time Series or Relational Database for Edge and IoT

Anže Luzar
XLAB Research
XLAB d.o.o.
Pot za Brdom 100
1000 Ljubljana, Slovenia
anze.luzar@xlab.si

Sašo Stanovnik
XLAB Research
XLAB d.o.o.
Pot za Brdom 100
1000 Ljubljana, Slovenia
saso.stanovnik@xlab.si

Matija Cankar
XLAB Research
XLAB d.o.o.
Pot za Brdom 100
1000 Ljubljana, Slovenia
matija.cankar@xlab.si

ABSTRACT

In Fog and Edge computing data management and processing is moving from the Cloud closer to the IoT devices. To perform the work in edge devices, different, more lightweight, small-footprint and specialized tools need to be employed. In this paper we perform a side-by-side comparison of relational and time series databases of their speed and resource consumption. The Results show better performance of time series over relational databases.

Keywords

database, SQL, timeseries, relational, cloud, fog, IoT, edge

1. INTRODUCTION

Nowadays plenty of databases are available for storing actual timestamped data to a database. In the past several years, there are growing appeals for reading and storing data from IoT devices [8]. In the close future, storing and manipulating time series data will play an important role in IoT [1]. A common technique is to gather the data using cloud or fog devices that read IoT sensors [4] and temporarily store it in relational or time series databases. As not all sensor readouts are required to be stored in the Cloud, it is useful to filter and process the sensors on the Edge near them and thus save Cloud resources and bandwidth. To achieve this, one of the most important and desired abilities for Edge devices is to handle volumes of time series data quickly with minimum latency and footprint in order to give the observer results as quickly as possible [11], [9]. IoT devices can rely on different database types behind them and the best choice mostly depends on the type and format of the data that is being stored and on the requirements of the edge device [6]. Among available database types that can be used for IoT devices are NoSQL (e.g. MongoDB) with its subtype time series database (e.g. InfluxDB, Prometheus, TimecaleDB) and relational database (e.g. PostgreSQL, MySQL, MSSQL). Global trends unveil that time series databases are currently the fastest growing database type [7].

This paper explores the efficiency of relational and time series databases on edge devices by measuring and comparing response times and memory footprints of two representatives. From the results a reader can conclude which type of the database is better for a specific edge device or fog-like environment.

The research of the database performance will be further used in a fog-to-cloud application called Smartboat, which is developed as a use case for EU H2020 funded project called mF2C[5]. The application's goal is to establish support for boats that would simplify sailing and detect different types of threats across the sea. The IoT sensors that are installed onto the boats are used to collect certain amount of data, for example they can retrieve the temperature, GPS position, pressure, humidity, they can detect whether doors are open or not, generate flood alarms and so on. Based on that data it is important to take different actions. And since it's important when to take these actions, a database that supports storing and aggregating the data annotated with timestamps is required. Different databases that fulfill the requirements for the project were reviewed in order to select the best one for the use case considering this article. Then a comparison between databases and testing of parameters, most importantly time efficiency and memory footprint, were made.

The paper continues as follows: Section 2 presents the problem, its background and the key parameters to evaluate the database. The experiments and results are presented in Section 3 followed by the discussion and conclusion in the last section.

2. DATA STORAGE IN EDGE DEVICES

Storing data always requires time and has a memory footprint – that means some CPU and memory usage. On top of that there are many performance problems that can arise due to several reasons that are occasionally hard to determine. The following section provides additional information for understanding the problem of storing the data and the problem itself.

2.1 Sensors, edge devices and cloud storage

In a combined fog and cloud environment the processing of data is distributed between edge devices and cloud. Processing in the cloud has no resource restrictions such as opposed to processing at the edge. To provide the best and to the user transparent experience of using cloud and edge environment, the appropriate software has to be applied to each

segment of the fog to cloud hierarchy. A similar stack is presented in Figure 1, which is similar to the one proposed by mF2C project. We focus on data management close to sensors, i.e. edge devices, such as routers or small computers like Raspberry Pi devices that store, filter and transmit data collected from IoT sensors. Beside being able to store and transmit data, edge devices can serve light-weight services and issue notifications based on thresholds. These devices are capable in variety of functions, but do not have an excess of resources, therefore the software needs to be selected carefully.

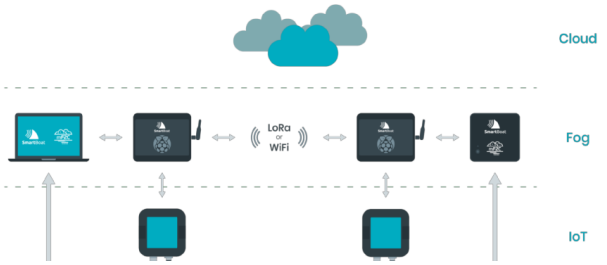

Figure 1: The proposed mF2C architecture.

