UNIVERSITY
OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Mikael Martinviita

Time series database in Industrial IoT and its testing tool

Master's Thesis
Degree Programme in Computer Science and Engineering
October 2018

Martinviita M. (2018) Time series database in Industrial IoT and its testing tool. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 62 p.

ABSTRACT

In the essence of the Industrial Internet of Things is data gathering. Data is time and event-based and hence time series data is key concept in the Industrial Internet of Things, and specific time series database is required to process and store the data. Solution development and choosing the right time series database for Industrial Internet of Things solution can be difficult. Inefficient comparison of time series databases can lead to wrong choices and consequently to delays and financial losses. This thesis is improving the tools to compare different time series databases in context of the Industrial Internet of Things. In addition, the thesis identifies the functional and non-functional requirements of time series database in Industrial Internet of Things and designs and implements a performance test bench. A practical example of how time series databases can be compared with identified requirements and developed test bench is also provided. The example is used to examine how selected time series databases fulfill these requirements.

Eight functional requirements and eight non-functional requirements were identified. Functional requirements included, e.g., aggregation support, information models, and hierarchical configurations. Non-functional requirements included, e.g., scalability, performance, and lifecycle. Developed test bench took Industrial Internet of Things point of view by testing the database in three scenarios: write heavy, read heavy, and concurrent write and read operations. In the practical example, ABB's cpmPlus History, InfluxDB, and TimescaleDB were evaluated.

Both requirement evaluation and performance testing resulted that cpmPlus History performed best, InfluxDB second best, and TimescaleDB the worst. cpmPlus History showed extensive support for the requirements and best performance in all performance test cases. InfluxDB showed high performance for data writing while TimescaleDB showed better performance for data reading.

Keywords: time series database, industrial iot, performance testing, test bench

Martinviita M. (2018) Aikasarjatietokanta teollisuuden esineiden internetissä ja sen testipenkki. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 62 s.

TIIVISTELMÄ

Teollisuuden esineiden internetin ytimessä on tiedon keruu. Tieto on aika ja tapahtuma pohjaista ja sen vuoksi aikasarjatieto on teollisuuden esineiden internetin avainkäsitteitä. Prosessoidakseen tällaista tietoa tarvitaan erityinen aikasarjatietokanta. Sovelluskehitys ja oikean aikasarjatietokannan valitseminen teollisuuden esineiden internetin ratkaisuun voi olla vaikeaa. Tehoton aikasarjatietokantojen vertailu voi johtaa vääriin valintoihin ja siten viiveisiin sekä taloudellisiin tappioihin. Tässä diplomityössä kehitetään työkaluja, joilla eri aikasarjatietokantoja teollisuuden esineiden internetin ympäristössä voidaan vertailla. Diplomityössä tunnistetaan toiminnalliset ja ei-toiminnalliset vaatimukset aikasarjatietokannalle teollisuuden esineiden internetissä ja suunnitellaan ja toteutetaan suorituskykytestipenkki aikasarjatietokannoille. Työ tarjoaa myös käytännön esimerkin kuinka aikasarjatietokantoja voidaan vertailla tunnistetuilla vaatimuksilla ja kehitetyllä testipenkillä. Esimerkkiä hyödynnetään tutkimuksessa, jossa selvitetään kuinka nykyiset aikasarjatietokannat täyttävät tunnistetut vaatimukset.

Diplomityössä tunnistettiin kahdeksan toiminnallista ja kahdeksan ei-toiminnallista vaatimusta. Toiminnallisiin vaatimuksiin sisältyi mm. aggregoinnin tukeminen, informaatiomallit ja hierarkkiset konfiguraatiot. Ei-toiminnallisiin vaatimuksiin sisältyi mm. skaalautuvuus, suorituskyky ja elinkaari. Kehitetty testipenkki otti teollisuuden esineiden internetin näkökulman kolmella eri testiskenaariolla: kirjoituspainoitteinen, lukemispainoitteinen ja yhtäaikaiset kirjoitus- ja lukemisoperaatiot. Käytännön esimerkissä ABB:n cpmPlus History, InfluxDB ja TimescaleDB tietokannat olivat arvioitavina.

Sekä vaatimusten arviointi että suorituskykytestit osoittivat cpmPlus History:n suoriutuvan parhaiten, InfluxDB:n toiseksi parhaiten ja TimescaleDB:n huonoiten. cpmPlus History tuki tunnistettuja vaatimuksia laajimmin ja tarjosi parhaan suorituskyvyn kaikissa testiskenaarioissa. InfluxDB antoi hyvän suorituskyvyn tiedon kirjoittamiselle, kun vastaavasti TimescaleDB osoitti parempaa suorituskykyä tiedon lukemisessa.

Avainsanat: aikasarja tietokanta, teollisuuden esineiden internet, suorituskyvyn mittaus, testipenkki

TABLE OF CONTENTS

FOREWORD

This master's thesis was carried out in ABB Helsinki, and the purpose of the research was to help different businesses when choosing a time series database for their use case. The research was also carried to get a more detailed overview of the time series database market.

The research consisted of three stages wherein the first stage the requirements for the time series databases were defined. In the second stage, performance test bench was designed and implemented. In the last stage with the defined requirements and implemented test bench, three databases were evaluated.

I would like to thank my technical supervisor Mika Luotojärvi and the whole ABB's cpmPlus team who provided the opportunity to do the Master's thesis and helped during the research process. The thesis had a lot of interest around it and it was a pleasure to work in a motivating environment.

I would also like to thank my principal supervisor Dr. Minna Pakanen and second examiner Dr. Simo Hosio who helped during the writing process and finalizing the thesis. Special thanks to my family who helped to support and motivate during the thesis-writing process.

Helsinki, 25.10.2018

Mikael Martinviita

ABBREVIATIONS

AI	Artificial intelligence
API	Application Programming Interface
AWS	Amazon Web Services
BDR	Bi-directional replication
BLOB	Binary Large Object
CPU	Central Processing Unit
DCS	Distributed Control System
DIRAC	Distributed Infrastructure with Remote Agent Control
HMI	Human-machine interface
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IIoT	Industrial Internet of Things
IOPS	Input/output operations per second
JBDC	Java Database Connectivity
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
M2M	Machine to machine
MQTT	Message Queuing Telemetry Transport
OBDC	Open database connectivity
OPC DA/HDA	OPC Data Access / Historical Data Access
OPC UA	OPC Unified Architecture
OS	Operating System
PIMS	Production Information Management System
RAM	Random Access Memory

REST	Representational State Transfer
SCADA	Supervisory Control and Data Acquisition
SQL	Structured Query Language
SSD	Solid-state drive
TSDB	Time series database
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
UUID	Universally unique identifier
WAL	Write ahead log
X	Data point
t	Timestamp
v	Value
q	Quality of the value

1. INTRODUCTION

Manufacturing companies have taken the first steps towards realizing their Industrial Internet of Things (IIoT) solutions. In the essence of IIoT is data gathering smart factories which is argued to be the future of factories. For data gathering purposes use case specific database is required in the IIoT solution.

IIoT solution can differ depending on the deployment context. Solutions still have in common enormous amount of data, large number of connected clients, and fast data processing. In industrial context, system liability and security are important factors.

Data gathering is heavily sensor and device based and the produced data is time and event-based. Time series data management is therefore an essential part of the IIoT solutions. Time series data requires specific data management and processing that is beyond the capabilities of traditional relational databases. As time series data can be used for multiple different kinds of purposes and in variety of environments, there are many different kinds of time series databases on the market.

Time series processing is highly essential part of systems and platforms for developing solutions for industrial internet of things. Different platforms offer a variety of possibilities to process time series data, and there are also independent components which can be integrated into a broader solution as a time series processor. Comparing these different solutions can be extremely difficult using only the public documentation, and it can lead quickly to wrong decisions. On the other hand, solution developers do not often know their requirements for their time series processing. This ignorance leads to wrong decisions and unusable solutions, or it can create long delays in product development projects when components and architecture need to be changed during the deployment phase.

Current literature includes some research around time series databases and their comparison. Most of the comparison is done without including certain usage context and hence lacking the IIoT point of view in their comparisons. There is also lack of clear requirements for time series databases in the IIoT and what are the key functional and non-functional features that the database should have in that context. Some performance benchmarking tools have been developed earlier but they all lack the IIoT point of view and use cases used for the testing does not reflect the IIoT usage.

The thesis responds to the problem of inefficient comparison and fragile knowledge of time series databases in the Industrial Internet of Things context. This thesis is to find the functional and non-functional requirements of time series database in the Industrial Internet of Things environment and improve time series database comparison methods. Knowing the requirements eases the solution development and comparison of possible database products. To help with the comparison, a performance benchmark test bench that is targeted for IIoT context is needed to be designed and implemented.

1.1. Research questions and objectives

The research questions of this thesis are the following:

- What are the functional and non-functional requirements of a time series database in IIoT environment?

- How do current time series databases fulfill these requirements?

Also, the research objectives are the following:

- Design and develop a test bench to measure the performance of a time series database
- Apply the test bench to different time series database to find out their strengths and weaknesses

1.2. Overview of the thesis

The structure of the thesis is following. First, all necessary concepts are defined, and the knowledge gap is identified in Chapter 2 and 3. The example uses cases on time series databases in IIoT are presented pointing out the critical factors for each use case in Chapter 4. Then the requirements for the time series database is constructed and identified based on related literature and interviews of ABB's employees in Chapter 5. That is followed by a design and implementation of a performance test bench for time series databases in Chapter 6. Lastly, current time series database products are evaluated based on the identified requirements and test bench is used for more precise performance comparison of three specific databases in Chapter 7. In Chapter 8 combining discussion about the previous evaluation and the evaluation of thesis objectives is provided. Conclusion of the thesis is in Chapter 9.

2. BACKGROUND

In this chapter, necessary concepts and definitions of the thesis are introduced. These include IIoT concept, time series data, time series databases with example products and lastly benchmarking. Current time series database benchmarks are introduced and evaluated for this thesis.

2.1. Industrial Internet of Things

As the thesis focuses specifically on the Industrial Internet of Things (IIoT), it is good to take a closer look at what is meant by it. In this sub-section concept is first defined and its characteristics identified. Then closer look why IIoT is raising awareness in the sense of benefits and challenges. Lastly, the current state of IIoT is inspected.

2.1.1. Definition and characteristics

Industrial Internet of Things is a field of IoT technology in which applications and solutions are targeted for the manufacturing industry. The industrial Internet Consortium [1] defines IIoT as an internet of things, machines, computers, and people, enabling intelligent industrial operations using advanced data analytics for transformational business outcomes by A more practical definition can be as a new industrial ecosystem that combines intelligent and autonomous machines, advanced predictive analytics, and machine-human collaboration to improve productivity, efficiency, and reliability. The target of IIoT is to connect embedded systems and products to create larger operational systems. [2]

IIoT and Industry 4.0 are often used together when describing the future of the industrial and manufactory field. Industry 4.0 is a broader viewpoint to the industrial field as it describes a new industrial revolution which focuses on automation, data, innovation, cyber-physical systems, processes, and people [3]. IIoT is in the core of this new revolution as it allows new communication infrastructure to the connected devices.

Similarly, as in the IoT, a central concept in IIoT is the connected devices. The main difference is that in IIoT communication is used for command and control of mission-critical information and responses. Reliability, availability, and accuracy are critical for IIoT applications. [4] Even though security is essential to consumer-based IoT, for IIoT it is a core component. Disruption to the production because of weak security can lead to loss of millions of dollars for the company. Advanced security is needed to keep the system secure and stable.

As IIoT solutions are going to be built on top of existing automation and monitor systems, there is a requirement that IIoT solutions support legacy protocols and technologies, e.g., SCADA and M2M protocols. Respectively the number of connected devices in a factory is multiple times more than in smart home which means that scalability is a key characteristic in IIoT. When there are more connected devices, there are also more data processed in the system. The developed system must be able to scale when in need. [5]

By having multiple different systems which are targeted for a different level of usage in the same factory, it makes the whole system hierarchical. Figure 1 presents an example scenario of an IIoT factory and shows the different level of systems that the factory has. Example scenario shows how there are multiple different devices and control systems connected to the edge node, where the time series database would be deployed. There can also be multiple edge nodes which can have a hierarchical relationship with each other. As stated earlier there are multiple legacy systems and protocols still in use, and many of these are not designed with current security principles and the control network is typically isolated as separate security zone with limited external connectivity. Edge nodes in IIoT need to provide secure communication so that the system and the factory are secure when connected to external endpoints, e.g., cloud.

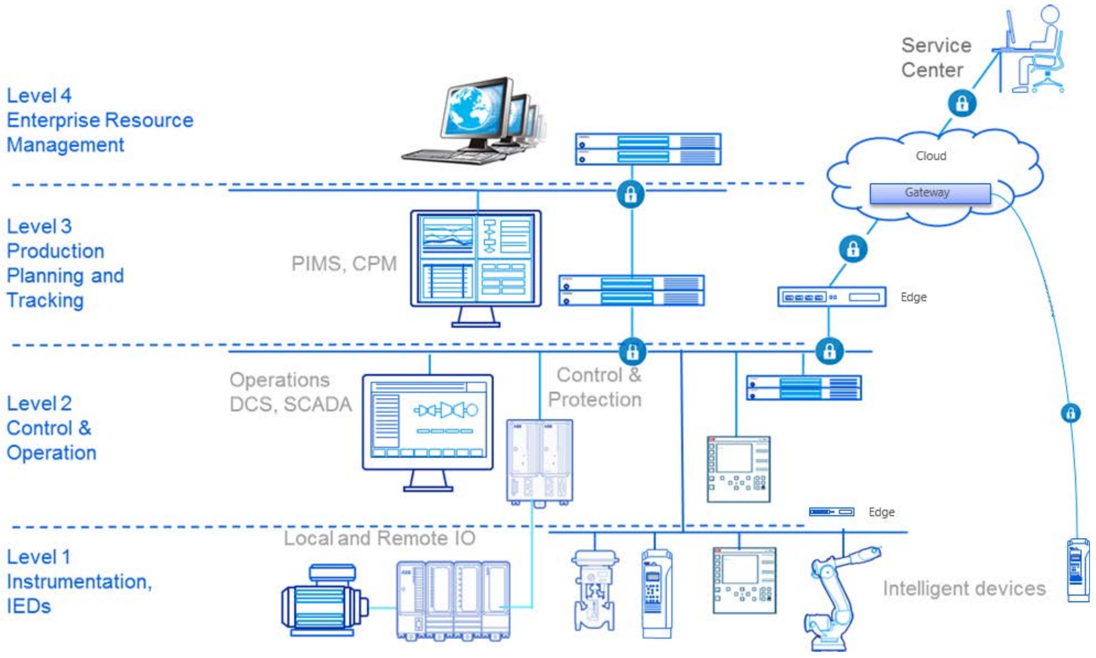


Figure 1. Hierarchical industrial system [6].

From figure 1 can also be seen that the data source is always some device, e.g., a motor, a protection relay, a robot, or a drive. A device can produce multiple values that represent different properties of the device. Semantics of the data that comes from these devices is typically known by the device manufacturer or process owner. For that reason, most of the data is structural, and information models are needed to represent these relationships in the data. Often IoT is associated with unstructured data, but it refers IoT data schematic which cannot be inserted into a traditional relational database with SQL format [7]. The data still has some structure as it has a relationship with the producing device. In the next chapter reason for why knowing the source device of the data is essential for IIoT is discussed.

Information models are data models which represent the concepts, relationships, constraints, rules, and operations of specific data for the chosen domain [8]. Information models are used to unify the system making different applications able to communicate with each other. Information models are also used to model logical entities, structure larger pieces of equipment and systems, and enable model specific analytics without detailed information about the actual physical device. One essential information model in the industrial field is time series data which represents the relationship between the measured value and the time of the measurement. One approach to information models in IIoT comes from the usage of OPC UA communication protocol which uses information models in its implementation [9].

As the IIoT has special characteristics, it creates special requirements for the systems that operate in it. Knowing these requirements when choosing or designing the system for industrial use is essential. The core of this thesis is to find these requirements for time series databases.

2.1.2. Benefits and challenges

As seen from the definition the primary goal of IIoT is to connect devices and enable digital applications and services based on the connectivity and data from the device or process it is part of. These, of course, requires successful application integration to the environment and accurate data collection and analyzing.

In more detail, IIoT may help at preventive maintenance. Adapting Big Data analytics and machine learning with the IIoT system allows users to gain more detailed data insights and predictions. Knowing the status of each component of a production line may allow detection of possible component failure before it actually happens which saves time and money for the company. [10 p.8-9] Planned component maintenance and replacement are cheaper than unscheduled maintenance because of unexpected component breakdown. Currently, 60% of performed maintenance is done unscheduled. [11] Similarly as the status of each component of the production line is known, possible bottlenecks of the production line can be identified. Optimizing the production by removing bottlenecks and keeping the system up allows the company to be more productive and efficient.

