Comparison of Relational and Time-Series Databases for Real-Time Massive Datasets

E. Musa[*], G. Delač [*], M. Šilić [*] and K. Vladimir[*]
[*] University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
ema.musa26@gmail.com, {goran.delac, marin.silic, klemo.vladimir}@fer.hr

***Abstract* - Many contemporary technical systems, like IoT environments, are designed to work perpetually and collect data in a form of time series. To enable efficient data aggregation and analysis with time interval constraints, a special type of databases optimized for time-series has been developed. In this paper, we conduct a thorough performance comparison of a time-series database InfluxDB and a widely used object-relational database PostgreSql. We present our findings and outline scenarios in which a certain database has a performance advantage.**

Keywords - time-series databases; performance analysis; InfluxDB; PostgreSql

I. Introduction

Historically, time-series data has mostly been associated with applications in finance, but with recent increase in popularity of Internet of things (IoT) platforms, the importance of time-series data has grown as well. Nowadays, time-series data exists in various areas, such as financial markets, weather forecasting, medical and biological experiments and more [1][2][3][4]. Large amounts of data exchanged between servers, applications and sensors in a particular time order resulted in time-series databases becoming a standard for storing and retrieving data. A time-series database (TSDB) is a database type which is optimized for time-series and time-stamped data [5].

In this paper we describe main characteristics of time-series databases and cases where it is more efficient to use time-series databases as well as cases where using relational databases has shown better results. Using Visual Studio .NET environment, pgAdmin platform and InfluxDB command line we measured the duration of querying data from a relational database PostgreSQL and a time-series database InfluxDB. We compared the time needed for: querying data with a certain time interval constraint, data aggregation with a time interval constraint, data aggregation with a time interval constraint grouped by time intervals, data aggregation with a time interval constraint grouped by indexed attributes and finally, inserting data into a database. Given results show that a relational database holds an advantage over a time-series database when it comes to querying raw, non-aggregated data. On the other hand, the main advantage of using a time-series database over a relational database includes faster aggregation and faster result grouping by various time intervals.

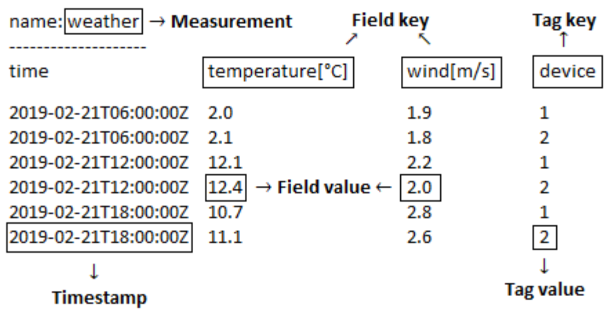
The rest of the paper is structured as follows: In Section 2 we describe main characteristics of time-series databases focusing on InfluxDB. In Section 3 we describe the experimental setup which includes PostgreSQL and InfluxDB database configurations. The results of measurements are presented in Section 4 and finally, the conclusion is given in Section 5.

II. Related Work

A time-series database is a database type which is optimized for time-series and time-stamped data. Time-series include measurements or events that are tracked, monitored, sampled and aggregated over time. They can be related to server metrics, application performances, network data, sensor data and other types of analytics data [6].

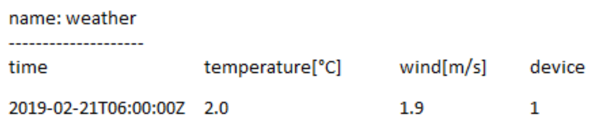
A time-series database is optimized for measuring change over time. It is built for handling metrics, events and measurement that are time-stamped and it allows users to create, update, destroy and organize various time-series more efficiently. The key difference between time-series database and a relational database is in the way data ordering is managed. While relational databases do not imply a particular order of inserted data (order by clause must be used), time-series databases index and process data with goal to optimize queries that imply ordering in time. Thus, one of the main differences between the two is the way indices are implemented. Otherwise, time-series and relational databases are architecturally similar. Other notable differences include the tradeoffs when it comes to adhering to ACID properties versus performance. Time-series databases do not need to necessarily ensure durability and strong consistency as time-series data is usually immutable and generated in a unique point in time. In addition, it is not necessary to collect all the possible data, e.g. when continuously collecting a data from a sensor, it is may not be necessary to store all data points. For instance, InfluxDB provides eventual consistency.