2.2 The performance degradation issue

From the experience we gained by working on Fog, Edge and IoT use-cases, we found that the relational database, PostgreSQL, can became unstable and an overkill due to the lack of resources. The issue occurs when data is continuously being written into the database for a long period of time. Performance degradation seems to be a common problem in PostgreSQL [10] and sometimes hard or even impossible to solve [3]. For shorter continuous periods of recording (e.g. single day) issue does not manifest due to the small amount of the data and also because the database requires a reasonable amount of RAM. When recording lasts longer (e.g. more than one day) writing becomes slower. To solve the issue different approaches of saving and different databases were taken into consideration.

2.3 Benefits of using relational or time series databases and their comparison

2.3.1 Criteria for filtering the databases

Before performing the evaluation of the databases, a selection of the testing candidates was required, one from each type of database. Our methodology preferred databases with better support for the following attributes:

Supported platforms Applications should run on all major platforms like Linux, Windows and macOS, therefore we required to be sure that there will not be any complication for the applications to use the database.

Official Docker support Docker, currently the most popular container technology for Linux that allows creating and packaging an application along with all its dependencies, was also very important for the implementation of the services in our project.

Rapsberry Pi Our project was focused and prototyped around the Raspberry Pi, therefore the options that include this were preferred.

Supported languages More supported languages are a plus, but our main focus was on Java and Python support.

Data types The support or special/faster handling of floats and timestamps was considered as a better option.

License An open-source solution is preferred due to better flexibility and potential costs if the databases would run on a large amount of edge devices.

All databases, relational and time series, were evaluated by those parameters and the best candidates of each type were selected for the testing phase. The attributes for databases are collected in the Table 1.

2.3.2 Relational database selection process

We chose PostgreSQL as the initial relational database for our endeavors because of its standards compliance, it offering a native JSON object storage which we aspired to use elsewhere in the application and because it was easily integrated into other frameworks already in use. Other databases may also be appropriate for this purpose, however PostgreSQL proved to be the most compatible choice at that point in development. The key parameters of our comparison, based on two comparative sources [12, 2] are shown in Table 1.

2.3.3 Time series database selection process

These days, time series data applications such as sensors used in IoT analytics, are growing rapidly due to their simplicity and SQL based query language. For the comparison we have chosen 8 time series databases (Table 1) and finally selected InfluxDB as the best candidate mainly because of official Raspberry Pi Docker support. Other databases were not selected because they did not fulfill expectations regarding Docker or Raspberry Pi support (OpenTSDB, TimescaleDB), a proprietary license (Kdb+), low data type flexibility (Prometheus, RRDtool) and a lack of Java support (Graphite, Druid).

3. EXPERIMENT AND RESULTS

3.1 The test between time series and relational databases

The performance was evaluated by integration of InfluxDB into our application and comparing it with the performance of PostgreSQL.

3.2 Measurement environments

The databases could be manipulated through their own terminal clients or by libraries that provide support for different programming languages. To eliminate the probability of poorly written library or additional latencies based on the language overhead, the tests were performed in both environments – through a Java program and through the terminal with official client. According to the presented limitations, the following tests were performed: reading and writing to the InfluxDB and PostgreSQL databases using different methods like Java, bash console, reading from file and so on. So we tested the database and created a table (Table 2) showing first stage results for measuring time taken for writing and reading. All times presented in the table are for writing ten million records to database or for retrieving one million lines from the base.