IIoT also allows companies to create new hybrid business models and find new technologies to increase innovation. As IIoT enables remote monitoring, service offerings are increasing business model for manufacturers. With more detailed knowledge of the current performance of the company, business leaders can make better decisions regarding the future of the company. In addition to production line efficiency, the logistics of the warehouse can also be enhanced. Autonomous conveyors, robots, and forklifts can be optimized and monitored with AI and machine learning. More detailed information about the production can also be used to minimize the production emissions. [12]

Even though IIoT has lots of benefits, there are still some challenges to be faced. The key challenge is to transform collected data to actual business value. Mining the right insights from the collected data so that it creates value for the user and the customer might not be as straightforward as it seems especially when the amount of collected data is enormous. Another challenge concerns the collected data and its security. As seen from past couple of years cyber-attacks against industrial systems have been rising. From the customer's point of view knowing that data is secured and who manages it is extremely valuable. These requirements create challenges even in architectural level for the developers of IIoT applications. [2] Another challenge for IIoT is the extensive use of different protocols, devices, and architectures. For efficient usage and ability to connect different devices to work together, some common protocols or middleware is needed. As the industry distributes to multiple different fields with their unique business scenarios, devices, protocols, and architectures finding this common ground can be challenging. A solution for this challenge has been tried to find through open-source development.

Another challenge for IIoT is also the cost of the IIoT solution. The implementation of an IIoT solution should decrease the cost of the production not increase it with expensive remote monitoring. The monitoring functionality itself does not provide a return for the company. [13]

2.1.3. Current state

Many of the big industrial companies have introduced their own IIoT solutions and approaches. There are also a couple of real IIoT solutions implemented and already in use. The implementation of the field theories is still in the beginning, and the field is going to grow enormously in the next couple years. According to research done in June 2016 by IndustryARC, the IIoT market will grow to $123,89 Billion by 2021 [14].

ABB offers their IIoT solution: ABB Ability, which was launched in March 2017. ABB Ability offers a unified cross-industry digital platform for devices, systems, solutions, and services. There are implementations in multiple fields, e.g., oil, gas, mining, manufacturing, microgrids, transport, and data centers including over 210 IIoT solutions. [15]

GE has, like ABB Ability, an IIoT platform Predix. Predix provides a scalable asset-centric platform which is secure and has machine learning and external application development support. GE started their digitalization already in 2012. [16] There are multiple actual implementations of the platform usage. One use case is with wind power company Exelon which uses the Predix platform for optimizing the wind predicting accuracy [17].

Bosch offers IoT Suite which is their offering for IIoT platform. IoT Suite is not just for IIoT usage but also for more general IoT usage. Currently, there are 6 million devices connected to the platform. The platform has couple use cases including use cases for factory, forklifts, and trains. [18] Other similar IIoT platforms are Condence [19], MindSphere by Siemens [20], and Watson IoT Platform by IBM [21].

A core component of these IoT platforms and IoT itself is cloud computing. Two main cloud computing providers are Microsoft Azure and Amazon Web Services (AWS). Azure provides specific support for IoT with its IoT Hub and IoT Edge products. On AWS IoT support is provided with AWS IoT Core and AWS Greengrass which implements the local edge functionality. [22]

After Hannover Messe 2018 Microsoft stated that: "*IIoT has reached the mainstream*." [23] More manufacturers are interested in using Azure to digitalize their business. Microsoft also stated that they are going to invest $5 billion in IoT during the next four years because of the high demand for their services [23]. Based on the feedback Microsoft received from manufactories at the Hannover Messe 2018 they started to develop new features to the Azure that fits better to the IIoT need. For example, some new features are OPC UA support, IoT Hub on-premise management, and new time series database. [24] The active development and discussion with manufactories indicate ongoing progress and deployment of IIoT.

Audi and Ericsson did public steps towards smart factory as they announced 5G connected smart factory trial where they start to test 5G connected wireless smart manufacturing. [25] Implementation is huge step forward especially for 5G communication, and the success of the testing can push other companies to implement their smart factory systems faster.

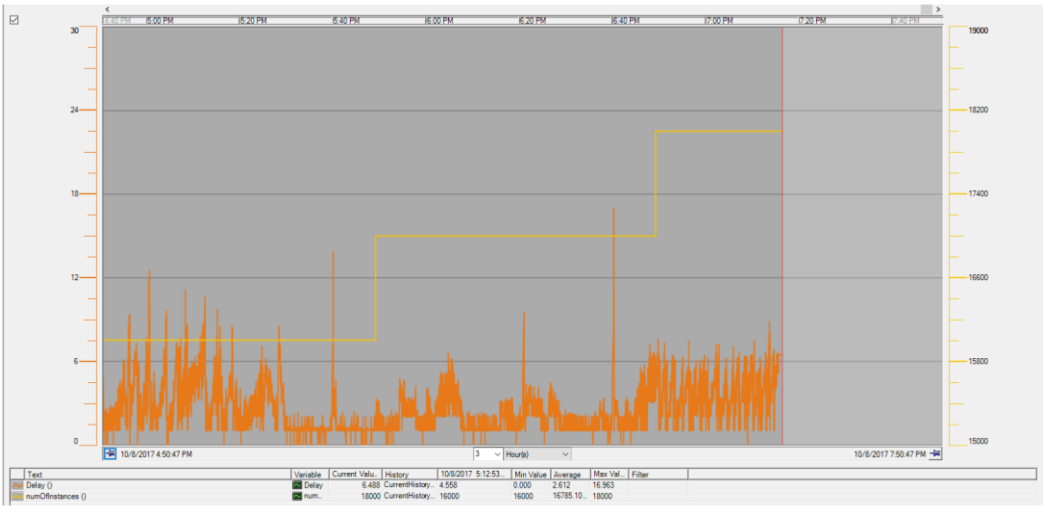
In summary, there are multiple solutions provided by leading industrial companies. There are also couple successful implementations to actual production plants. IIoT is in good progress for more widespread implementation over multiple fields.

2.2. Time series data

Structural data and usage of information models were typical in industrial systems and devices. One structural information model is time series data point which is a combination of a timestamp value(t), the actual value (v), and quality (q). Time series data point X can be present as in Equation (1).

$$X = (t, v, q) \tag{1}$$

The quality factor, for example, can indicate the reliability of the value. Data point usually has also some defining names and tags that are used to group the data points. Time series data is then a sequence of these data points creating a continues data stream. Time series data has some unique characteristics. Most of the time every data point is handled as a new entry instead of as an update for already saved data entry. This sort of handling means that the amount of time series data in the database is always growing. Another characteristic is that time series data usually is in time order when it arrives. There are exceptions for this characteristic as there can be user entered values and application calculated values. The third characteristic is time indexed data, making time the primary axis for the data. By having the time dimension in the data, it can be used to measure change over time which is the main reason to use time series data [26, 27]. Time series data is usually presented as a trend as it is the most natural interpretation of time-based data. An example of a time series data trend presentation can be seen in figure 2.



Text	Variable	Current Valu...	History	10/8/2017 5:12:53...	Min Value	Average	Max Val...	Filter
Delay ()	Delay	6.488	CurrentHistory...	4.558	0.000	2.612	16.963	
numOfInstances ()	num...	18000	CurrentHistory...	16000	16000	16785.10...	18000	

Figure 2. Time series data trend.

Time series data is used to describe the surrounding environment and for this purpose knowing the context and understanding the meaning of the time series data is essential. To make the data more easily interpretable for the user time series data usually has some metadata with it. This metadata allows more specific queries and gives a richer description of the measured source environment. Using detailed metadata creates a challenge for the data processing and data managing as the size of each data point increases. As time series data can have extreme velocity, meaning new data is received multiple times in a second, the size of the handled data is enormous. [28]

One of the most used fields for time series data has been the financial sector. Knowing the exact time of events is critical for banks and stock exchange when doing transactions. The Chicago Mercantile Exchange has around 100 million live contracts, and the derivative exchange handles around 14 million contracts per day. That amount of data produces around 1.5 to 2 million time series data points per day. [29 p.187] In industrial field data per day can grow even larger. For example, in a smart factory unit with 100 devices or sensors that produce one time series data point each second. That kind of environment produces 8.6 million time series data points per day. Another real-life example could be a situation where the quality of the electricity (voltage, current) is measured with 20 kHz sampling rate and in a single active component such as a protection relay which may contain, e.g., 10 such signals which result in 200 000 values per second. In an electricity distribution station, there can be 100 of such components which results in 17.28 billion values per day.

2.3. Time series databases on the market

As mentioned in chapter 2.2 time series data grows fast, because of its new entry and velocity characteristics. For this purpose, traditional relational databases are not suitable to handle time series data in an efficient way. Traditional relational databases are designed by the set theory where the order of the elements does not matter. For time series data the order is a key factor. Time series databases give priority for the timestamp which allows them to scale quickly as the data amount grows. Time series databases also allow multiple timestamp inserts under same key element when in relation database each timestamp insert would need a new key element. These databases also have some built-in functions for time series data, e.g., size management, online aggregation, and continuous queries. [27] Time series data has properties that differ it from other data workloads. These properties are data lifecycle management, summarization, and large range queries. To efficiently process time series data the database must support these properties. [30] Time series data can also be in many different unstructured forms which creates a requirement for the database to support semi-structured and unstructured schemas [29 p.187].

The benefits of time series databases are that they are incredibly scalable, and they have high performance as they are optimized for time series data. Time series databases also reduce operational downtime and improve business decisions as they provide data analyzes in real time. [31] Time series databases have gained popularity during the last couple of years and are one of the most trending database types in use [32]. Next couple of time series databases are introduced in more detail.

cpmPlus History

cpmPlus History is ABB's own time series database engine written in C++ and C#. cpmPlus History is targeted for process history data gathering and analytics. It is a highly scalable software platform which is targeted for products and system for manufacturing process industries and utilities. The core of the cpmPlus History is RTDB which is a relational database. RTDB is designed and optimized for time series data management by having built-in columnar features. cpmPlus History is currently on version 5.1, and it is part of multiple ABB solutions e.g.,

- Process information and reporting systems
- Energy management systems
- Long term history embedded in control systems
- Power plant information and condition monitoring systems.

cpmPlus History provides the standard data models for process information, events/alarms, and equipment model, all with granular role-based access control. The engine also provides stream processing, alarm detection, and online aggregation. Equipment model is user-defined data model which allows the creation of application specific data models. Database engine also implements data acquisition, redundant data storage, hierarchical system structure, calculation tools, public interfaces including e.g. OPC UA and SQL, and visualization.

InfluxDB

Lately, an open-source time series database InfluxDB has gained popularity. InfluxDB is a relatively new database product as its development started 2013 and it is currently on version 1.6. It is written in Go and targeted for collecting, storing, monitoring, visualizing, and alerting of time series data. InfluxDB has two versions; free open-source and paid enterprise version. The enterprise version brings clustering, manageability and security features on top of the open-source version. The database offers interfaces for Chronograf, Kapacitor, and Telegraf for the interface, stream processing, and metrics and event collecting. There is also a possibility to run InfluxDB on the cloud which is hosted by InfluxData.

Data schema in InfluxDB consists of a timestamp and set of key-value pairs. Key-value pairs have value fields and tags, and the possible value types are strings, floats, integers, and Booleans. The only restriction in data schema is that data point must have timestamp and at least one field key-value pair. Data are grouped into so-called 'measurements' which act like an SQL database table. The database has built-in functions for aggregation, selection, transforming, and prediction. InfluxDB does not support native SQL for querying, but it has SQL-like query language InfluxQL. API's for InfluxDB that are provided by InfluxData are only HTTP API and Go API library. Other language API's are external open-source API's. Hardware recommendations scale based on the load. In a single node solution for low load hardware recommendations are 2-4 CPU cores, 2-4 GB RAM, and 500 IOPS when in contrast for the high load they are 8+ CPU cores, 32+ GB RAM, and 1000+ IOPS. [33]

Microsoft Time Series Insight

Microsoft Time Series Insight is time series database in Microsoft's Azure cloud computing platform. It was released on April 2017. Time series Insight contains functionality for analytics, storage, and visualization. The database has functionality for SQL-like filtering and aggregating, and the ability to construct, visualize, compare, and overlay time series patterns. There are two versions of the database: smaller limit of storage (30 GB) per unit and daily ingress (1 GB) version which cost around $100 per month and expanded version with a storage limit of (300 GB) per unit and daily ingress (10 GB) which costs over $1000 per month. Time Series Insight provides REST API to access the database. Data is sent in JSON format, and the database's data schema is configurable. The timestamp is the only mandatory item for the schema and produced data items. [34, 35]

Vertica Analytics Platform

Vertica Analytics Platform is a commercial solution designed for Big Data analytics. Vertica company was founded 2005 and platform is currently on version 9.0. The platform has column-oriented storage which is capable of handling time series data. Data schema uses traditional SQL based table format. The database has built-in machine learning functionality which allows outline detection, missing values, normalizing, sampling, clustering, and classification. There are also built-in time series analysis functions, e.g., missing values, interpolation, and filtering. The platform can be self-hosted or run on third-party platforms, e.g., Microsoft Azure, Amazon Web Services, or Hadoop. There are two editions of Vertica where the extended premium version allows, e.g., machine learning and advanced time series analytics. Data can be accessed through SQL, ODBC, JDBC, and ADO.NET [36].

OpenTSDB

OpenTSDB is a time series daemon which works on top of HBase database. OpenTSDB is open source, written in Java, and it was released in 2010. Currently, OpenTSDB in on version 2.3. The data schema is optimized for fast aggregations of similar data to minimize the used storage space. Data point consist of the metric name, timestamp, actual value, and tags. Value can be a 64-bit integer or single-precision floating point or JSON formatted event. OpenTSDB provides telnet-style protocol data queries and HTTP API. As OpenTSDB is built on top of HBase, the time series database supports scalability and replication. [37, 38]

KairosDB

KairosDB is an open source time series database written in Java on top of Cassandra database. Development started 2013, and it is currently on version 1.2.1. Data is accessed through Telnet or REST API. Data point consists of the metric name, timestamp, type, value, and tags. Value can be either long or double type. KairosDB has multiple built-in functions for aggregating, grouping, and filtering. There is also roll-up functionality support which allows creating new aggregated data value points based on existing data points. As KairosDB uses Cassandra database, it also supports replication and distribution. [39]

Elasticsearch

Another open source solution is Elasticsearch which is distributed search and analytics engine. It is built on Apache Lucene. Development of Elasticsearch started in 2010, and it is currently on version 6.2.4. Elasticsearch support near real-time reading which allows it to be used as NoSQL database. Elasticsearch uses JSON formatted documents as it datatypes which allow users to create their own data schema. As Elasticsearch is mainly a search and analytics engine, it has support for extended querying with multiple aggregating, filtering, and indexing functions. Data can be accessed through the REST API. Elasticsearch support scalability to multiple nodes and replication. The solution can be self-hosted or external cloud service, e.g., Amazon Web Services can be used. [40, 41]

TimescaleDB

TimescaleDB is also an open source time series database. It is re-engineered from PostgreSQL and by that support full SQL. TimescaleDB still supports NoSQL-like vertical and horizontal scalability. The first release of TimescaleDB was on April 2017, and it is currently on version 0.9.2. The data model in TimescaleDB is wide-column based which is the traditional relational database type. Data model consists of hypertables which are sets of smaller tables called chunks. Data is accessed only through the SQL interface. Standard SQL analytical functions are supported but also time series specific first, last, and histogram functions are provided. [42, 43]

CrateDB

CrateDB is a distributed SQL database which is targeted for storing and analyzing large amounts of machine data. CrateDB employs NoSQL storage and indexing under its SQL database engine. CrateDB is written on Java and it is currently on version 3.1.1. CrateDB is owned by Crate.io and it has community and enterprise editions. CrateDB has document-oriented approach and dynamic data schemas. It has interfaces for example ANSI SQL, ODBC/JDBC, and HTTP. There are also custom plugins for, e.g., Grafana and Apache Kafka. For data analysis CrateDB offers standard aggregation functions, geospatial queries, data and location mapping, SQL joins, user-defined queries, and anomaly detection. Automatic data retention and size management functionality is not implemented on the 3.1.1 version. [44]

2.4. Benchmarking

In this sub-chapter, benchmarking, in general, is discussed in more detail. Characteristics of benchmarking metrics and quality measures for the used metrics are described. Also, current time series database benchmarks are provided.

As one of the objects of this thesis is to evaluate and compare the performance of time series databases, it is required to conduct some statistical measured comparison. This act of measuring is called benchmarking. Benchmark program is a specific program to measure performance metrics of some other program. [45 p.111] Benchmarking consist of an initial set of values and set of queries. Benchmarking should also happen in a closed controllable environment where test results can be replicated. This way result comparability is ensured.

As Lilja [45 p.9] states in his book, the core of benchmarking are the used performance metrics that are measured. When measuring the performance of a computer or a program usually is measured: a count of some events, duration of some specific time interval, or the size of a parameter. From measuring these characteristics, the actual describing so-called performance metric can be derived. As there can be variance and uncertainty during the benchmarking, some performance metrics can give different results when repeated. To evade this error, there are some characteristics of which determine a good performance metric. The used metric should be linear as if the performance of the system under test changes on some ratio the value of the metric should change on the same ratio. The metric should also be reliable as if the metric shows that A is better than B this relation should remain overall tests. As stated earlier, the benchmarking and therefore the metric should be repeatable. With the same experiment for the same system, the result should be every time the same.