Nowadays, an increasing number of companies are generating large streams of metrics and events prompting the need for time-series databases in order to facilitate efficient data access [5]. However, time-series databases, due to their optimizations, do not always show favorable properties when it comes to tackling more general purpose tasks. From an engineering standpoint, it is important to understand in which use cases time-series databases perform better when it comes to query execution time, as well as general resource consumption. Some research effort has been devoted into addressing these issues [4][7][8]. For instance, in [8] the authors test InfluxDB



time	temperature[°C]	wind[m/s]	device
2019-02-21T06:00:00Z	2.0	1.9	1
2019-02-21T06:00:00Z	2.1	1.8	2
2019-02-21T12:00:00Z	12.1	2.2	1
2019-02-21T12:00:00Z	12.4	2.0	2
2019-02-21T18:00:00Z	10.7	2.8	1
2019-02-21T18:00:00Z	11.1	2.6	2

Figure 2. Data structure



name: weather

time	temperature[°C]	wind[m/s]	device
2019-02-21T06:00:00Z	2.0	1.9	1

Figure 3. Point

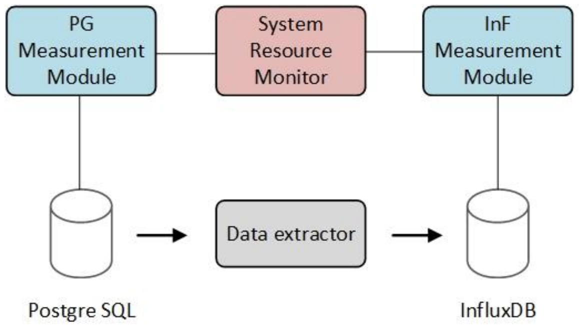


Figure 1. Experimental Environment Architecture

performance on a very large real-world dataset containing 450 million power consumption readouts. The authors focus the tests on data insertion speed and disk storage consumption parameters. In our paper, we aim to further strengthen the results found in the literature by performing additional experiments on a real-world dataset. Apart from the data insertion experiments, we also conducted several data aggregation experiments.

Currently, the one of the most popular time-series databases is InfluxDB [9] and as such, we have decided to utilize it for the purposes of performance comparison with a relational database. InfluxDB is an open-source schemaless time-series database with a set of optional components. It is written in Go programming language and it is optimized to handle time-series data in particular. The database environment provides InfluxQL, a SQL-like query language. The open-source version TICK Stack provides full time-series database platform including the InfluxDB core [5].

All data in the InfluxDB database has a column named *time*. Column *time* stores timestamps in RFC3339 UTC format and is associated with particular data. Another required part of InfluxDB's data structure are fields which consist of a field key and field value. Field keys are strings and they store metadata. Field values can be of type string, float, integer or Boolean and they represent the data. The collection of field-key and field-value pairs form a field set. Fields are not indexed and they are a required piece of InfluxDB's data structure. Tags are optional, and they consist of a tag key and tag value which are both strings that store metadata. Different combinations of all the tag-key and tag-value pairs form a tag set. Unlike fields, tags are indexed and optional. Measurement acts as a container for tags, fields and the *time* column and the measurement name represents the description of the data that is stored in associated fields. A single measurement can belong to different retention policies. Retention policy describes how long InfluxDB keeps the data. In an InfluxDB database, a series is a collection of data that share a retention policy, measurement and tag set. Finally, a point is the field set in the same series with the same timestamp [10].

In the example shown in Fig. 1, *weather* represents the measurement name. Columns *temperature* and *wind* form fields with field keys *temperature* and *wind*. Values 2.0-11.1 are field values referring to *temperature* and values 1.9-2.6 are field values referring to *wind*. Column *device* is a tag with a tag key *device* and two tag values (1 and 2). An example of an InfluxDB point is presented in Fig. 2.

In a general sense, an InfluxDB measurement is similar to a table of a relational database, tags are similar to indexed columns, fields are similar to unindexed columns and points are similar to table rows.

III. Experimental Setup

In this section, we describe the experimental setup used to perform measurements on the PostgreSQL and InfluxDB databases.

A. *Environment*

All measurements were performed on a 64-bit computer with an Intel Core i5 processor and 12 gigabytes of RAM available.