Table 1: Relational and time series database feature comparison table

	Database	Supported platforms	Official Raspberry Pi Docker support	Number of supported languages	Python and Java support	Written in	Supported data types	License	Release year
RELATIONAL	OracleDB	Linux, Windows, OS X, Solaris etc.	Yes	24	Yes	C and C++	Int, float, bool, string, date etc	Proprietary	1979
	MySQL	Linux, WIndows, OS X, Solaris etc.	Yes	19	Yes	C and C++	Int, float, bool, string, date etc	GPLv2 or proprietary	1995
	Microsoft SQL Server	Linux, WIndows, OS X, Solaris	Yes	11	Yes	C++	Int, float, bool, string, date etc	Proprietary	1989
	PostgreSQL	Linux, WIndows, OS X, Solaris etc.	Yes	9	Yes	C	Int, float, bool, string, date etc.	PostgreSQL License	1989
TIME SERIES	InfluxDB	Linux, OS X	Yes	16	Yes	Go	Int64, float64, bool, string	MIT	2013
	Prometheus	Linux, Windows	No	8	Yes	Go	Only numeric data (float64)	Apache 2.0	2015
	Kdb+	Linux, Windows, OS X, Solaris	No	10	Yes	Q	Int, float, bool, string	Proprietary	2000
	OpenTSDB	Linux, Windows	No	6	Yes	Java	numeric data for metrics, strings for tags	LGPL	2011
	RRDtool	Linux, HP-UX	No	9	Yes	C	Only numeric data	GPLv2 and FLOSS	1999
	TimescaleDB	Linux, Windows, OS X	No	8	Yes	C	Int, float, bool, string, date etc.	Apache 2.0	2017
	Graphite	Linux, Unix	No	2	Only Python	Python	Only numeric data	Apache 2.0	2006
	Druid	Linux, OS X, Unix	No	7	Only Python	Java	Numeric and strings	Apache 2.0	2012

Table 2: Time taken table with first-stage results.

Testing type	InfluxDB [s]	PostgreSQL [s]
Writing to database using CLI terminal	71	256
Writing to database in Java from file generated in bash	556	652
Writing randomly generated lines to database in Java	562	678
Writing randomly generated lines to database in Java structured with 3 classes and one interface	576	673
Reading from the database in Java	5.58	8.35
Reading from the database in Java using mappers	8.20	/

3.3 IoT characteristics for databases

IoT devices, especially sensors, usually have the ability of gathering the data accompanied by data analysis to detect anomalies. Those devices write to database in bursts and are often operating on a lot of data.

3.4 The experiment metrics

The metric chosen for performing the experiment is speed, measuring the time for writing a million rows to the database in chunks of 15 points, which appears to be a common request on an Edge device connected around a dozen IoT devices.

3.5 The results

The results in Figure 2 present a comparison between InfluxDB and PostgreSQL. The x-axis shows the consecutive block number of 15 records and the y-axis shows time required to save the block. While the results are dispersed a trend curve showing the expected time taken is added to the graph. The plots present three cases. From the PostgreSQL plot (Figure 2) below we can see that most of the blocks of fifteen records take 5–8 milliseconds to be stored in database, which means that time is constant most of the time with some deviations that are occurring periodically. Most of the data in InfluxDB (Figure 2) is stored to database very quickly and it takes between 0–1 milliseconds. There are also some deviations of records that take around 50 milliseconds to be stored to database. The curve that shows the average time taken to write to database is also more diverse than in PostgreSQL plot. To perform thorough testing an InfluxDB faster batching was enabled to be included into the evaluation. The threshold for storing was set to 10000 points per batch or every 200 milliseconds. The result was lower times for writing and faster program execution. However, comparing InfluxDB results with and without faster batching (Figure 2) showed similar performance. The results undoubtedly show advantage of InfluxDB over the PostreSQL focusing on time consumption. PostgreSQL has less variance in time taken for transaction, but nevertheless InfluxDB is better in the average case.

Table 3: Time for writing million lines to database

Database	Time taken [s]
InfluxDB	58.92580
InfluxDBFasterBatching	5.25127
PostgreSQL	300.48325

3.5.1 Database setup times and resource consumption

Beside runtime performance, setup time and resource consumption were measured. The database setup times, including connecting to the database and building tables and indexes is significantly faster with InfluxDB, which is evident in Table 3. The resource consumption comparison was performed by writing and querying data for PostgreSQL and InfluxDB via CLI (Command Line Interface). 100000 records were written to database using CLI and then retrieved back. The results of this test are presented in Table 4, where it is evident that InfluxDB uses less storage, prepares database