As the benchmarking operation should be easy to perform the used metric similarly should be easy to measure. If the value is hard to measure it can indicate that it is not interesting and valuable for the users. The metric should also be consistent, so the produced results are comparable over different systems and tested programs. Alongside with consistency, the metric should be independent. No external influencer should be able to influence how the metric performs. [45 p.9-12]

2.4.1. Test bench design

In this thesis benchmarking is done as a simulation type where the testing environment is close to the real-world scenario. That includes all real-life scenario system configurations, e.g., cybersecurity measures are enabled during the testing. For this purpose, a test bench is needed which is used for performing the simulation. In figure 3 general database test bench design is presented [46].

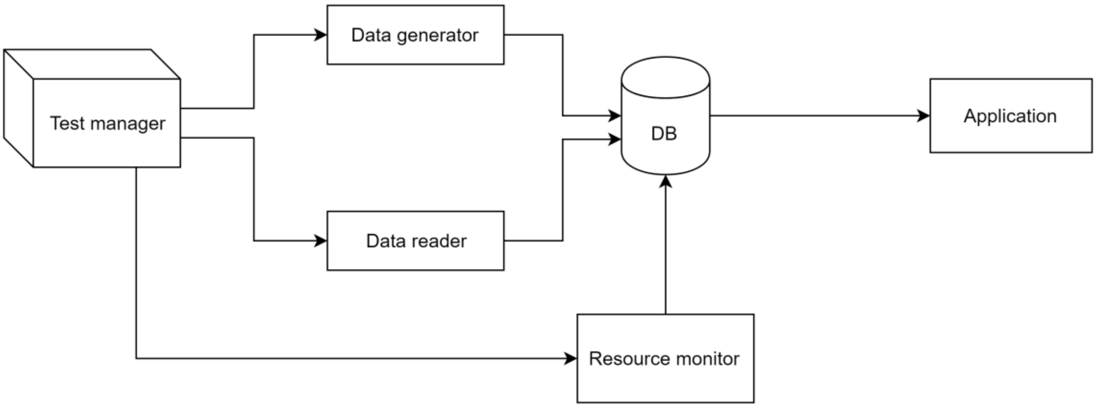


Figure 3. General database testing design.

The test bench should consist of data generator which creates and inserts the test data to the database, data reader who queries the inserted data from the database, and resource monitor which observes the resource usage of the database during the testing. Business application is simulating separate user or application that can be used to test user experience during the test run. [46]

For comparable results, the data write and data read operations must be able to use some fixed parameter that ensures that same amount of data is handled during each test with same paraments. For example, when testing data reading each time, the same amount of data must be returned from the database. Similarly, as the test bench should be a simulation type, all the test parameters should be configurable so that different simulation scenarios can be produced.

2.4.2. Current TSDB benchmarks

TPC consortium defined benchmarks are in practice the standard in evaluation and comparison of the performance of database systems. TPC has many benchmarks for databases including TPC-C, TPC-DI, TPC-E, TPC-H, and TPC-VMS. Benchmarks are not especially for time series databases but databases in general. The TPC-C benchmark is an On-line Transaction Processing (OLTP) Benchmark. Benchmark uses wholesale supplier specific data and is measured in transactions per minute. [47] The TPC-DI benchmark is for Data Integration and its used for a situation where data is combined from multiple sources to one database [47]. The TPC-E is also an On-line Transaction Processing (OLTP) Benchmark. It has more complex transaction types and execution structure. The benchmark uses stock brokerage data as an initial set of data. [47] The TPC-H is a decision support benchmark. The benchmark gives results as to how the database performs queries against its size or prize. [47] The TPC-VMS is a combination of TPC-C, TPC-E, TPC-H, and TPC-DS benchmarks and it is targeted for virtualized databases [47].

These benchmarks are good in the sense of extensive documentation and open-source implementation of most of them. As seen in chapters 2.1 and 2.3, in the industrial point of view scalability and acting in different kind of scenarios are the core of the characteristics. These benchmarks implement only a single scenario type of benchmarking and are therefore not good enough for this thesis.

Yahoo researchers have developed YCSB (Yahoo! Cloud Serving Benchmark) benchmarking framework which is targeted for NoSQL databases. The framework includes a set of workloads that have, e.g., a different mix of read and write operations, data sizes, and request distributions. The architecture of the framework consists of three components: a client, a data generator, and a core measurer. As the framework is open-source, it is highly extendable and modifiable. [48] As the framework is not targeted for time series data Bader extended the framework to YCSB-TS. As the YCSB uses delete and update queries and most of the time series databases do not support those, they were changed to SCAN, AVG, SUM, COUNT queries. Also, a timestamp column was added to the data schema. [49] As the YCSB was already seen fitting for time series database benchmarking and it was modified to fit especially to time series data YCSB-TS can be taken as design and benchmarking reference for this thesis.

There is STAC-M3 benchmark suite which is targeted for solutions with high-speed analytics in time series data. As the benchmark suite is closed source code and only members of the STACK Benchmark Council have access to it and its detailed test specification, it is not suitable for this thesis. [50]

3. RELATED WORK

In this chapter, relevant projects and papers for this topic are presented. The scope of the topic is limited to time series database and in general database comparison and benchmarking. This chapter is used to identify the scientific knowledge gap that this thesis tries to fulfill.

Schemakeit [51] has created a test bench for time series databases was developed in 2017. The test bench had a building energy management system's point of view on the topic. The test bench was developed on top of Docker, and it was a combination of benchmarking time series database and MQTT broker. Test bench had only writing speed, CPU time, memory, and pricing as a measurement metrics. Metrics were decided based on the building energy management system's viewpoint. In the thesis, the test bench was used to compare InfluxDB, KairosDB, and OpenTSDB. The results were that InfluxDB and KairosDB outperformed OpenTSDB. [51] As the thesis and the development test bench lacks the viewpoint and requirements of IIoT, it cannot be used for this thesis' purpose. In the implementation of the test bench the main lacking components are reading speed and the disk size of the database. It is also noticeable that all tested databases have released new versions of their product since the publication of the thesis.

Bader [49] developed another test bench, TSDBBench, in 2016 which was targeted for time series databases. Test bench uses two metrics in comparison: query latency and space consumption. The test bench is scenario-based which means that benchmarking is performed in two scenarios where each scenario has five different sub-scenarios differencing by replication factors and cluster size. Two main scenarios are differencing on query types. The first scenario was 1000 read queries and second scenario 250 scan, avg, sum, and count queries. The test bench was used against InfluxDB, OpenTSDB, Druid, Rhombus, MonetDB, Blueflood, and KairosDB. Results showed that tested databases performed very differently under different scenarios. When looking over all scenarios, Druid performed the best. [49] Developed test bench provides relevant metrics and scenario-based support for time series database benchmarking, but it lacks the IIoT point of view. The implementation and design are useful and can be used as a design reference in this thesis.

Only a feature-based comparison of time series databases was made by Wlodarczyk in 2012 [52]. Four database products: Chukwa, OpenTSDB, TempoDB, and Squwk were under comparison. The concerned features were: storage infrastructure, data acquisition, GUI, and support for advanced analysis. For advanced analysis purposes, OpenTSDB seemed to be the best solution. If full IT support is lacking and the solution is wanted to be self-hosted TempoDB seemed to be better for that situation. [52] These findings compensate Bader's findings that databases perform differently on different situations. As databases have different features, knowing what features there should be in IIoT is the core of this thesis.

Goldschmidt et al. [53] took the industrial point of view on their time series database benchmarking. Benchmarked databases were OpenTSDB, KairosDB, and Databus. Instead of just benchmarking multiple databases the target of the authors was to prove if these databases can linearly scale, handle the industrial workload, tolerate crashes, and have independent read and write performance. The benchmarking was done by having two different kinds of workloads and different cluster sizes. The results showed that KairosDB was the only one that fulfilled the hypotheses. KairosDB could handle a workload of 6 million smart meters with a cluster size of 24. [53] For this thesis research showed that when having the industrial point of view in benchmarking the time series databases, the benchmarking workload needs to mimic real-world environment as the performance depends on the test data.

InfluxData [54] has published four technical papers of their implemented database comparisons. They have compared their database against OpenTSDB, Elasticsearch, MongoDB, and Cassandra. The comparison has been made with write throughput, disk storage, and query performance metrics. In each comparison, InfluxDB is shown to be better. [54] For example, InfluxDB seemed to have five times faster write throughput, 16.5 times less disk usage, and 3.65 times faster query performance than OpenTSDB [38]. Results should be taken with caution as they are used for marketing and the benchmarking scenarios might have favored InfluxDB. Also, for the write speed test, they used bulk load write method which is not a practical way of writing in IIoT scenario. The used dataset had 10 000 unique values where each value had a new measurement in 10s intervals. This dataset does not represent the typical scenario for an industrial environment. It is still good to have some reference values that can be used to evaluate the correctness of the implemented test bench.

Mathe et al. [55] have made time series database comparison from the DIRAC system point of view. In their comparison, OpenTSDB and Elasticsearch databases were used. The benchmarking was done by having three different scenarios 10, 50, and 100 clients that performed database queries which had 1-day, 3-day, 7-day, or 30-day intervals. The performance was measured with the average response time and query throughput. Results showed that Elasticsearch performed better when the query interval increased. Similarly, when the number of clients increased in the test, the performance of the OpenTSDB decreased. [55] As mentioned in chapter 2.1 one essential characteristic of IIoT is scalability. The database should be able to scale without severe performance impact. Knowing that this might be an issue for some databases the test bench should include a test for that scenario.

Rudolf [56] evaluated different databases for Smart Spaces. The author argued three hard requirements: time series data support, available API, and ACID support. Also, five soft requirements were argued: versioning, replication and partitioning, access control, encryption, and a messaging system. The author also conducted performance benchmarking for the databases with insert, read, and delete queries. The author suggested PostgreSQL, Redis, and InfluxDB databases for Smart Spaces. [56] The research paper conducted good performance benchmarking and feature-based comparison. As the point of view is targeted for general Smart Spaces and not specifically for IIoT the paper does not provide an answer to this thesis' questions.

Lautenschlager et al. [57] conducted time series database comparison as a part of introducing their database implementation. The evaluation included the author's own Chronix database, Graphite, InfluxDB, and OpenTSDB. The comparison was made with write throughput, storage efficiency, and query performance as metrics. The results showed that Chronix had 4-33 times faster write throughput, 5-171 times more efficient disk usage, and 26-85% faster query times. The evaluation had three different scenarios in the sense of data writing and queries had four different query intervals. [57]

In summary, previous related work showed that there had not been implemented an extensive time series benchmarking test bench and feature-based comparison which would have the industrial point of view. Previous test bench implementations did not use all relative metrics in their performance evaluation or the used test cases does not reflect on IIoT use cases. Goldschmidt et al. benchmarked the time series databases on cloud with large node count. Their focus was also more on database scalability than overall performance. For example, their evaluation did not evaluate how the requested timespan affects on the database’s read performance. It hence leaves a knowledge gap on how databases perform as a single node solution and in more detail. Regardless previous implementations showed good design practices for the test bench and benchmarking. Designed testbench should have scenario-based testing and use at least query latency and space consumption metrics. Test cases should also include test case for scalability.

4. USE CASES OF A TSDB IN IIOT

In this chapter, different use cases for time series database in IIoT are presented and argued what is required from the time series database in those use cases. Presented use cases are: write heavy scenario solution, read heavy scenario solution, concurrent write and read operations scenario solution, and hierarchical solution. Use cases show how the critical factors for the database differ based on the business scenario.

In a single node solution, the system consists of only one database. The database can act as a data collector where the database is located on-premise near the data producing devices. The database is under heavy write operations. In this use case the write throughput, write scalability, database size management, and data aggregation are critical features. The motivation for this solution is just to gather the on-premise data without hectic data usage. Example use case of this kind of solution is on-premise IoT solution with IoT gateway which presented in figure 4. Data is collected from the devices to the database and only aggregated values are forwarded with the IoT gateway to, e.g., a cloud for more detailed and processing with a more extended history length.

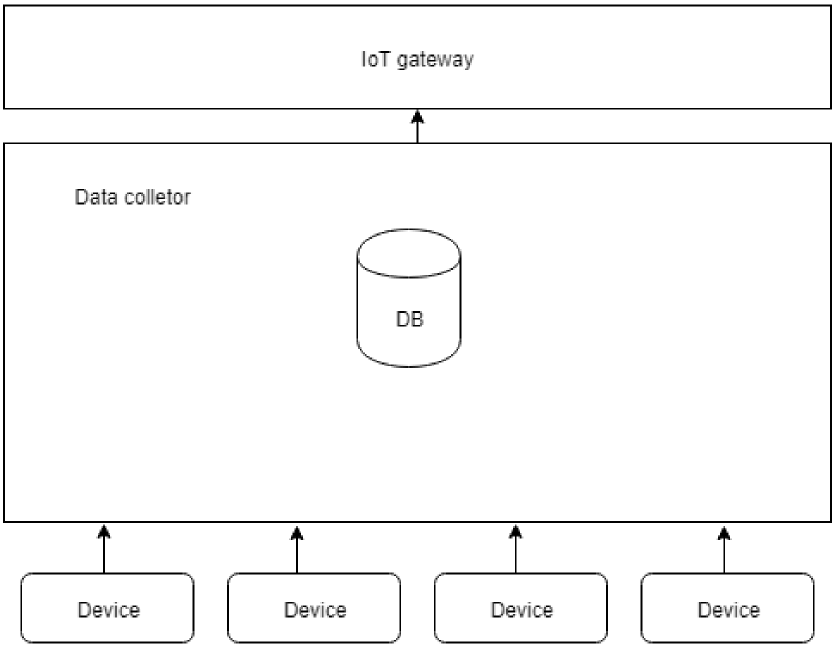


Figure 4. Write heavy data collector solution.

Another use case is where the database acts on more data analysis purposes, and that way is more under heavy read operations. In this scenario the database’s read throughput, analytical features, e.g., data processing and predictions, and data modeling features are critical. Example use case of this kind of solution is plant information management system (PIMS) or cloud-based database which is presented in figure 5. PIMS collects and integrates data from multiple sources and offers that data for different business levels. PIMS system is used for analyzing plant-wide data for business decisions. With IIoT PIMS type of systems can be deployed on the cloud. Data can have multiple different users, e.g., process managers, business controllers, buyers, and chief executives. As the collected data consists of a large area which has multiple sub-areas and devices, data modeling becomes critical for efficient usage. Custom and hierarchical data models are necessary for subclass type of data.

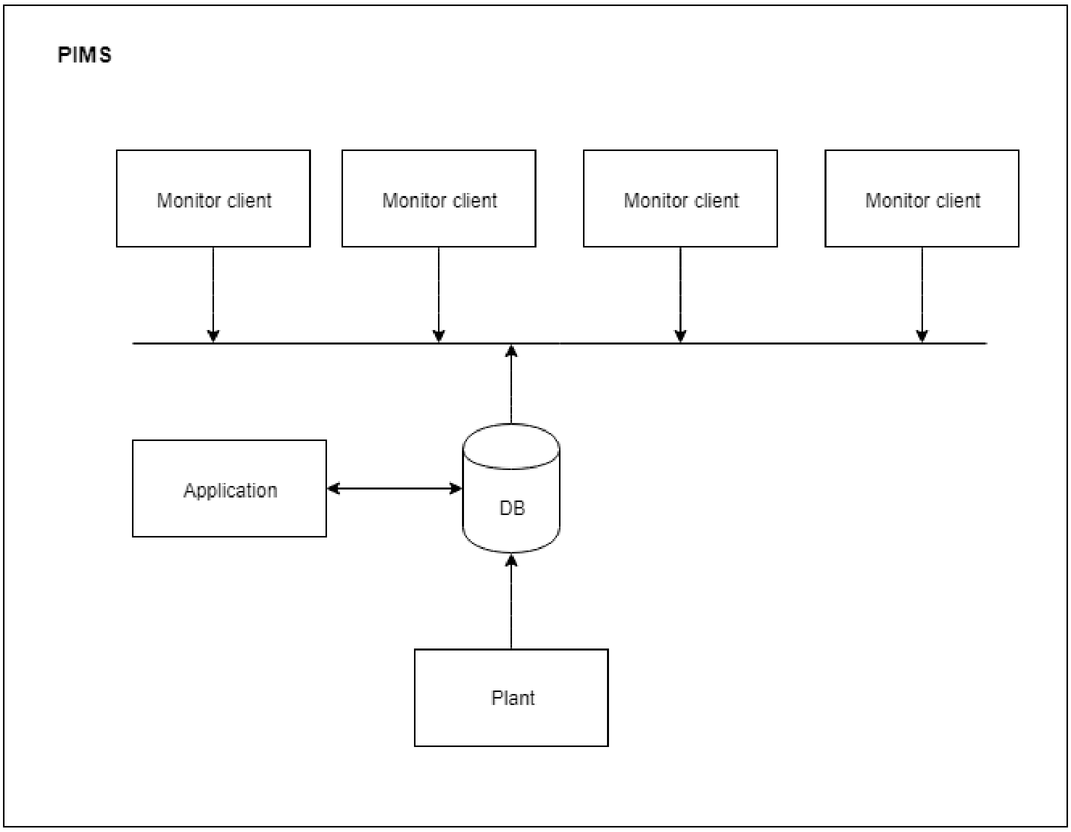


Figure 5. Read heavy PIMS solution.