In the conducted experiment, we have collected a real-world real-time dataset consisting of twenty million records and have stored it in a PostgreSQL table. Time differences between two records in the table varied depending on numerous parameters and can vary from milliseconds to days. Specifically, the table stored twenty million records and had thirty-nine attributes. The data has a time range from December 8th, 2017 at 19:11:49 to January 12th, 2018 at 22:59:52 (UTC). There is no recorded data from December 25th, 2017 to January 1st, 2018. The first most significant attribute contained a timestamp which corresponded to the moment in time in which the record was received and the second most significant attribute contained an object identifier. We created indexes on the mentioned columns.

The architecture of the implemented experimental environment is presented in Fig. 3. In order for the InfluxDB database to contain the same data as the PostgreSQL database, a *Data extractor* module has been created. Its purpose is to retrieve records from PostgreSQL, create an InfluxDB point depending on a received record and insert the point into the InfluxDB measurement. Indexed columns are equivalent to tags inside an InfluxDB measurement which means that the tag is going to store the same object identifiers stored inside the mentioned indexed column. The rest of the data was stored in fields, except for timestamps which were stored in a special *time* column.

Measurement module has been implemented in order to facilitate measuring operations of query execution time and to monitor general system resource consumption. It consists of three submodules: a *PostgreSQL Measurement* module which measures PostgreSQL query execution time, an *InfluxDB Measurement* module which measures InfluxDB query execution time and a *System Resource Monitor* which measures system performances during query executions. The first two submodules create a stopwatch instance, starting it before executing the query and then stopping it when there are no more query results to read. The execution time is measured in milliseconds. The third submodule is started simultaneously with one of the submodules which measure query execution time. It measures the processor utilization and the available memory during the query execution.

B. Experiments

The following experiments were performed to compare the performance of databases in specific working conditions. After each experiment the machine was restarted to ensure equal testing conditions.

1) Querying data

This experiment was designed to check a performance of a standard query, without data aggregation, on several time intervals. Specifically, 3 interval lengths were selected:

- Interval 1: December 9th, 2017 to December 10th, 2017 (one day)
- Interval 2: December 9th, 2017 to December 16th, 2017 (one week)
- Interval 3: December 9th, 2017 to December 23rd, 2017 (two weeks)

Experiments were performed five times each, with the average values plotted in Fig. 4. The SQL query used for this experiment is given below (InfluxQL query is similar):

```
SELECT timestamp, coilid
FROM positionsample
WHERE timestamp >= '2017-12-09' AND
timestamp < '2017-12-10'
```

2) Aggregating data

The aggregation experiment was designed to test how the databases perform when an aggregation function is used on the data selected from a specific time interval. In this experiment, the average value of a PostgreSQL column and an InfluxDB field was selected. The following intervals were used:

- Interval 1: December 9th, 2017 to December 16th, 2017 (one week)
- Interval 2: December 9th, 2017 to December 23th, 2017 (two weeks)
- Interval 3: December 9th, 2017 to January 9th, 2018 (one month)

Experiments were performed five times each, with the average values plotted in Fig. 5. The SQL query used for this experiment is given below (InfluxQL query is similar):

```
SELECT AVG(position)
FROM positionsample
WHERE timestamp >= '2017-12-09' AND
timestamp < '2017-12-16'
```

3) Aggregating data grouped by time intervals

This experiment was performed to evaluate how the aggregation function performs on a data selected from a certain time interval when it is necessary to group the data by a certain parameter. In this experiment, the data was groped by the hour, as presented in the query below. The same intervals were used as in the experiment described in Section 3.B.2.

Experiments were performed five times each, with the average values plotted in Fig. 7. The SQL query used for this experiment is given below (InfluxQL query is similar):

```
SELECT date_trunc('hour', timestamp),
AVG(position)
FROM positionsample
WHERE timestamp >= '2017-12-09' AND
timestamp < '2017-12-10'
GROUP BY 1
```

4) Aggregating data grouped by indexed attribute/tag

The following experiment is set up in the same way as the aggregation experiment described in Section 3.B.3. However, in this experiment, the data was grouped using an indexed attribute (PostgreSQL) and a tag (InfluxDB). The same intervals were used as in the experiment described in Section 3.B.3.