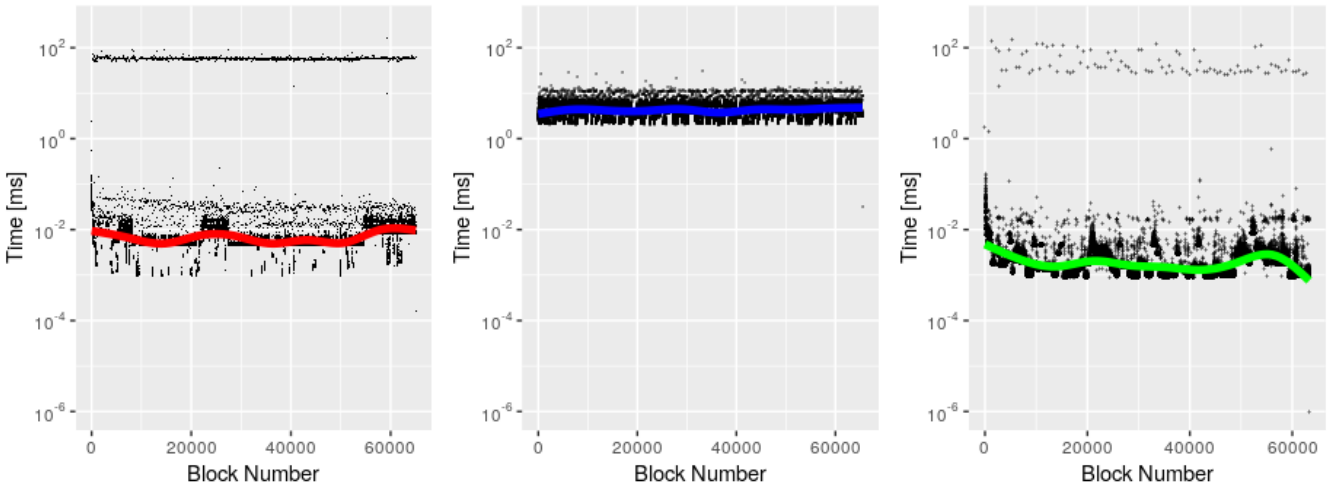

Figure 2: Plots for InfluxDB and PostgreSQL and InfluxDB faster batching.

faster, a query takes only 30 % of time. It seems that RAM and CPU consumption is higher for InfluxDB.

Table 4: Different test methods performed in CLI

	InfluxDB	PostgreSQL
Writing time	0.694 s	223.374 s
Query time	1.492 s	5.617 s
Database size	5.5 MB	15 MB
Memory usage	63.98 MB	20.53 MB
CPU usage	107 %	98.3 %

4. CONCLUSION

This paper presented the approach towards comparing relational and time series databases including comparing characteristics and performance. We explored several time series databases and their use related to the Cloud, sensors and IoT. The results indicates why time series database can be a better solution when it comes to storing IoT-generated data. We concluded that InfluxDB is a more suitable option for handling data gathered from IoT sensors and is also significantly faster in comparison to a relational database.

5. ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 730929.

6. REFERENCES

[1] A. Bridgwater. Iot time series data is 'of the hour'. https://internetofbusiness.com/iot-time-series-data-hour-influxdata/, Dec 2017. Accessed on 2018-09-03.

[2] DB-Engines. Db-engines ranking of relational dbms. https://db-engines.com/en/ranking/relational+dbms, Sep 2018. Accessed on 2018-09-05.

[3] J. DiLallo. Solving a postgres performance mystery. https://medium.com/flatiron-engineering/solving-a-postgres-performance-mystery-51544ceea584, Apr 2018. Accessed on 2018-09-06.

[4] J. W. Flory. How time-series databases help make sense of sensors. https://opensource.com/article/17/8/influxdb-time-series-database-stack, Aug 2017. Accessed on 2018-09-03.

[5] Horizon2020. mf2c project. http://www.mf2c-project.eu/, Jan 2017. Accessed on 2018-09-05.

[6] R. Kumar. 4 steps to select the right database for your internet of things system. https://thenewstack.io/4-steps-to-select-the-right-database-for-your-internet-of-things-system/, Apr 2018. Accessed on 2018-09-12.

[7] M. Risse. The new rise of time-series databases. https://www.smartindustry.com/blog/smart-industry-connect/the-new-rise-of-time-series-databases/, Feb 2018. Accessed on 2018-09-05.

[8] D. G. Simmons. Pushing iot data gathering, analysis, and response to the edge. https://dzone.com/articles/pushing-iot-data-gathering-analysis-and-response-to-the-edge, Apr 2018. Accessed on 2018-09-02.

[9] E. Siow, T. Tiropanis, and W. H. Xin Wang. Tritandb: Time-series rapid internet of things analytics. https://arxiv.org/abs/1801.07947v1, Jan 2018. Accessed on 2018-09-13.

[10] A. Vorobiev. Performance degradation of inserts when database size grows. https://www.postgresql.org/message-id/BANLkTi%3DVKBmRLVLDjy8qxpWx_6-rmbUaXg%40mail.gmail.com, May 2011. Accessed on 2018-09-04.

[11] D. G. Waddington and C. Lin. A fast lightweight time-series store for iot data. https://arxiv.org/abs/1605.01435, May 2016. Accessed on 2018-09-12.

[12] Wikipedia. Comparison of relational database management systems. https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems, Sep 2018. Accessed on 2018-09-05.