A third use case can be supervisory control and data acquisition (SCADA) system where data is concurrently written and read for monitoring and control functions. SCADA is an old system concept, and it is not specific to IIoT. IIoT allows more control and monitoring type of functionality to be done and hence SCADA like control and monitoring system is used as an IIoT use case example. SCADA system consists of the underlying devices, central control server, database, and human-machine interface (HMI) which is used by the process operator to control and monitor the connected devices. The architecture of a SCADA system is presented in figure 6. In SCADA like system scenario the database is required to handle concurrent write and read operations without extensive impact to each other. For the control purposes ability to send commands back to the device based on the received measurement from the device is functionality that is expected from the database.

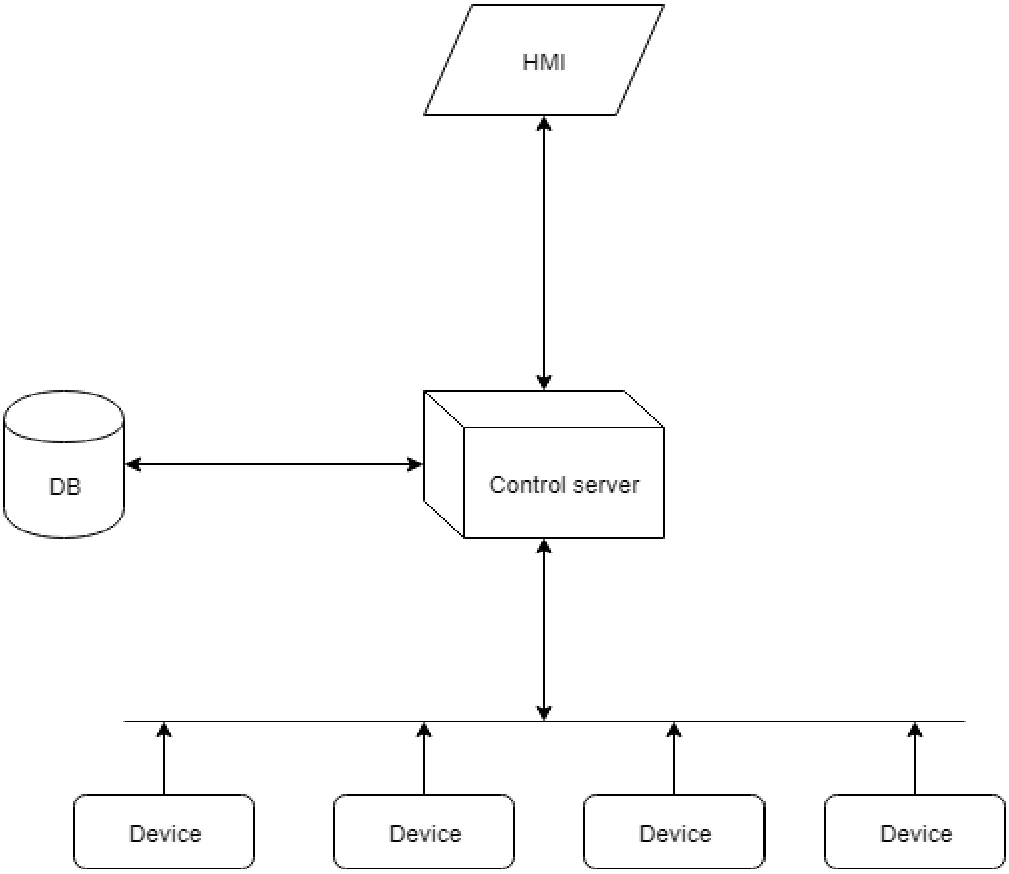


Figure 6. SCADA monitor solution.

A fourth use case can be a hierarchical solution where multiple databases are connected to each other from different levels and working as one single system as presented in figure 6. In a factory environment this scenario is typical where the production line consists of multiple parts which all can have their own database for, e.g., SCADA system and these databases forward their aggregated data to an upper-level database which holds the whole production line wide data for, e.g., PIMS systems. This kind of solution relays on the database's hierarchical configuration and functionality to co-operate with other databases.

As reliability is an important factor in industrial field databases usually are replicated to allows data availability all the time. In a replicated system data database consists of two database nodes which have the same data and this way allow the data to be accessible and writable even if one of the database nodes malfunctions. For SCADA like control systems, data availability is critical as the monitored device needs to be able to be controlled.

5. REQUIREMENTS FOR A TSDB IN IIOT

The core of this thesis is to identify the requirements that the time series database has in the IIoT environment. The identification is made by reviewing the IIoT literature, derived from earlier use case scenarios and IIoT characteristics, and interviewing employees of ABB who have the practical knowledge of manufactory customers' demand and behavior of their usage context. The interview was a group interview where five ABB employs who have been working with time series databases in industrial environment over 10 years and are part of time series database developing team. Interview was unstructured interview were identification of the requirements were done by going though different real client use cases and key characteristics in them. The literature review was targeted for IIoT solutions and databases in industrial usage. Used keywords included: *IIoT*, *IIoT on premise, IIoT challenges, time series database, IIoT forecasting, smart factory, control systems, SCADA,* and *PIMS*. Literature review included over 30 papers from journal databases and subject specific websites. Based on literature and interview findings the requirements are divided into functional and non-functional requirements. Functional requirements define what the database should do and have while non-functional requirements define how to database should perform.

5.1. Functional requirements

Based on the related literature and interviews the following functional requirements were identified:

- Deployment/platform support
- Time series support
- Datatype support
- Aggregation support
- API's
- Information models
- Hierarchical systems
- Write-to-device support.

Next, each of the requirements is described and reasoned in more detail.

Even though cloud computing is a core component of IIoT, Edge computing is as an essential component for interfacing the devices, especially in the case of an old installed base. Also many manufacturers want and need local data processing and management for privacy, cybersecurity, application functionality, and performance reasons. The platform may be selected in the Cloud according to the database requirements, but in on-premise Edge the installation platform is frequently defined by other requirements than the database, especially in the case when it is embedded in the device itself.

The most popular platforms for on-premise are Windows, Linux, or Docker. Especially lately Docker has gained popularity as a deployment platform [58]. Docker's easy way to produce multiple similar environments with one docker image is valuable as there can be multiple deployments of the database on the same smart factory.

As the focus is on time series databases the database needs to have support for time series data. The database should have a built-in data model which implements Equation (1) from chapter 2.2. The database should present the current value of the data point efficiently for supporting real-time data visualization and other monitoring applications. For fast data processing and visualization, the database should have specific storage for aggregated values which contains pre-aggregated values, e.g., average of the last hour.

As shown in chapter 2.2 the number of data points per day can grow in enormous sizes. When processing data over multiple days or weeks the processing can be extremely slow if it is done only with the raw values. The database should have pre-aggregation functionality so that it can calculate the pre-aggregated values for the incoming data. Pre-processing functions, e.g., alarm limit detection, data validation, and value unit converting are valuable for accurate and efficient data processing. Alarm limit detection functionality is necessary in process information management use cases. Data validation would validate that the received value is in a reasonable range and by that improving data reliability and facilitating application development. As e.g. the USA and Europe use different units for some metrics, unit converting is valuable so that the same database product and data can be used across the different locations. The database should also have specific storage for events and alarms as they are key concepts in an industrial system. In the industrial context, events could be, e.g., too low oil level of a motor or if the motor's RPM is abnormal.

As with any database, the broader the database's data type support is, the more useful it is in various scenarios. In an industrial environment, these data types could be different lengths of integers, enumerations, doubles, floats, strings, timestamps, UUIDs, arrays of previous, and BLOBs. Broad datatype support is part of efficient data processing. It should also be noted that some of the values can vary in length, and the database should be able to handle varying length data efficiently.

In industrial systems knowing how the system performs under critical scenarios, for example under maximum load, is crucial. A database that is integrated into performance critical system should provide support for simulations where these scenarios can be tested without modifying the actual data. Simulation enables the investigation of phenomena when actual observed data is not available. Fields, where this simulation functionality is used, are especially oil, gas, and energy management. [59] From the analytical side, the database should provide some support for prediction and forecasting functionality. This functionality is required e.g. in preventive maintenance use case. The analytical side of the database is also valuable for the user to achieve more detailed data insights.

As seen in chapter 2.1, in IIoT there can be multiple different protocols and technologies in use. The database should provide the necessary API's for data injection, data acquisition, and querying. Especially in an industrial environment, OPC UA is a standard communication protocol for data acquisition as well as retrieving to applications. Having the possibility to use OPC UA protocol from the device to inject data into the database or the database to subscribe the data from the device would enable various use cases. [60] Other relevant data injection API's could be ODBC/JDBC, HTTP, or WebSocket. In the industrial environment, SCADA systems are common data acquisition systems and the database should support communication with them [61 p.9]. With the installed base it is essential that the database provides protocol master services (active client that subscribes the data from devices) and supports older generation protocols such as classic OPC and Modbus. The database should also have a subscription type messaging system between different nodes or the database and a client. Subscription functionality would be used for example when the monitoring system wants to subscribe just to critical failure events. Publish-subscribe pattern reduces database and network load as the client does not need to make read queries at constant intervals.

In IIoT, the data comes from some device and produced values are mapped to some specific device or equipment ID in the database. For better usability of the data in various kinds of applications, the database should support information models or so-called digital twins. Information models help to make the raw measurements more understandable for the end users and enable model-based applications and data visualization. [62, 63] The data should be able to access through a model which combines all the relevant data. A database should also support external applications and protocols which use an information model, e.g., OPC UA. Database's data acquisition services should be configurable with the created information models keeping the data modeling uniform.

In chapter 2.1 scalability, failure-tolerance, and hierarchy were listed as critical characteristics for IIoT. By having multiple nodes, measured data is not lost in case of database failure. In large plants and factories distributed hierarchical control systems are standard. This base system hierarchy also creates a requirement for the database to support hierarchical system configurations. An example hierarchical solution with four database nodes is presented in figure 7. Especially in control systems, the database with its analytics features should be able to write the desired setpoints back to the control system. The database therefore should have write-to-device support which allows secure bidirectional communication between the database and the connected device. Plant level applications are typically run against the upper level database and the write to device shall be supported over the hierarchical configuration.

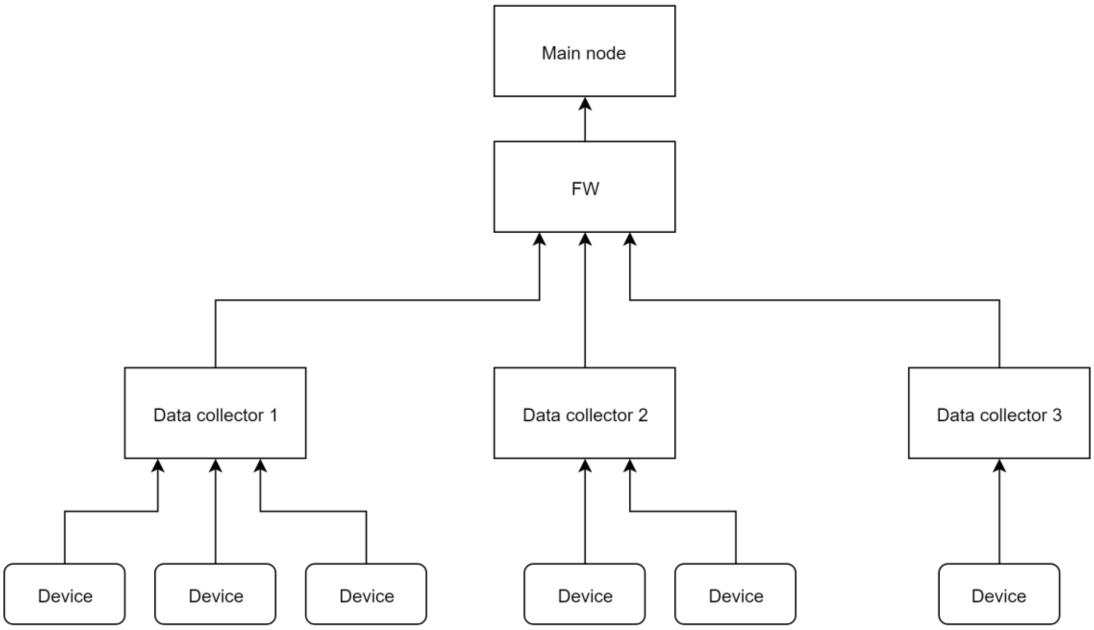


Figure 7. Hierarchical system.

5.2. Non-functional requirements

Based on the related literature and interviews the following non-functional requirements were identified to relate to:

- Scalability
- Security
- Availability
- Performance
- Resource usage
- Costs
- Engineering experience
- Lifecycle.

Next, each of the non-functional requirements are described and reasoned in more detail.

In chapter 2.1 scalability is identified as one key characteristic of the IIoT. The database should be scalable from small few signals data logger to plant wide or Cloud based central system. In this context, scalability contains many viewpoints. As the idea of the IIoT is to connect multiple (thousands of) devices which each may produce up to hundreds of frequently changing time series signals, the database should be scalable to handle this vast amount of time series data and clients. As the amount of data grows in the database as the time passes, the database should also scale by managing the size of itself ensuring the database does not grow over the disk size or some other predefined limit.

If the system is configured for high availability, there should be a load balancing functionality which allows multiple databases to serve the same data efficiently. Database nodes may also support federations that makes them seem as one database and provide horizontal scalability. Scalability can also be addressed with hierarchical systems where the detailed raw data is collected in lower level nodes and aggregated values transferred to central higher-level system.

In industrial context, the data may include sensitive business information which must not leak outside the operating context. The database should implement basic AAA-principles and the fundamental information security CIA-model [64, 65]. Data in transit must be secured in device to database, database to database, and database to clients communications. Users and clients should be authenticated, and data access authorized. There are multiple ways to implement the authentication, e.g., certificate-based or supported by ID management system. Authorization should provide role-based access control which enforces the object of keeping the data available only for the authorized operating context in corporate wide deployments. In a situation of multi-tenancy, the database should provide a secure and private environment for each tenant. The database should also keep audit log for security and fault-tolerance reasons. Changes should be trackable to a specific user, and in case of failure, the database should be able to recover without data loss.

One characteristic of IIoT systems is demand for high availability to enable continuous data collection and service for the clients. This may be achieved in different ways, but typically it is implemented with multinode architecture where data is replicated to multiple nodes as presented in figure 8. Data replication allows data to be accessible even if one of the nodes suffers from a failure. The database should also have a backfill support which allows filling missing values if there are, e.g., network failure in the system. If the database fails, there should be a backup and restore functionality allowing database restoring. Possible data loss can mean significant financial loss for the company and the risk should be minimized with described methods.

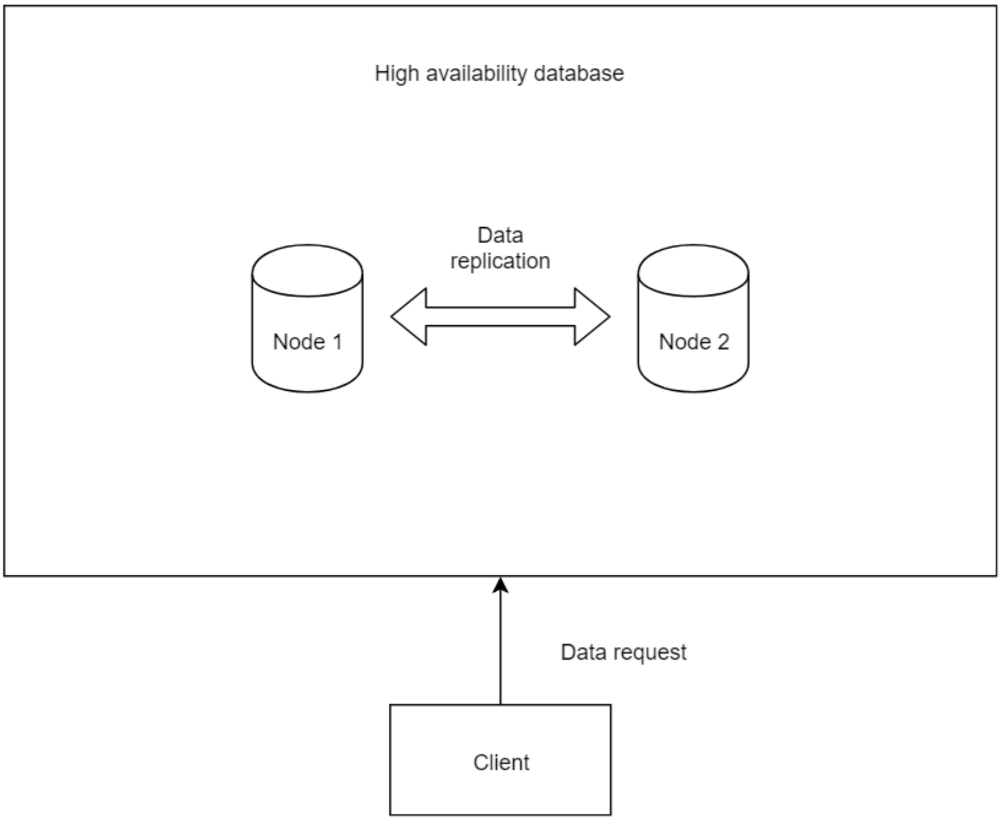


Figure 8. Architecture for high availability system.