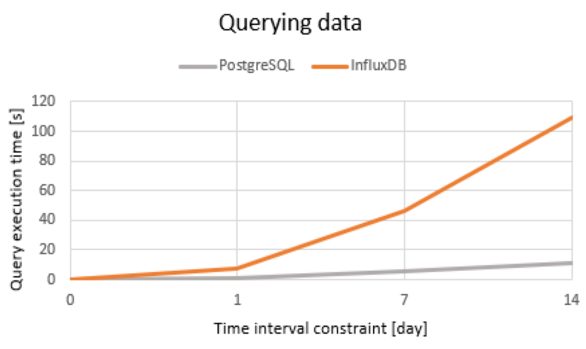
Experiments were performed five times each, with the average values plotted in Fig. 8. The SQL query used for this experiment is given below (InfluxQL query is similar):

```
SELECT AVG(gapdriveside)
FROM positionsample
WHERE timestamp >= '2017-12-09' AND
timestamp < '2017-12-16'
GROUP BY equipmentoid
```

Here it is important to note that the `equipmentoid` has an index in PostgreSQL and is a tag in the InfluxDB.

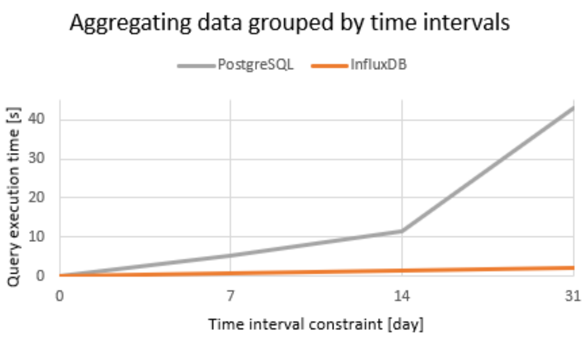
5) Inserting data

Experiments were performed five times each, with the average values plotted in Fig. 9. For both PosgreSQL and InfluxDB the data was inserted in batches (bulk insert) and the batch size was 1000 data points. There were 3 test cases each containing 500, 5000 and 50000 data points. After each experiment the machine was restarted to ensure equal testing conditions.



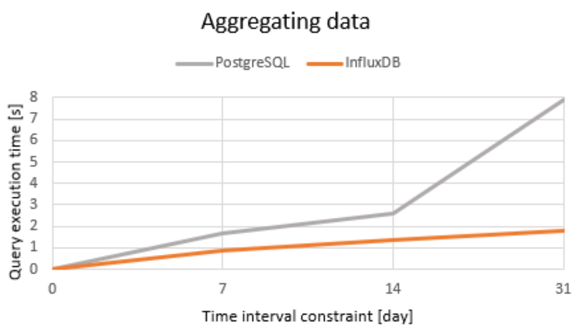
Standard deviation			
DB	1	7	14
PostgreSQL	0.071 s	0.287 s	0.565 s
InfluxDB	0.121 s	0.393 s	0.535 s

Figure 4. Querying data



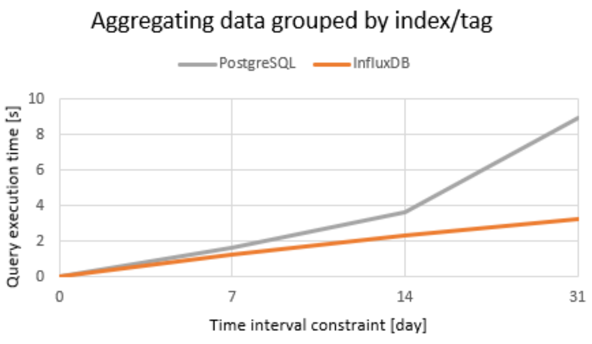
Standard deviation			
DB	7	14	31
PostgreSQL	0.037 s	0.038 s	1.741 s
InfluxDB	0.014 s	0.036 s	0.055 s

Figure 7. Aggregating data grouped by time intervals



Standard deviation			
DB	7	14	31
PostgreSQL	0.065 s	0.010 s	0.010 s
InfluxDB	0.015 s	0.015 s	0.071 s

Figure 5. Aggregating data



	Time interval [day]		
DB	7	14	31
PostgreSQL	0.012 s	0.012 s	0.072 s
InfluxDB	0.004 s	0.018 s	0.033 s