As many of the tasks in IIoT are time-critical the performance of the database is critical for successful implementation. In this context, the performance consists of data injections speed, data retrieval speed, and latencies between the client and the database. As the amount of data may be enormous in IIoT, the database should handle even tens of thousands of values per second as continuous load. That kind of performance requirement, especially as continuous load, is not typically met by the regular databases.

Resource consumption is related to performance as it shows the cost of the gained performance. Resource management consists of CPU, RAM, networking and disk space usage. As the time series databases typically compress the data when its saved on the disk, the smaller the disk space usage is, the better compress performance the database has, but there must not be any significant loss in the data accuracy – this means that reducing the amount of data samples is not the preferred way for compressing data.

From the financial point of view, the cost of the database is also significant. Primarily if the database is deployed on the cloud service, there is pay-as-you-go type pricing where the price is bound with the level of usage. If the solution is deployed on-premise, there can also be costs on software licenses and the hardware.

The database which is chosen should give the optimal price-performance ratio. In multimode solution the overall price of the system can grow fast which gives more value for the price requirement. Price should also be examined in other features that it holds, e.g., dedicated support and related own development costs.

From the user's point of view, the usability of the database is critical. The usability can be measured in the level of engineering experience and work required to install, configure, and maintain the database. The configuration consists of configuring info models, instances, signals, data acquisition, and role-based access control. The amount of documentation and its quality also affect to the usability of the database.

Availability of usage support, and its quality in the sense of response time and location, for the database affects on the usability of the product. Nowadays online support service is relatively standard, and it eases the support availability. Even an on-premise support person can create significant value for the manufactory customer.

In addition, the database's lifecycle and product support availability are essential in the industrial use. For example, for many industrial devices the lifetime and support for maintenance can be over 15 years [66]. The services based on the database should have as long life-cycle expectation and support available for updates and upgrades for the installed solutions.

6. TEST BENCH DESIGN AND IMPLEMENTATION

One research objective of this thesis was to design and implement performance test bench for time series databases. In this chapter, the developed test bench' design and implementation are presented. First, the overall architecture of the system is examined. Then the main components of the system: data production, data consumption, and telemetry recording are presented in more detail.

6.1. System architectural overview

The test bench is written on C#, and it has a configuration file that is used to control the test parameters. A tested database is deployed on a virtual machine as a single node solution. The test bench does not automate virtual machine deployment or database configuration that must be done before the test bench is run. As it was seen in the previous chapters and related work, the test bench should support different scenarios. The test bench has three different test cases which try to simulate three common IIoT use cases which are:

- On-premise IoT, heavy data insertion with random reads
- PIMS/cloud-based, random inserts with heavy reading
- Monitoring, concurrent data insertion, and reading

With these use cases, the test bench tries to take IIoT point-of-view to the performance measuring. On-premise IoT scenario is a smart factory type scenario where the raw data is sent to a local database, and from there more aggregated values are sent in batches to another database. In this scenario database load is more data write-oriented and the write performance of the database is more critical. PIMS/cloud-based scenario would be another database where the aggregated values are sent from the premise. The database is under heavy read-type load as the data is used for more analytical tasks and the read performance of the database is more critical in this scenario. The third scenario simulates on-premise monitoring scenario where the raw data is under real-time monitoring and the combined performance of read and write operations of the database is crucial. Figure 9 presents the overall architecture of the system. The architecture inherits the normal database testing system architecture from chapter 2.4.1.

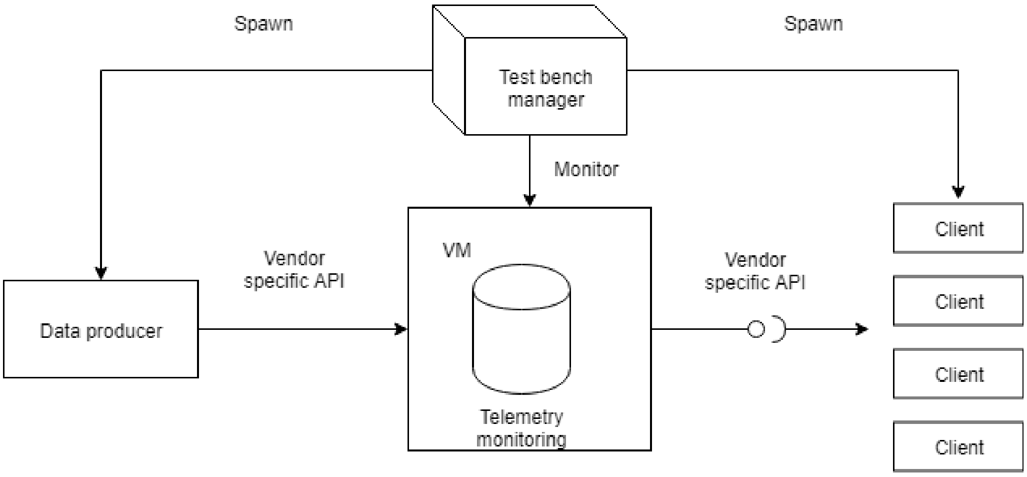


Figure 9. System architecture.

Test bench manager maintains the test runs and spawns the necessary data producer and clients who read the data. Test bench manager also spawns the telemetry monitoring to the virtual machine that holds the database under the test.

The first test case tests the write performance of the database with measurements as the number of data points to write per second and the delay of when the written data is available for queries. The second test case tests the read performance of the database with measuring the time it takes to complete the query. Testcase has three different subcases where timespan, the number of variables, and the number of connected clients is increased. The third case tests the effect of reading the concurrently written data and how it affects on write and read performance of the database. All test cases can be run separately or in sequence. From the earlier identified requirements, the test bench is targeted for scalability, performance, resource consumption, and lifecycle. Other requirements (e.g., redundancy, API support, cost) are evaluated based on documentation as they are hard or impossible to measure in the test bench.

As the database's performance might differ over time because of disk saturation, the test bench is designed to support long test cases. In these long test runs, test bench writes data to the database and starts to measure the database's performance after a given time, e.g., one week. This way the real performance of the database can be measured. Also, the lifecycle management that was one requirement for the database can be tested with longer test runs.

The test bench is modular enabling testing of a new database by only implementing the read and write functions that are typically vendor specific. Also, the virtual machine that has the database should be created.

In chapter 2.4.1 was stated that for simulation purposes the test bench should be configurable for different simulation types. Hence as the number of variables to write and write frequency, number of variables and timespan to read can all be modified for different workload scenarios, the test bench allows the user to run the test bench for user's specific scenario. This design differs from earlier implementations which use some specific dataset, and it could not be modified.

Test bench saves all collected measurements on own text files which use tab separation format. Each test, subtest, and telemetry collection measurements have their own recording file. Test bench doesn't implement result processing, but it is left for dedicated programs, e.g., Microsoft Excel.

6.2. Data production

Data production is done by using vendor-specific API and especially the API that is recommended for best performance. Produced data is time series type of data which was presented in chapter 2.3 where the datapoint has time, value, and status. Produced data also has a variable name which represents a measurement, e.g., the rotation speed of a motor. Hence the produced datapoint has four columns as

$$X = (timestamp,\ variable\_name,\ status,\ value), \tag{2}$$

where datatypes are timestamp, text, text, and integer. Data is written into a single table or another similar type of data structure to keep the databases comparable as their data structure can vary. In the write performance test case the number of variables is increased after configurated iterations. Iterations define how many measurements of write performance is taken with one variable count. During each iteration, a new random value is inserted for variables. The frequency of write operations can be configurated for different types of write-loads. Write performance is measured as an average time it took to write the specified number of data points. Depending on the write operator's behavior the execution time is either the write operation(s) duration or the read-back duration with a threshold value. As the default write frequency is one per second, the number of written variables is equal to the number of written datapoints. The most recent written value in each write iteration is read back to verify a successful write operation. Value is also read back to measure the delay that the data has until it is available for reading once it has been written. Each iteration duration is fixed for one second. If the write frequency is more than once per second, only the last write operation of that iteration is verified. Hence write is verified only once per second. Data production can be configurated with initial iterations where data is written to the database without performance measuring to fill the database. By filling the database with initial data, the average on-production performance can be measured.

Measured metrics, write operation duration and duration to read the value back, are standard and hence valid benchmarking metrics. Metrics are dependable only on the number of written values and used communication protocol. As some databases only offer one interface to it the used communication protocol and part of databases overall performance. Duration to write the same number of values should also be repeatable. If the values differ then it shows that the database's performance is not static. Duration should also be linear when the number of written values is changed linearly.

6.3. Data consumption

Data read test case has three different subcases where timespan of one variable, number of variables, or number of connected clients are varied. The timespan variations measure the read performance of the database over time field, and it measures how many data points can be read from the database per second. Testcase measures how time increase in timespan and hence the number of queried datapoints affects on query completion time. As the key query in time series databases is against time field of the datapoint, this performance is key read performance type of the database. The second subcase measures how the database performs when multiple variables are read from the database. As in the IIoT environment, one device can send multiple measurements which correspond to multiple variables, read performance over multiple variables can be important for cases that kind of devices are monitored. In this test case, the timespan of each variable is static and can be configurated. The third subcase measures how the number of reading clients affect on query competition time of one client. In this test case, other clients query a configurable number of random variables on given timespan. As seen in the requirements the database should be scalable as there can be multiple devices trying to write and read to the database. Hence the performance of multiple connections should be measured.

Read query tries to read all datapoint fields and the search query is targeted with variable name and time fields. Figure 10 presents an example search query of multiple variables, but the query can differ because of database own query languages. The result of read queries is also verified that all queried data was received, and the correct amount of data points was retrieved. To be sure of the amount of data that the database has before read tests, data is written to the database with fixed timestamps and data is queried against those timestamps.

*Select * from data where variable_name in ('testinstance_1','testinstance_2') and time > now() – interval '5 min';*

Figure 10. Example search query.

As measured metrics are time-based, they are also valid performance metrics. Read operation duration can differ based on how the used drive is fragmented. Fragmentation can be compensated with long test runs. Also, the more static the duration is, the better performance the database has. Hence the linearity of the metric indicates worse performance for the database. When comparing different databases, results should be comparable then same hardware and number of values are used.

6.4. Telemetry recording on resource consumption

Telemetry recording measures the resource consumption of the database. Measured metrics are CPU usage, RAM usage, Disk usage, Disk space, and Network usage. As the virtual machines are Microsoft Hyper-V virtual machines, the telemetry recording is done via Powershell. Powershell gives access to Hyper-V statistics which record the resource usage of each virtual machine. In the write test, resource consumption statistics are saved before each variable increment so the effect of write load increment to the resource consumption can be examined.

Similarly in the write test, before each increment in timespan, variable count or client count the resource consumption statistics are saved. As the disk space of the virtual machine is measured before and after the test case, the compression of the database can be evaluated with the difference of those measurements. Network usage can also be crucial in some situations and measuring that gives insights how network depended the database is.

7. EVALUATION

In this chapter, the second research objective of this thesis is conducted. The test bench is applied to three time series databases: cpmPlus History, InfluxDB, and TimescaleDB. Databases are first evaluated with the identified requirement from chapter 4. As the test bench evaluates only the performance aspect of the database, the evaluation for most requirements is done via the database's documentation. Lastly, the test bench is applied to these three databases, and the results of the tests are presented. The analysis of the evaluation is done in chapter 8.

7.1. Selected TSDBs for evaluation

For practical evaluation and demonstration of the test bench, three time series databases are chosen. ABB's time series database cpmPlus History is chosen as it represents business product which is dedicated for industrial field within ABB. InfluxDB is chosen as it has gained much popularity during the past few years and having reputation to provide good performance. InfluxDB is also a mixture of opensource and commercial closed-source product as the main core of the database is free. TimescaleDB is chosen as it is fully open-source and built on top of PostgreSQL. TimescaleDB is also popular within time series databases, and there was a practical case where a company used InfluxDB first but switched to TimescaleDB for better performance in their use case [67].

cpmPlus History

cpmPlus History is ABB's own time series database that is provided as part of ABB Ability^{TM} platform common components. cpmPlus History is developed and maintained by Digital ABB and free to be used and sold by ABB business units. Outside ABB only paid edition is available. Windows is currently the major deployment platform, though there is Linux version available, and Docker version coming soon. cpmPlus History supports high availability with online replication between two full functional nodes. Data is replicated in real-time between both nodes and they are also capable to provide full functional services the clients and data acquisition concurrently and designed to be always available if one of the nodes goes down. cpmPlus History has built-in data retention component that keeps the database in its configured size by deleting the old data when it is older than the defined life time.

For security requirement, cpmPlus History has user authentication and authorization, and secure communication protocols. cpmPlus History can use Windows domain or workgroup, Linux PAM, LDAP supported ID management, and Azure federated ID management for user authentication. User permissions on data can be set on three levels: classes (like a table), properties (like a column), and instances (like a row). For each of these levels user permission can be defined for five actions: read, write, execute, create, and delete. Data access permissions can also be set to some specific time range. For improving the engineering experience access control definitions support inheritance in information model structures that reduces the number of required definitions significantly.

Based on setting up the database for the performance testing and integrating it into the test bench cpmPlus History doesn’t require extensive engineering work. Installation of the database was the most straightforward of the three databases as all the tables were automatically created. The only configuration before actual usage was to set the hardware bonded configurations, e.g., maximum database size and cache usage. Database installation comes with a tool that can be used to set those configurations. cpmPlus History as a database has relational data model, but through the public APIs it exposes higher-level object-oriented information models (class, property, instance) and especially for time series management so called equipment model. Understanding these concepts and how to use them on the database can create some extra work. However, as the database is targeted for industrial field actual data is more natural to model on these concepts than on relational data model with tables, columns, and rows.

cpmPlus History is currently on version 5.1, and its first release was in 1999. Products long lifetime and dedicated development team indicate long lifecycle and support for the product. cpmPlus History’s version 4.5 which is five years old is still under maintenance and bug fixing support.

cpmPlus History support timestamps on 100 ns resolution. cpmPlus History implements by default the time series data point structure which was presented in chapter 2.2. Supported data types for the value field are Boolean, enumerated binaries, integers, floats, text, arrays, and blobs. There is also support for events and alarms. For data aggregation and processing, cpmPlus History provides preprocessing, alarm/event detection, validity and substitute handling, online aggregation, and data recording for high-frequency signals. Aggregation functions contain, e.g., time average, sum, std, min/max, time integral, and operating time. There is also support for simulations and what-if situation processing.

As public API’s, cpmPlus History supports OPC DA/HDA, OPC UA, .NET, ODBC, C++, MQ, REST, WebSocket, and for data acquisition, e.g., Modbus interface. cpmPlus History implements a publish-subscribe pattern allowing, e.g., clients to subscribe to some specific variable. cpmPlus History also allows users own data models as it has data abstraction layer on top of the database.

cpmPlus History support hierarchical system configurations where there are lower level data collector nodes which can be connected to the higher master node. cpmPlus History also support bi-directional interfaces for SCADA and DCS implementing write-to-device support.

InfluxDB

The second evaluated database is InfluxDB. As InfluxDB was briefly introduced in chapter 2.3, it is a relatively new database. For cost requirement, InfluxDB can be free as it has TICK-stack product collection which is open-source under MIT license and one product under AGPL license. TICK-stack is a single-node solution, and it has all the main components. InfluxDB has an enterprise edition which gives clustering functionality which gives high availability and scalability, more tailored security, advanced backup and restore functionality, and complete support. From the industrial point of view clustering and security functionality can be mandatory. Pricing for an on-premise solution is done based on nodes.

For deployment and platform support requirement InfluxDB supports Linux, OS X, and Docker. InfluxDB also has binaries for Windows, but it is currently on experimental phase. InfluxDB can also be deployed on AWS. For hardware, InfluxDB has one soft limitation as it is designed for SSDs. The database can be used with normal hard drives also, but it is not recommended for production purposes as the database does not perform as well on it.

As stated in pricing InfluxDB supports scalability and high availability only in the enterprise edition. High availability is achieved with data replication. Clustering is done with meta and data nodes where meta nodes have the cluster management and data location information, and data nodes have the actual data. Data is distributed and replicated by shard files which are time divide set of the actual data. Scalability, when data grows, is handled with retention policies concept which allows data deletion after a certain period of time. There can be multiple retention policies on the database allowing a different length of histories for the data. For the security requirement free version of InfluxDB has only authentication and authorization, and database level read and write permissions. Enterprise edition adds fine-grained authorization which allows measurement, series and tag level permissions where measurement equals table, series equals variable, and tag equals column. For cluster management, there are also 16 different permissions. Permissions include managing nodes, managing databases, managing users, and monitoring the cluster.

Based on setting up the database for the performance testing and integrating it into the test bench InfluxDB doesn't require extensive engineering experience. Installing the database is exceptionally straightforward, and documentation for setting up the database is clear. The configuration of the database is done via one configuration file. Because of InfluxDB has built own data schema, efficient data management might require an extensive understanding of the use case's data layout from the user.

As InfluxDB is relatively new, its lifecycle and future support should be secured. The database has had one big architectural change during its lifetime that divides the product support in half. The database is currently on version 1.6.1 and during last one-year timespan is has had 13 version updates. As the main core database and its other add-ins (Telegraf, Chronograf, and Kapacitor) are open-source, support and bug fixing is not bonded to a limited amount of developers. The more the customers and users take part in the product's development the more support they get. With enterprise edition there comes also dedicated support team from InfluxData.

InfluxDB has broad support for time series data. If no timestamp is given when writing the data, InfluxDB uses server's locale nanosecond UTC timestamp for the written data point. The database engine uses the timestamp for managing and locating the data. InfluxDB does not have specific concepts or storage for current value or events and alarms. For datatype support, InfluxDB supports strings, floats, integers, and Booleans. InfluxDB has built-in functions for basic data aggregation (sum, mean, median, stdev, and others), querying data with sampling and filling, and data prediction with the Holt-Winters method. More complex functions are left for the user to build from those simple functions. InfluxDB has continues query concept which allows, e.g., automatic aggregation calculations, value limit detection, and data transfers for different measurement tables. For events, alarms and data processing there is another TICK-stack component, Kapacitor. With Kapacitor user can create alerts, events, and custom data pre- and post-processing functions. InfluxDB does not have support for simulations.

For API's InfluxDB supports HTTP as the main write and read interface. There is also support for UDP, but as it is connectionless, it is not meant for data writing. InfluxDB has also support for CollectD, Graphite, OpenTSDB, and Prometheus.

There is a subscription type of messaging support, but it is used only with Kapacitor. Kapacitor is implemented with publish-subscribe pattern and expands supported API's as it has an interface for example for MQTT. As the InfluxDB's data schema was briefly introduced in chapter 2.3, the data schema is flexible in some constraints. There is no support for information models or user's data models, but data must be modeled in terms of InfluxDB's concepts: measurements, tags, and fields. Hierarchical data models are not supported.

There is no support for hierarchical systems in InfluxDB. In clustered solution all databases are equal. InfluxDB itself does not have write-to-device functionality, but it can be achieved with Kapacitor. The remaining limitation is that the device and Kapacitor should be able to communicate with each other.

TimescaleDB

The third evaluated database is TimescaleDB which is fully open-source database and hence has no costs. It is built on top of PostgreSQL which is also open-source database. TimescaleDB is under Apache 2.0 license and PostgreSQL under its own PostgreSQL license. For deployment and platform support requirement TimescaleDB supports Linux, OS X, Windows, and Docker. TimescaleDB is also able to be deployed on AWS and Azure. There is also TimescaleDB Enterprise edition which is paid edition. Enterprise edition adds deployment support, proprietary features and functionality, and SLA level support. More detailed description of proprietary features and functionality is not publicly told.

TimescaleDB has replication functionality which allows high availability. Replication is done by mirroring the primary database's data to another node. Replication is done by streaming the WAL file to standby nodes. The database can also scale horizontally by making the replica nodes read-only nodes, allowing them to be used for read queries. TimescaleDB does not have built-in automatic size management or old data deletion. Functionality can be created, but it will run as a scheduled job by the OS. Data retention policies can be created for each hypertable. By PostgreSQL, TimescaleDB also supports backup and restore functionality. For TimescaleDB security functionality support comes from PostgreSQL. PostgreSQL allows user authentication, access control which includes a database, table, column, row level of permissions. Different roles and user groups can also be created.

Based on setting up the database for the performance testing and integrating it into the test bench TimescaleDB itself does not require extensive engineering experience but it is highly dependable from PostgreSQL. Working effectively with TimescaleDB knowledge of PostgreSQL is required as most of the functionality and syntax is supported by that. Installation and documentation for TimescaleDB are straightforward and clear but compact.

As TimescaleDB and PostgreSQL are both fully open-source lifecycle and support are dependable of the activity of the product's community and users. The database has Timescale company managing the product which gives some static support and a static element for the product. TimescaleDB was released in 2017, and hence there are not much data of lifecycle support. The product is currently on version 0.11, and it has some public deployments with other companies. The backbone of the database, PostgreSQL, is nearly 20 years old and currently on version 10.5 which indicates long lifetime support for the TimescaleDB.

TimescaleDB creates the time series data support for PostgreSQL database. TimescaleDB implements wide-column data model allowing multiple values associated with one timestamp instead of creating a unique time series data point for each value which has the same timestamp. Data consists of chunks which are time partitioned individual tables. Chunks are integrated into one big table, hypertable. Smallest supported time unit is a microsecond. As the hypertable consists of time partitioned chucks, the table is indexed by time on default. Supported data types are charm char, varchar, text, integers, floats, Boolean, UUID, arrays, JSON, and JSONB. From PostgreSQL TimescaleDB supports full SQL functions. TimescaleDB adds time series specific functions like time bucketing, first and last values, histogram, and data filling. TimescaleDB also supports data backfilling. Predictions can be made by combining time series specific functions and PostgreSQL functions. Currently, TimescaleDB does not support alerts or events. There is support for triggers which can be used for data validation. Specific built-in processing or aggregating functions are not implemented, but full SQL support offers basic arithmetic aggregation functions. Automated queries or jobs, e.g., automated data aggregation is not currently supported.

As Timescale is built on top of PostgreSQL and supports SQL, TimescaleDB supports the same APIs as PostgreSQL. These are, e.g., OBDC, JDBC, ADO.NET, and REST API. There is also a specific adapter for Prometheus which is metric collecting tool. Because of open-source background, there are multiple different plugins created for PostgreSQL including MQTT adapter. Correctness and stable function of these plugins can of course vary. PostgreSQL allows custom data types, composite types, and views which can be used for user's information modeling.

There is BDR plugin for PostgreSQL which allows multi-master replication which would allow hierarchical system configurations with selective replication. Making BDR compatible with TimescaleDB hypertable format requires still some work. A built-in solution for hierarchical systems and write-to-device is not implemented on TimescaleDB or PostgreSQL.

Summary of how each database fulfilled the functional and non-functional requirements is presented on table 1. Each database is graded with scale of zero to two how extensively it supports the requirement where 0 present no support, 1 some support, and 2 extensive support. For the costs requirement the scale is product with no costs is graded with 2 and product with mandatory costs is graded with 0. Lifecycle grade is affected by the history of product support.

As cpmPlus History and InfluxDB does not have stable release for Linux or Windows they have 1 for deployment/platform support. TimescaleDB does not have built-in automatic data retention policies as others have and hence it has 1 for time series support. InfluxDB is lacking large datatype objects while others have support for then and hence it has 1 for datatype support. TimescaleDB is lacking some more advanced aggregation functions compared to the other two. InfluxDB has only HTTP interface itself and hence lacking in API's. InfluxDB does not support own information models and TimescaleDB supported them on some level. Both InfluxDB and TimescaleDB does not support hierarchical systems. TimescaleDB did not support writing back to the device while with InfluxDB it could be done with Kapacitor component and MQ protocol. TimescaleDB was scalable on only for reading and hence it has 1 for scalability. Security was on the same level on all the databases. All the databases supported availability with hot standby. cpmPlus History and InfluxDB have free versions with limitations (only inside ABB or with limited functions). Usage of TimescaleDB requires good knowledge of PostgreSQL and hence has 1 for engineering experience. InfluxDB and TimescaleDB are relatively new and the long-time support for the product cannot be evaluated at this point and hence they have 1 for lifecycle.

Table 1. Summary of requirement fulfillment

	cpmPlus History	InfluxDB	TimescaleDB
Deployment/platform support	1	1	2
Time series support	2	2	1
Datatype support	2	1	2
Aggregation support	2	2	1
API’s	2	1	2
Information models	2	0	1
Hierarchical systems	2	0	0
Write-to-device support	2	1	0
Scalability	2	2	1
Security	2	2	2
Availability	2	2	2
Costs	1	1	2
Engineering experience	2	2	1
Lifecycle	2	1	1

7.2. Evaluation and test results

In this subchapter the test results are presented from the test bench which targets for measuring the performance and resource consumption requirements for the selected databases. Tested database version and used configurations are presented in table 2. Virtual machines that run the databases had 4 core 4,2 GHz CPUs and 12 GB RAM. For InfluxDB only one measurement was used, and each variable has two tags and one field value. For TimescaleDB PGTune tool was used for configuration settings. One hypertable was created, and an additional index was created for the name column.

Table 2. Database versions and configurations

	cpmPlus History	InfluxDB	TimescaleDB
Version	5.1	1.6.0	0.11 and 10.5
OS	Window 10	Ubuntu 16	Windows 10
Interface	Websocket	HTTP	C# PostgreSQL driver

As described in chapter 5 test bench consists of three different test cases. The test bench was run with the configuration shown in Tables 3, 5, and 6. For write test database was first filled with initial data that corresponds to 100 GB. This way can be verified that the database cannot be fully kept in-memory.

Table 3. Write-test configurations

Write frequency (updates/s)	10
Database size before test	100 GB
Initial variable #	15 000
Measurement iterations	1800
Variable increment	1 000

Figure 11 and 12 present results of write-test. Figure 11 presents the write operation as the number of updated variables is increased. Figure 12 presents the delay to read back the recently written value. Delay present the database's data pipeline's speed.

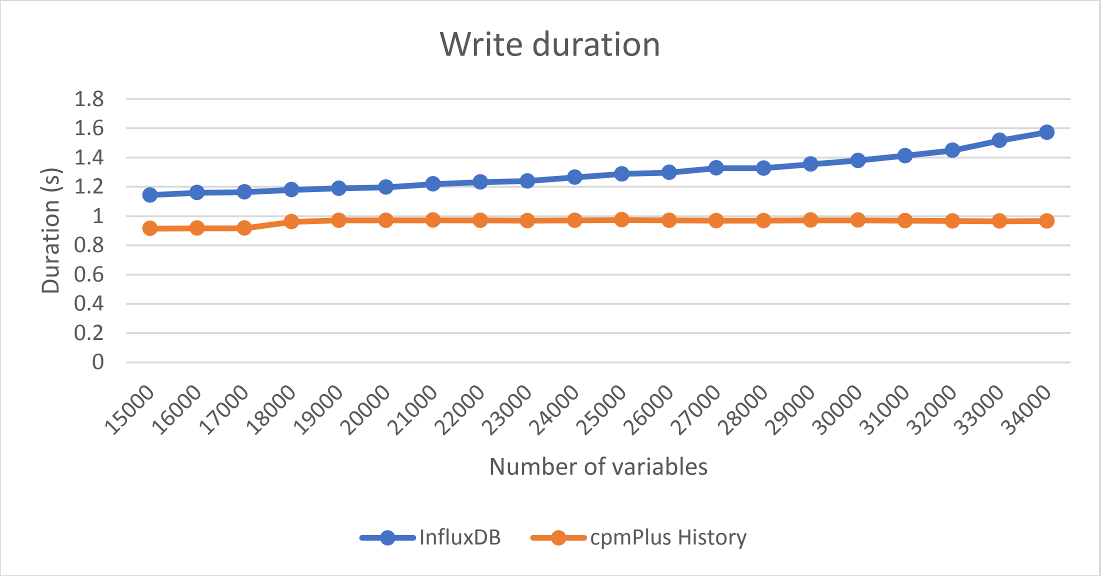


Figure 11. Write duration.

As the write duration is fixed to one second, write duration should be around 1 second. When the variable count is 21 000, the write duration is 1,2 s and could be seen as a limit for per second speed. InfluxDB's client's write operation waits until the write operation on the database is completed successfully hence the databases write speed can be calculated from the write operation's duration. As the write frequency was 10, the actual written datapoints per second are 210 000. This value is in line with InfluxDB's hardware recommendations and its upper write limit for hardware size that the used virtual machine had. TimescaleDB does not appear on the figures as the write duration for 15 000 variables was 1,4 s giving write speed around 110 000 data points per seconds. Read-back duration with 15 000 variables was 0,1 s. From the write duration can be seen that the database is not able to write 15 000 variables within one second and hence the test was not necessary to run for larger number of variables.

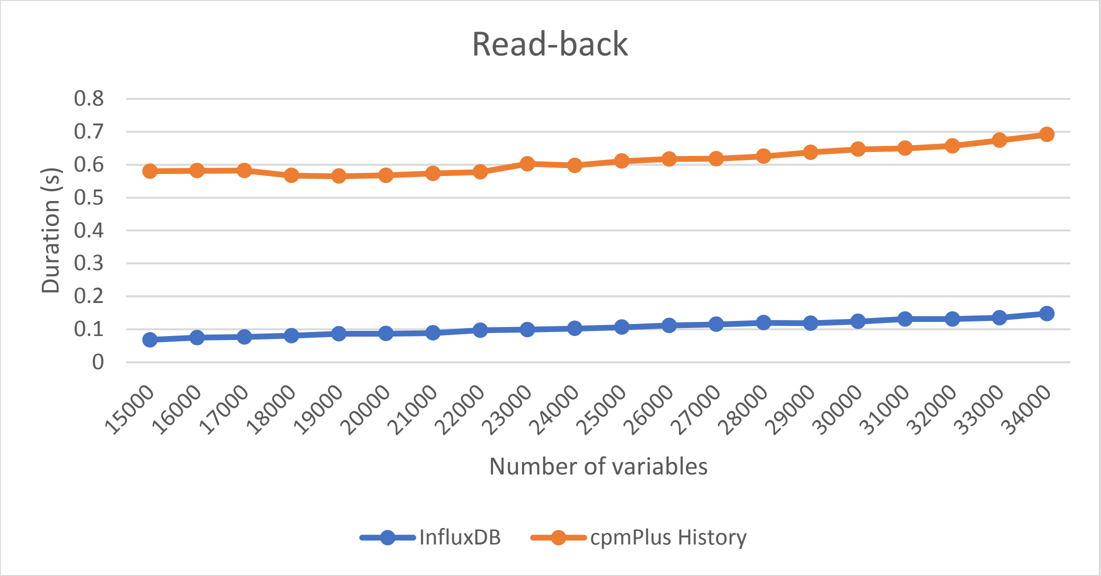


Figure 12. Read back duration.

cpmPlus History specific statistics with increased variable count are presented in figure 13. cpmPlus History client's write operation does not wait until the write is completed on the database hence the write operation's duration cannot be used to determine the write speed. The number of variables does not affect on write operation duration, but the database's performance can be seen starting to weaken from the duration to read the recently written value back. If the threshold for the duration to read the value back is 2,5 s as it can be identified as near real-time response time, then cpmPlus History write speed can be seen as 1 000 000 data points per second as the write frequency was 10 times per second for 100 000 variables.

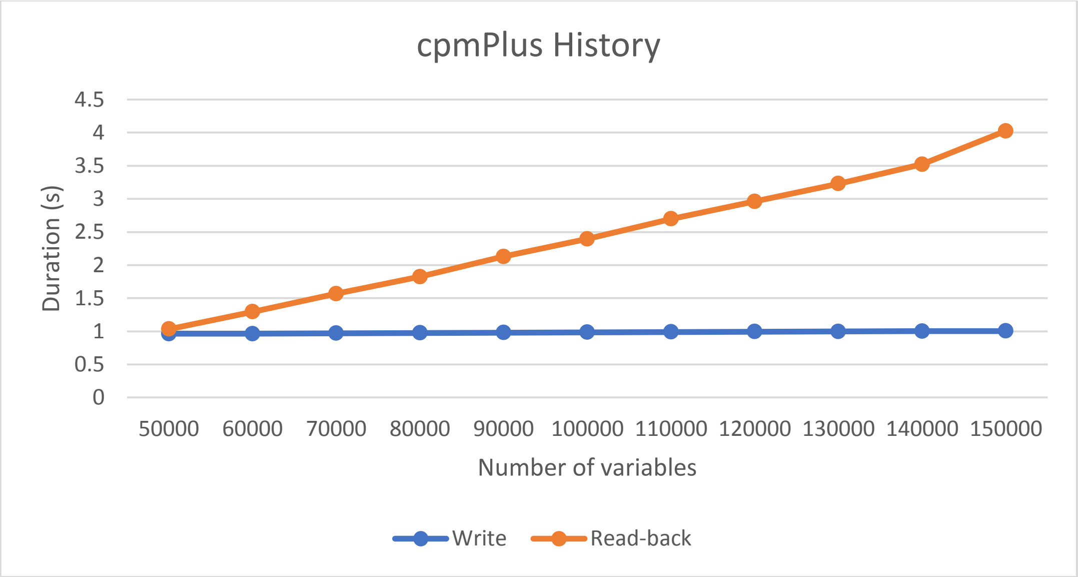


Figure 13. cpmPlus History specific statics.

Telemetry statistics during write-test with 15 000 are presented in table 4. For RAM usage it must be taken into account that database's RAM usage is linked with the amount the unique variable currently is written and held on the database. As the results are shown for 15 000 variables which are a relatively small number of unique variables concurrently updated, the RAM usage is also low. If the variable count were for example 150 000, then databases would try to use all available memory in the system resulting in around 8 GB of RAM usage.

In addition, data compression on disk was tested with writing 15 000 variables with the frequency of ten times per second over one hour. As one data point size is between 26 and 30 bytes, so the total written data is 15,8 GB. InfluxDB and cpmPlus History compressed it to 2,5 GB and Timescale to over 50 GB. Timescale uses more than the initial data as it has an additional index.