Figure 8. Aggregating data grouped by indexed attribute/tag

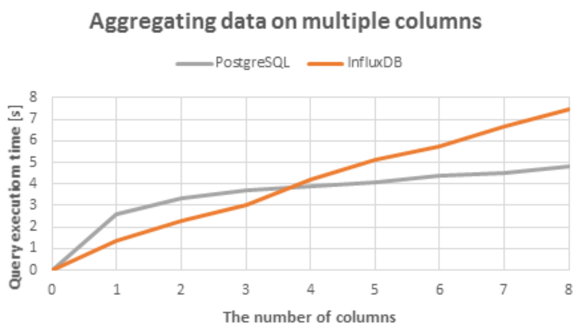
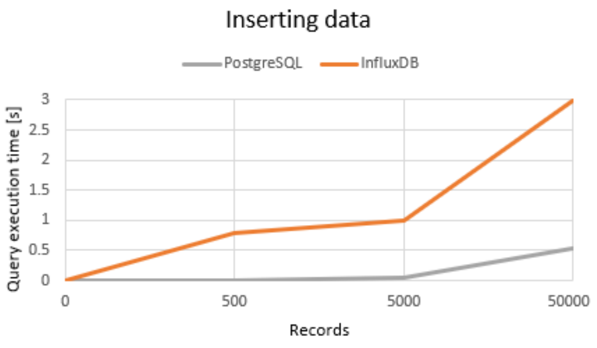


Figure 6. Impact of number of columns on performance



Standard deviation			
DB	500	5000	50000
PostgreSQL	0.002 s	0.004 s	0.031 s
InfluxDB	0.024 s	0.014 s	0.070 s

Figure 9. Inserting data

IV. RESULTS AND DISCUSSION

A. Querying data

Querying raw, non-aggregated data with a certain time interval constraint from a single column/field is more efficient using the PostgreSQL database as indicated by the shorter execution time. In addition, the InfluxDB database consumes more processor time and memory. Comparison of average query execution time for this case is presented in Fig. 4. The standard deviations for the measured values are relatively small and are therefore presented in the table below the figure. With a time range of one day to two weeks, query execution time is 83.52%-89.42% shorter in the case of using PostgreSQL database compared to using InfluxDB database. Since the results are not required to be in a particular order (in this case ordered by time), the result is within expectations.

PostgreSQL is fast and well optimized in selecting unordered data.

B. *Aggregating data*

Aggregating data with a time interval constraint from a single column/field is more efficient using the InfluxDB database due to exhibited shorter execution time. Comparison of average query execution time for this case is presented in Fig. 5 with standard deviations presented below the figure. By increasing the number of queried columns/fields, certain changes in InfluxDB performance occur. With each additional queried field, query execution time rises almost linearly. This effect can be seen in Fig. 6. (data for the experiment having a 2 week time interval is presented). For a smaller number of queried columns, PostgreSQL spends more time on query execution than InfluxDB. However, with an increasing number of queried columns, query execution time increases slower in comparison to InfluxDB and eventually query execution time becomes shorter compared to InfluxDB. With a time range of one week to one month, query execution time is 46.87%-77.62% shorter in case when InfluxDB database is used compared to using PostgreSQL database. This effect shows that PostgreSQL performs better in cases when it is necessary to perform simultaneous aggregation over multiple fields.

C. *Aggregating data grouped by time intervals*

Aggregating data with a time interval constraint from a single column/field and with result grouping by time interval is more efficient using InfluxDB. Comparison of average query execution time for this case is presented in Fig. 7 with standard deviations presented below the figure. Querying data with the same time interval constraint but with different time grouping (e.g. by day and by week) doesn't significantly affect InfluxDB query execution time but it significantly affects PostgreSQL query execution time. Shorter grouping time interval results in longer PostgreSQL query execution time. With a time range of one week to one month, query execution time is 76.82%-95.30% shorter in the case of using InfluxDB database compared to using PostgreSQL database.

D. *Aggregating data grouped by indexed attribute/tag*

Aggregating data with a time interval constraint from a single column/field and with result grouping by indexed attribute/tag is more efficient using the InfluxDB database. It is also visible that the execution times are lower in both experiments than it was the case in the previous experiment. Comparison of average query execution time for this case is presented in Fig. 8 with standard deviations presented below the figure. With a time range of one week to one month, query execution time is 24.76%-63.20% shorter in the case of using InfluxDB database compared to using PostgreSQL database.