Table 4. Write-test telemetry statistics for 15 000 variables

	InfluxDB	cpmPlus History	TimescaleDB
CPU avg	26%	26 %	22%
RAM usage	6,3 GB	1 GB	40 MB
Disk usage(norm.)	654 IOPS	633 IOPS	5778 IOPS
Network traffic	10,5 Mbps	2,2 Mbps	30 Mbps

For read-test data was first written with fixed timestamps to ensure consistency between the databases. After that more initial data was written to the database to ensure that the data that is read on the first time, comes from the disk rather than from the cache. The test was run with configurations presented in Table 4. Read-test consist of three subcases: timespan increment, number of variables increment, and the number of connected clients increment.

Table 5. Read-test configurations

Write frequency	500
Fill time	1 hour
Max timespan	34 min
Max variables	1010
Variables timespan	1 min
Max clients	115
Client variables	5
Client variable timespan	5 min

Read-tests were run twice to measure how the database performs if the data is read from the disk or from the cache if exact same query result is kept on the cache. The results for timespan increment for one variable with the cold read, read from the disk, are presented in figure 14 and the results for executing the same query second time are presented in figure 15.

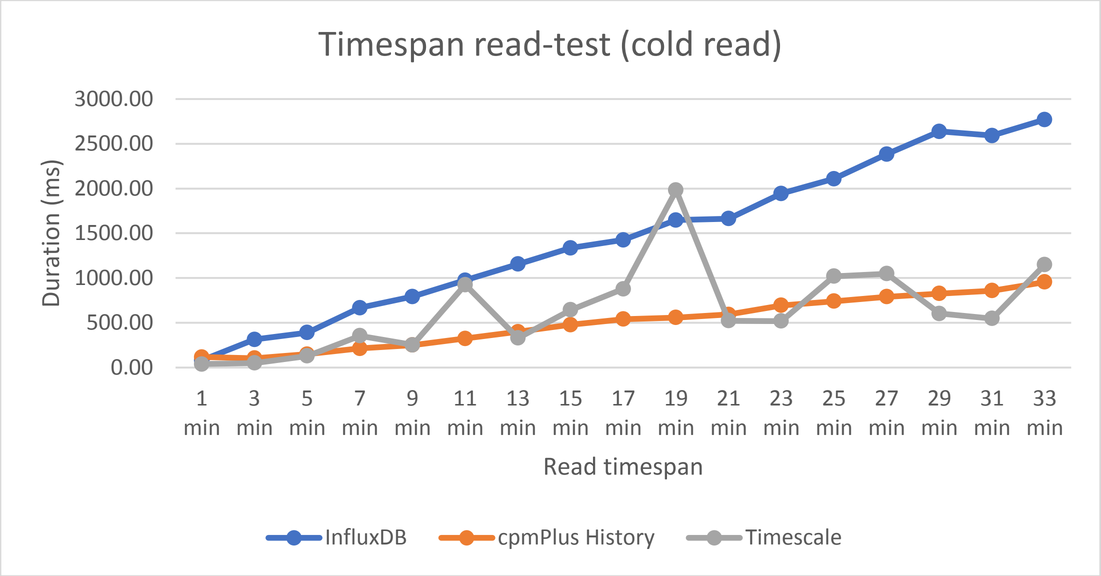


Figure 14. Timespan read-test results (cold).

InfluxDB and cpmPlus History give linear performance against linear increment in timespan being read. TimescaleDB gives a more divergent performance and read query duration can vary significantly. In one-second InfluxDB gives 11 min timespan which results to 330 000 data points. For cpmPlus History the one second limit comes in 33 min timespan which results in 990 000 data points. For TimescaleDB the actual limit value is not explicit, but the trend can be seen to pass one-second limit also at 33 min time span.

When rereading the same data, for InfluxDB there does not seem to be much impact. The one second limit for InfluxDB is passed at 13 min timespan which results in 390 000 data points. For cpmPlus History the impact for reading from the cache is more efficient as the one-second limit is not passed within the tested timespans. Linear predictive calculation results that the limit would be passed at 1 100 000 data points. For TimescaleDB the performance adapts same the performance as from the disk with smaller performance decrement spikes.

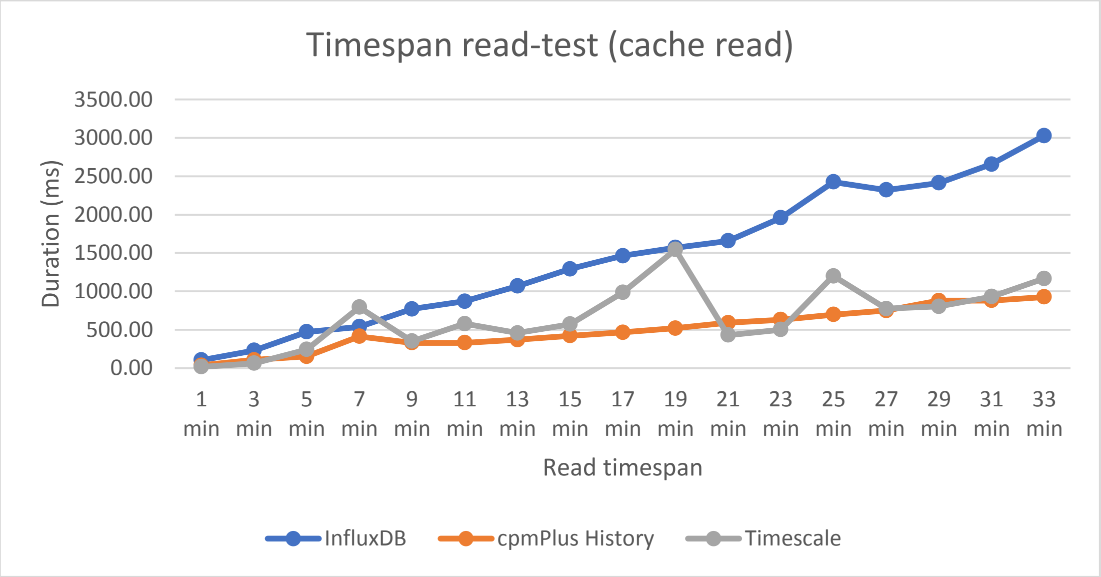


Figure 15. Timespan read-test results (cache).

The second read-test increased the number of requested variables. Data for second and third subcase uses data which is written with the frequency of 100. The results for the test are presented in figures 16 and 17. Figure 16 presents the results for a cold read and figure 17 for second iterations which should use cache for the requests. Again, cpmPlus History and InfluxDB have a linear performance against the number of requested variables while TimescaleDB gives a more divergent performance. cpmPlus History performs best as it returns 1-minute timespan for 1005 variables in 6,3 seconds while for InfluxDB it takes around 22 seconds and 16 seconds for TimescaleDB.

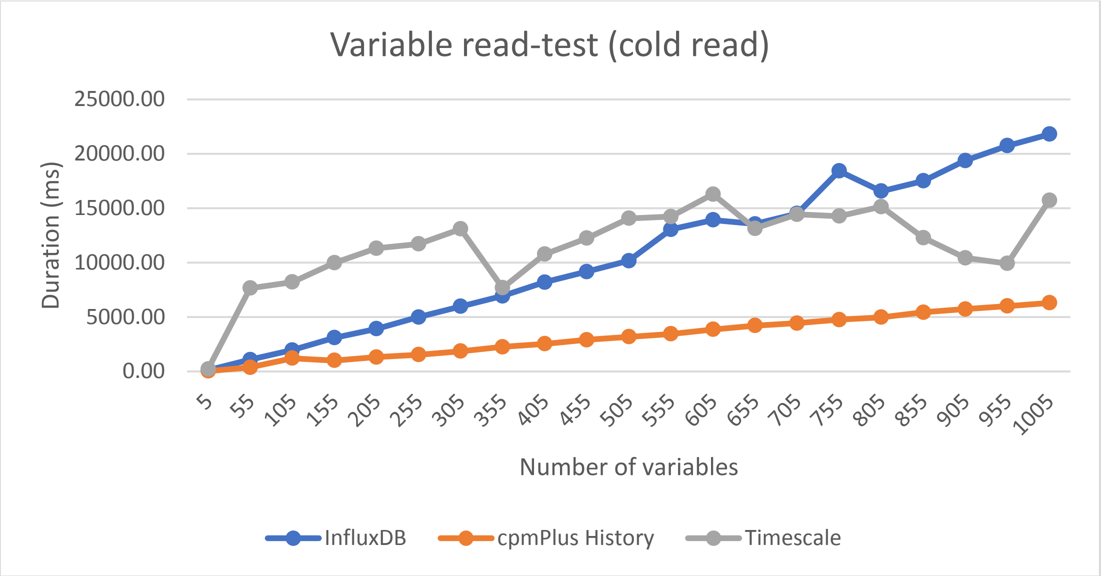


Figure 16. Variable read-test results (cold).

From figure 17 can be seen that when requesting the same data again cpmPlus History and especially TimescaleDB can use cache efficiently and their performance increases. Timescale performs similarly as cpmPlus History both returning one-minute time span for 1005 variables in 5,8 seconds while for InfluxDB it still takes over 20 seconds.

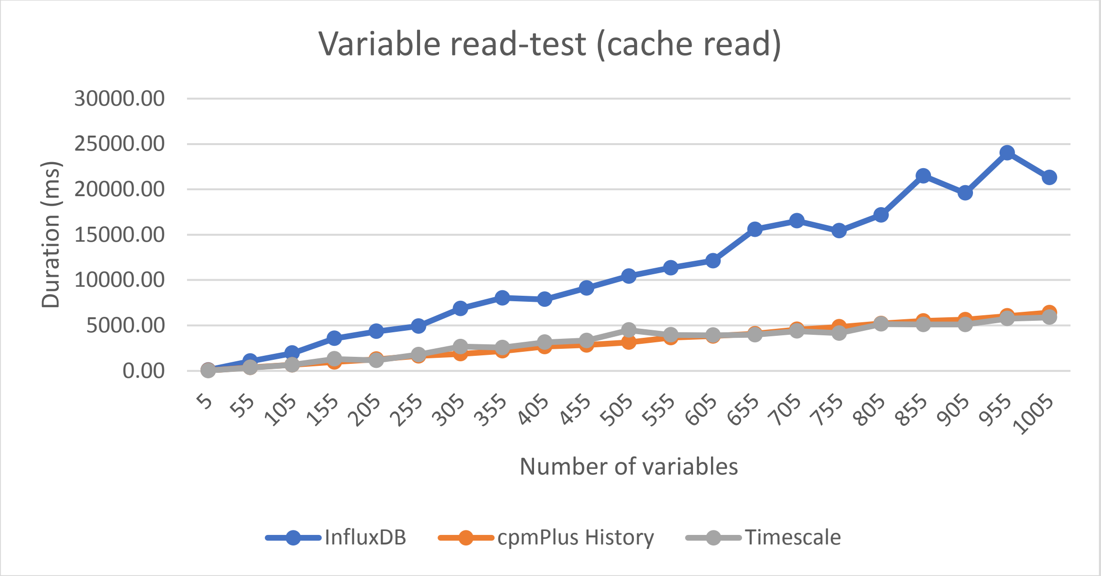


Figure 17. Variable read-test results (cache).

The third subcase was the increment of the number of connected clients. Testcase should identify if the database is scalable for a large number of concurrent clients and that request competition time for one client is not affected by the load. The figure presents the results of the test case. On the y-axis is the duration of the same request for one client which requests 5-minute timespan for 5 variables. Other connected clients use the same request but for random variables. X-axis presents the number of connected clients. InfluxDB shows high scalability as there is no increment in the duration time while the number of clients increases. cpmPlus History shows similar performance. TimescaleDB starts to have performance impact around 80 connected clients and especially around 100 clients. TimescaleDB was configurated for 110 clients.

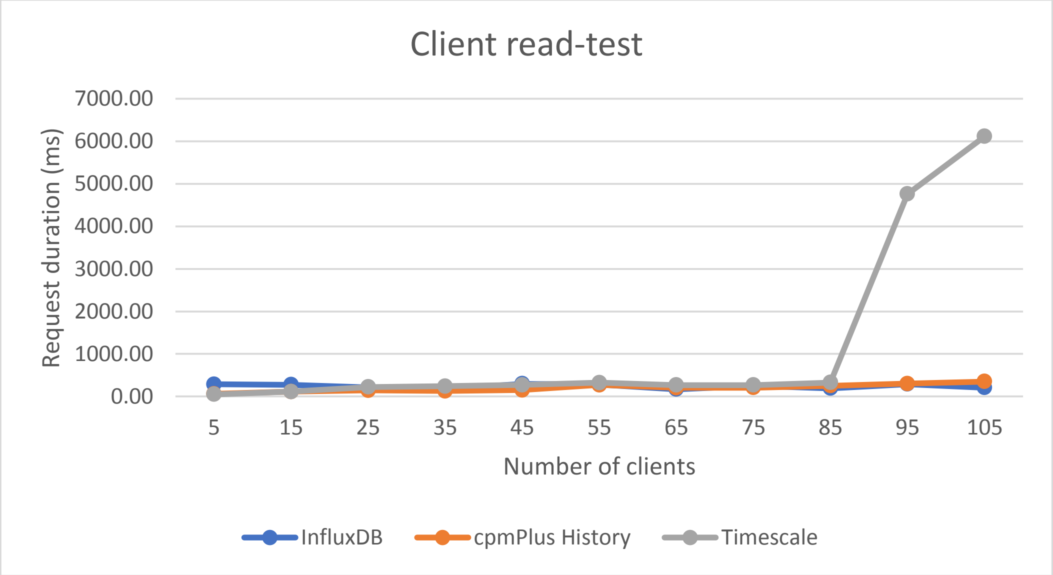


Figure 18. Client read-test results.

The third test case measures how the database performs under concurrent write and read operations. The used configurations are presented in table 6. There is one data producer who writes 15 000 variables ten times per second and one data consumer who reads 10 variables of those and trying to read the latest 15-minute timespan of those variables.

Table 6. Mix-test configurations

Write variables	15 000
Write frequency	10
Read variables	10
Read timespan	15 min

Critical factors in this test are if the database can keep the write performance during concurrent data reading and if the data consumer can read all the requested data. For cpmPlus History no performance decrement was identified, and all the requested data was successfully retrieved from the database. Write performance was not affected for InfluxDB, but it did not manage to return all the requested data to the data consumer. Data consumer did not get the latest 5 minutes. Similarly, TimescaleDB was not able to return all the requested data, and the client got only data for the five-minute timespan. Also, the data producing showed to take a performance hit as the time to produce 15 000 variables with a write frequency of 10 was 2,2 seconds while in the write-test it was 1,4 seconds.

In table 7 continues telemetry statistics while the test was running are presented. Again, it must be notified that small number of variables affect on low RAM usage. For disk usage is used disk active time which presents the utilization of disk's maximum speed. Hence the higher the active time percentage is the more bottleneck effect disk has for the database performance. cpmPlus History uses CPU, RAM, and network the least showing that it can handle the load successfully. InfluxDB uses the most CPU and RAM making it resource heavy database. TimescaleDB does not use RAM efficiently, and it uses disk heavily. TimescaleDB also has the most network traffic while concurrent data writing and reading. cpmPlus History had 1% disk usage but it only meant that cpmPlus History did not had to use the disk actively, cpmPlus History still used the disk to save the data.

Table 7. Mix-test telemetry statistics for 15 000 variables

	InfluxDB	cpmPlus History	TimescaleDB
CPU avg	50%	27%	45%
RAM usage	7,3 GB	1,2 GB	100 MB
Disk active time (utilization of max speed)	10%	1%	30%
Network traffic	13 Mbps	8 Mbps	44 Mbps

8. DISCUSSION

In this chapter, research questions and objectives are gone through and analyzed whether they are answered and reached. The implemented test bench is also evaluated against earlier implementation presented in chapter three. Performance measurement results from chapter 6.2 are analyzed, and an overall inspection of the thesis' success and significance is presented. The chapter is divided into two subchapters: conclusion of the test results and evaluation of the test bench and research objectives.

8.1. Conclusion of the test results

Identified requirements included multiple different point-of-views, and the weight of each of them can differ among business scenarios. For example, a hierarchical system configuration might not have high requirement weight if the developed system is to be implemented as a single level solution. When the three time series databases were evaluated with the identified requirements, differences in how they fulfilled these requirements were able to be seen. Hence the second research question was to analyze how current time series databases perform against these requirements.

There were both paid, and free databases evaluated giving the cost-functionality aspect. All three databases supported time series data and aggregation functionality broadly. TimescaleDB lacked the event and alarm functionality which is crucial for industrial field. There were also some differences on the time precision support as InfluxDB supported nanoseconds, cpmPlus History 100 nanoseconds, and Timescale only microseconds. Aggregation and processing function support was more advanced on cpmPlus History as for InfluxDB, and TimescaleDB more complex and advanced functions were left for the user to build from the simpler functions. cpmPlus History and InfluxDB both supported automated and online aggregations. Scenario support was only found on cpmPlus History, but all databases supported future forecasting in some sense. For datatypes, cpmPlus History and TimescaleDB implemented extensive support while for InfluxDB it was more limited. InfluxDB lacks support for UUIDs, arrays and large objects, e.g., BLOBs or JSON. cpmPlus History offers a readier solution as it has specific storages for current values, history values, alarms, and events. For InfluxDB and TimescaleDB this kind of separation needs to be done manually. From this core functionality and industrial point of view, TimescaleDB suffers still from its newness and is not suitable for IIoT usage yet.