E. *Inserting data*

Inserting records/points into a table/measurement is more efficient using the PostgreSQL database due to shorter execution time in the performed experiment, using batch size of 1000. Comparison of average inserting time is presented in Fig. 9 with standard deviations presented below the figure. With a range of five hundred to fifty thousand records/points, inserting time is shorter by 81.90% - 98.45% in case of using PostgreSQL database compared to using InfluxDB database. The main conclusion of this experiment is that writing performance of InfluxDB requires careful consideration and fine-tuning. It is recommended to use larger batches, when applicable, as this can improve the writing performance.

F. *System resource monitoring*

Selecting values from a PostgreSQL column with a time interval constraint of two weeks resulted in processor utilization maximum increase of 35.43% when compared to the idle state, presented in Fig. 10. The query hasn't significantly affected the memory usage as shown in Fig. 11. Selecting values from an InfluxDB field with the same time interval constraint resulted in a maximum processor utilization increase of 50.69% presented in Fig. 12 and a maximum drop in the available megabytes of 63.21% presented in Fig. 13.

Selecting the average value of a PostgreSQL column with a time interval constraint of one month resulted in processor utilization increase of 60.82% at most compared to the idle state. The query hasn't significantly affected the memory usage. Selecting the average value of an InfluxDB field with the same time interval constraint hasn't significantly affected general system resource consumption.

Selecting the average value of a PostgreSQL column with a time interval constraint of one month and the result grouping by one day resulted in a maximum processor utilization increase of 58.56% and a maximum decrease of the available megabytes of 2.56% when compared to the idle state. Selecting the average value of an InfluxDB field with the same time interval constraint and result grouping hasn't significantly affected general system resource consumption.

Finally, selecting the average value of a PostgreSQL column with the time interval constraint of one month and the result grouping by indexed attribute resulted in a maximum processor utilization increase of 59.89% when compared to the idle state. The query hasn't significantly affected the memory usage. Selecting the average value of an InfluxDB field with the same time interval constraint and result grouping hasn't significantly affected general system resource consumption.

V. CONCLUSION

One of the advantages of using a time-series database InfluxDB over using a relational database PostgreSQL is the shorter time needed to execute aggregation functions over data. Test cases described in Section 4 show that the execution time for the operation of aggregation with time interval constraints is 46.87%-77.62% shorter in the case of using InfluxDB when compared to using PostgreSQL database. Another advantage includes faster result grouping by various time intervals without creating additional indexes as well as result grouping by indexed attributes/tags. In the mentioned test cases, aggregation

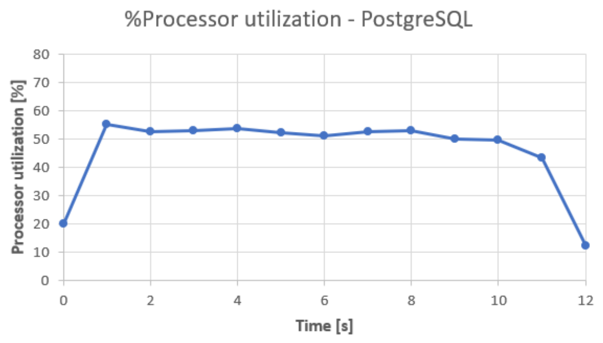


Figure 10. Processor utilization - PostgreSQL

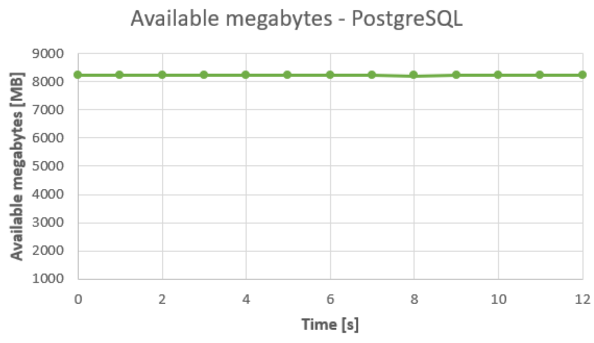


Figure 11. Available megabytes – PostgreSQL

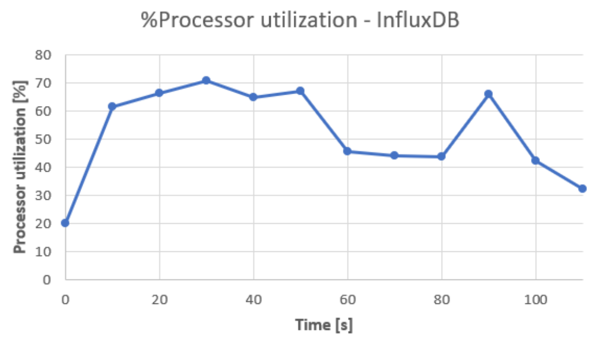


Figure 12. Processor utilization - InfluxDB

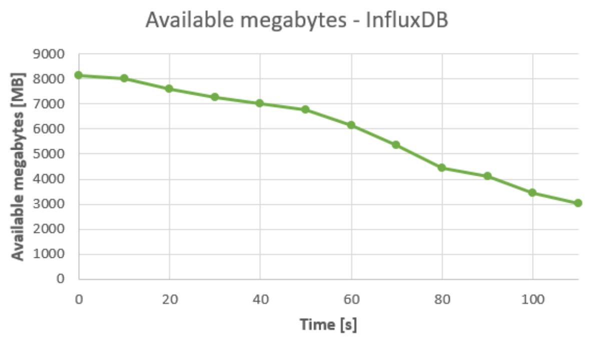


Figure 13. Available megabytes - InfluxDB

execution time with a time interval constraint and result grouping by time interval is 76.82%-95.30% shorter and aggregation execution time with time constraint and result grouping by indexed attribute/tag is 24.76%-63.20% shorter in the case of using InfluxDB compared to the use of the PostgreSQL database. These characteristics represent the main advantages in using time-series databases because massive datasets are mostly useful to users when they are aggregated and grouped.

The disadvantage of using the InfluxDB database lies in slow querying of non-aggregated data. Querying data with a time interval constraint results in an 83.52%-89.42% shorter query execution time in the case of using PostgreSQL compared to using InfluxDB database.

Using time-series databases greatly benefits users whose main concern is processing a large amount of data with time interval constraints in a short amount of time. A large amount of available memory is desirable for optimal InfluxDB database usage.

The future research efforts will focus on checking how the batch size impacts the performance of InfluxDB insertion operations. Additionally, it would be beneficial to explore how the varying data set size impacts the performance of the experiments presented in this paper.

ACKNOWLEDGMENT

This work has been partially supported by the Croatian Science Foundation under the project: Reliable Composite Applications Based on Web Services (IP-01- 2018-6423) and by the European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS).

REFERENCES

[1] Feng, Y., Chi, L., Chi, H., Liu, C., Liu, Z., QoM: An Effective Querying Method for Time Series Database, Hangzhou, (2012), 1-1

[2] Dunning, T., Friedman, E., Time Series Databases: New Ways to Store and Access Data, O'Reilly Media, Incorporated, 2014

[3] Balis, B., et al., Towards an operational database for real-time environmental monitoring and early warning systems, Procedia Computer Science, Vol. 108, pp. 2250-2259, 2017

[4] Tahmassebpour, M.,: A New Method for Time-Series Big Data Effective Storage, IEEE Access, vol. 5, pp. 10694-10699, 2017

[5] Naqvi, S., Yfantidou, S., Advanced Databases: Time Series Databases and InfluxDB, Bruxelles: Université libre de Bruxelles, 2017

[6] Dix, P., Why Time-Series Matters For Metrics Real-Time and Sensor Data: What is time-series data, 2016

[7] Rhea, S., et al., LittleTable: A Time-Series Database and Its Uses, In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17), New York, NY, USA, pp. 125-138, 2017

[8] Zaina, A., Reinhardt, A., Huchtkoetter J., Relational or Non-Relational?: A Comparative Evaluation of Database Solutions for Energy Consumption Data, Proceedings of the Ninth International Conference on Future Energy Systems (e-Energy '18). ACM, New York, pp. 474 – 476, 2018

[9] DB-Engines Ranking of Time Series DBMS, https://db-engines.com/en/ranking/time+series+dbms, February 2019

[10] InfluxDB key concepts, https://docs.influxdata.com/influxdb/v1.5/concepts/key_concepts/, February 2019