For platform and deployment support both InfluxDB and TimescaleDB showed extensive support while cpmPlus History had more limited support than them. InfluxDB was limited with deployment size and price while TimescaleDB was cost-free regardless of the deployment size. As the clustering functionality for InfluxDB was only achievable for the paid edition, it forces the system developers to use the paid version because as seen in IIoT characteristics, the solutions are generally multinode solutions. That adds more weight to the cost requirement. All the databases can be deployed on premise as a self-hosted solution which was requested by many manufacturers. InfluxDB is designed for SSDs which can limit the deployment areas as many current industrial time series database deployments are done for regular HDDs. Also, as one of the characteristics for IoT is an extensive amount of data, the prices for large SSDs are still rather high which can lead system developers to choose regular HDDs for their solutions.

Scalability and high availability were requirements that the evaluated databases differentiated from each other on the implementation of these requirements. InfluxDB's clustering functionality allows scalability for both write and read operations as well as high availability with distributed data replication. TimescaleDB implements scalability only for data reading, but high availability is supported with complete data replication. cpmPlus History supports more scalable architecture with data and master nodes allowing scalable write and read operations and high availability with complete data replication. Differences in scalability and data replication can make one database more suitable for some specific business scenario. TimescaleDB's lack of scalability for write operations is critical as the one key characteristics for IIoT is a massive amount of data to be written.

For API support databases mostly supported the basic interfaces. InfluxDB and TimescaleDB both benefit from their open-source background allowing them to have extensive support for different protocols through external plugins. From the industrial point of view, OPC DA/HDA and OPC UA protocols are commonly used protocols, and only cpmPlus History supported them. This can be a critical decision maker for some business scenarios where other protocols cannot be used. Also, the need for publish-subscribe pattern support was identified in the requirements, and only cpmPlus History supports it natively. For InfluxDB and TimescaleDB it can be achieved through external plugins. For monitor applications, having the ability to subscribe to some values is a necessity.

For user's custom information models, there was support in cpmPlus History and TimescaleDB. In the requirements, the need for the user's own information models came from, e.g., usage of OPC protocols. As there was support for OPC protocols only in cpmPlus History, the lack of support or limited support for own information models was expected. The lack of support for hierarchical data models in InfluxDB can create extensive work in data modeling for more complex information models in the industrial field, e.g., plants or production lines. cpmPlus History and TimescaleDB both have composition functionality which can handle that kind of multi-system information models.

The main lacking requirement for the evaluated databases was support for hierarchical system configurations. Only cpmPlus History supported hierarchical systems with master and data collector nodes. InfluxDB and TimescaleDB allowed only equal level system configurations. If the business scenario requires hierarchical system, for example, a production line wide system with multiple databases, InfluxDB or TimescaleDB might not be ideal as they would require extended work to combine and use them in a hierarchical setting. Similarly, write-to-device support was missing from InfluxDB and TimescaleDB. The feature is specific for industrial use cases and hence can explain why it is not implemented on for more general use case databases.

For security requirement databases performed equally. InfluxDB offers full security control only for enterprise edition which also forces to use the paid edition for industrial use cases. All the databases supported even column and row level permission and supported secure communication protocols. From the security point of view, evaluated databases are acceptable for IIoT usage.

From the required engineering experience point of view evaluated databases performance equally. The installation for cpmPlus History and InfluxDB was extraordinarily straightforward, and other required configurations before usage were minimal. TimescaleDB required some configuration work before it could be used. cpmPlus History was more completed product than InfluxDB or TimescaleDB as it created different storages automatically for, e.g., current values, aggregations, and alarms.

Time series databases have gained popularity during the past few years. This can be seen from the young age of InfluxDB and TimescaleDB. Because of their young age, lifetime support and suitability for long lifetime demanded industrial environment cannot be appropriately evaluated. Both databases have open-source support which gives efficient bug fixing support and development for new features. cpmPlus History has long lifetime support and shows suitability for long support for older installations. As a closed-source product bug fixing and development for new features might not be as efficient because it is bonded to a fixed size development team.

Overall to answer the second research question as seen from the summary table 1, cpmPlus History performed best against the identified requirements as it is targeted for industrial use cases. InfluxDB performed second best as its most critical lack was the support for hierarchical systems, information models, and support for OPC protocols. TimescaleDB lacked critically in core functionality as it had limited support for data aggregation, size management, and no support for alarms and events.

The developed test bench was applied to three time series databases. Test results showed how databases differ in performance and how the database performs better for a certain type of usage. cpmPlus History showed best write performance giving around million data points per second write throughput. InfluxDB got only 210 000 data points per second, and it especially higher write frequency created performance cost to it as it uses HTTP protocol for writing. HTTP usage might not be the best protocol to use for high-frequency data, and as InfluxDB requires all column values to be given it the write message, the HTTP message size grows inefficiently. TimescaleDB was showed slowest write throughput with 110 000 values per second, and from telemetry statistics can be seen that TimescaleDB used most IO operations for writing the data making it slow. TimescaleDB’s high disk usage also showed that the database size was ten times as big as for the other two with the same data written to them.

Measured write speeds show similar results and are on the same limits as others have measured. InfluxDB’s write speed is the upper limit of write speed for the same type of hardware that was used in the test [68]. TimescaleDB was similarly measured to have write speed around 110 000 data points per second by TimescaleDB itself [69]. Similar test results show that the database is correctly implemented, and measured values are valid by being in the same spectrum with previous measurements.

The read-test showed that cpmPlus History and InfluxDB act mostly linearly to timespan and variable count increment. TimescaleDB performance was more distributed and gave random performance bottleneck spikes. Overall cpmPlus History gave best read performance giving 1 million data points per second read speed in timespan read test and high scalability with returning 800 variables with two-minute timespan in 5 seconds. Even though InfluxDB has relatively high write speed, the read speed was worst from the three databases. Similarly, TimescaleDB had worst write speed but performed almost as good as cpmPlus History in the read-test. This shows how databases can be designed for a certain type of usage. In fact, InfluxDB is designed more for high write throughput than for high read throughput [33].

Variable and client increment test cases showed that cpmPlus History is highly scalable for higher loads and higher data requests. InfluxDB showed that when requesting data with multiple variables, the performance started to decrease. From timespan and variable read test cases can be seen that InfluxDB suffers from using HTTP as the communication protocol and JSON response format as with higher data requests the response time started to grow significantly. cpmPlus History which used WebSocket did not have a similar performance impact. Client increment did not affect on InfluxDB's performance which makes it still scalable for high connectivity. Variable increment test case strengthened the view that TimescaleDB performs better for data reading as it showed almost similar read speed as cpmPlus History. TimescaleDB had more divergent performance than the others which do not give the status of high quality and reliability for the database.

Mix test showed that cpmPlus History could serve concurrent data writing and reading operations making it capable of monitoring type of use cases. InfluxDB did not manage to retrieve the latest values for the client which can be a significant factor for IoT and especially monitoring use cases. In fact, InfluxDB is designed with that value in mind that being able to write and read data is more important than having a consistent view and hence latest values cannot be retrieved [33]. InfluxDB also showed heavy resource usage among the three databases during the mix testing. cpmPlus History performed better with less resource usage making it a more efficient database. TimescaleDB also suffered performance lost in write speed, and it did not manage to return all the requested data. This shows that TimescaleDB is not suitable for monitoring use cases at least in used hardware size and as a single node solution.

8.2. Evaluation of the test bench and research objectives

The first research question was to identify functional and non-functional requirements for a time series database in IIoT environment. As stated in the introduction, the current problem is the lack of knowledge of the requirements for time series processing among the system developers. This lack of knowledge results to ineffective way of comparing different time series databases and typically also to wrong choices. By interviewing operators on the specified field, the requirements were identified and listed in chapter 4. The identified requirements have a high inheritance from the IIoT characteristics and potential use cases. Hence the requirements are on a similar spectrum with the characteristics and therefore fulfills them. The created list can be used for analyzing own business scenario and identify the weight of each requirement on the user's own scenario. The list can be used for going through possible time series databases as done in chapter 6.1 and that way identify the right time series database for the user's business scenario. The identified requirements list also presents a more unobstructed view of what time series databases are and what kind of usage they are for. For more general technical usage, the requirements list can be used for presenting how time series database differs from the more general relational database. Hence the first research question was successfully answered.

From the IIoT characteristics and the implementations of earlier test benches one key design feature raised. It was that the usage scenarios could differ a lot and hence the test bench should support simulation of different scenarios. The implemented test bench is highly configurable, and it does not use some specific dataset, like the earlier implementations, which makes it able to support many different scenarios. As written datapoints and read queries are kept simple for measuring highest performance, the test bench is not suitable for measuring the performance of more complex queries. As seen from databases' requirement analysis, there can be high variation in supported functions and processing tools so implementing comparable queries for several databases would be impractical.

As most of the earlier implementations lacked the industrial point of view from measuring and evaluation, in this thesis developed test bench' the industrial point of view is brought by testing three common industrial scenarios: on-premise, PIMS, and monitoring. Used write method also mimics the IIoT scenario where the data is written on high frequency rather than bulk loading. Also, testbench can be used for longer test runs, and it measures a larger number of metrics from the database than the earlier implementations.

From earlier implementations, Bader's TSDBBench [49] was closest to the required test bench. In this thesis developed test bench differs on the type of data written and queried but TSDBBench allows automatic database deployment and configurations for the test. In the developed test bench, the database needs to be deployed manually to a virtual machine, and all necessary configurations also need to be done manually. TSDBBench is hence in usability point of view more advanced.

Overall first and second research objective was successfully reached, and performance tests showed that cpmPlus History performs best from the three tested databases and acts as IIoT environment demands: high write throughput, scalable for the big data request, and scalable for large connectivity. Similar results to earlier measurements indicated that the first research objective was successfully reached. InfluxDB suffers from HTTP and JSON protocol usage as it creates performance lost in high write frequency and large data requests. TimescaleDB showed high performance for data reading, but it lacks demanded performance for data writing. It can be argued that the use of additional index reduces write throughput and increases disk usage, but for efficient data reading and simulation of a real environment, it is required. In general aspect, results showed that business product which is targeted for industrial usage outperforms general and open-source products. This supports the general assumption that with the higher price comes better performance. Evaluation of these three different price level databases also presented for the more general audience what kind of functionalities and performances can be reached at different price levels. Performance results can also be used in the future as a reference level for product development or other performance benchmarking purposes. It must be noticed that the presented results are measured for single node solution with limited hardware size. Tested databases can perform differently as a clustered solution and with bigger hardware. The assumption would be that the write and read throughput would be higher with more computing power.

In overall developed test bench and identified functional and non-functional requirements help system developers in IIoT field to find the best time series database for their business scenario and identify the important requirements of a time series database in their business scenario. Requirements list can be used as a checklist for evaluating own business scenario and how meaningful that requirement is for the scenario. From the practical example of database analysis based on identified requirements and developed test bench, differences on available time series databases were extensive. Database's performance level might differ between writing and reading, and hence it can be suitable for different parts of larger IoT system. Thesis' research questions were successfully answered, and research objectives were met. Thesis broad light to time series databases on IIoT and what are some key concepts and requirements for them.

8.3. Future work

Future work of the thesis is targeted for the future development of the test bench. As stated above the usability and automation of the database can be developed further. Data production could also be multiplied into multiple clients. The results from one client and two clients write tests could be then compared to verify that the measured limit is genuinely the performance limit of the database and not the limit of the client. Similarly, for more precise simulation results tested database might need to be on physical hardware, and the test should be run with more extensive data sizes. The current implementation is bonded to virtual machines only for telemetry recording purposes which allows the testbench to be run even for physical hardware solutions. Test bench could also be further developed by allowing custom read queries and performance test of the more complex time series functions that the database supports. Currently the read queries are simple and fixed. Bader's TSDBBench is in this sense more advanced as it uses scan, avg, sum, and count functions in its test queries. Of course, this requires that the tested databases support the used functions which might limit the number of the possible databases to be tested.

9. CONCLUSION

The problem that this thesis was targeted to provide a solution for, was the lack of knowledge of the requirements of the time series database in IIoT. The lack of knowledge derived from the inefficient way of comparing different time series databases. The purpose of this thesis was to identify the functional and non-functional requirements of time series database in IIoT and design and implement a performance test bench for time series databases. A practical example of how time series databases can be compared with identified requirements and developed test bench was also needed to provide.

Thesis provided a theoretical examination of IIoT, time series data, time series databases, and benchmarking which was used for identifying the characteristics that the scope of the thesis has. Current time series database market was also examined, and key characteristics of the databases were presented. For performance test bench design, earlier implementations were examined and evaluated against the thesis' topic. To help identify different requirements, different use cases of time series database in IIoT were presented. Use cases consisted of write heavy, read heavy, and concurrent write and read operation scenarios.

Identified functional requirements were deployment/platform support, time series support, datatype support, aggregation support, API's, information models, hierarchical systems, and write-to-device support. Non-functional requirements were identified to relate to scalability, security, availability, performance, resource usage, costs, engineering experience, and lifecycle. The requirements were also explained and reasoned in more detail.

Earlier test bench implementations showed lack of industrial point of view while testing the time series database. The developed test bench in this thesis took IIoT point of view by testing the database in three scenarios: write heavy, read heavy, and concurrent write and read operations. Test bench measured database's write and read throughput and resource usage. Test bench's another main difference was the level of configurable.

Three databases were selected for evaluation: ABB's cpmPlus History, InfluxDB, and TimescaleDB. Requirements evaluation of the three databases showed that cpmPlus History performed best against the identified requirements as it is targeted for industrial use cases. InfluxDB performed second best as its most critical lack was the support for hierarchical systems, information models, and support for OPC protocols. TimescaleDB lacked critically in core functionality as it had limited support for data aggregation, size management, and no support for alarms and events. The developed test bench was also applied to these three databases. Test results showed that cpmPlus History performs best from the three tested databases and acts as IIoT environment demands: high write throughput, scalable for the big data request, and scalable for large connectivity. InfluxDB suffers from HTTP and JSON protocol usage as it creates performance lost in high write frequency and large data requests. TimescaleDB showed high performance for data reading, but it lacks demanded performance for data writing.

In overall the contributions of this thesis are the developed test bench and identified functional and non-functional requirements. They can help system developers in IIoT field to find the best time series database for their business scenario and identify the important requirements of a time series database in their business scenario.

10. REFERENCES

[1] Lin S, Miller B, Durand J, Joshi R, Didier P, Chigani A, Torenbeek R, Duggal D, Martin R & Bleakley G (2015) Industrial internet reference architecture. Industrial Internet Consortium (IIC), Tech.Rep .

[2] Sadiku MN, Wang Y, Cui S & Musa SM (2017) Industrial internet of things. IJASRE 3.

[3] I-Scoop (2018) The Industrial Internet of Things (IIoT): the business guide to Industrial IoT. URI: https://www.i-scoop.eu/internet-of-things-guide/industrial-internet-things-iiot-saving-costs-innovation/. Cited 8.6.2018.

[4] Elrod K (2016) IoT, IIoT, Industry 4.0: What's the difference and does it matter? - Sealevel. URI: http://www.sealevel.com/community/blog/iot-iiot-industry-4-0-whats-the-difference-and-does-it-matter/. Cited 8.7.2018.

[5] Chan B (2018) IoT vs. Industrial IoT: 10 Differences That Matter. URI: https://www.iotforall.com/iot-vs-industrial-iot-differences-that-matter/. Cited 8.7.2018.

[6] ABB (2017) Unpublished internal document.

[7] Mishra SK (2016) Handling the Unstructured Data in IOT. ICJTA 9(37): 377-385.

[8] Lee YT (1999) Information modeling: From design to implementation. Proceedings of the second world manufacturing congress. , International Computer Science Conventions Canada/Switzerland: 315-321.

[9] OPC Foundation (2018) Unified Architecture. URI: https://opcfoundation.org/about/opc-technologies/opc-ua. Cited 25.7.2018.

[10] Gilchrist A (2016) Industry 4.0: the industrial internet of things. , Apress.

[11] ABB (2017) ABB Annual Report 2017. URI: https://new.abb.com/docs/default-source/investor-center-docs/annual-report/annual-report-2017/abb-group-annual-report-2017-english.pdf. Cited 28.6.2018.

[12] Masson C (2018) Five takeaways from Hannover Messe 2018 - Internet of Things. URI: https://blogs.microsoft.com/iot/2018/05/10/five-takeaways-from-hannover-messe-2018. Cited 14.6.2018.

[13] Frost & Sullivan (2018) The major challenges facing the IIoT solution market in 2018. URI: https://iiot-world.com/connected-industry/frost-sullivan-the-major-challenges-facing-the-iiot-solution-market-in-2018/. Cited 30.8.2018.

[14] IndustryARC (2016) Industrial Internet of Things (IIOT) Market to Reach $123.89 Billion by 2021. URI: https://industryarc.com/PressRelease/60/industrial-internet-of-things.html. Cited 12.6.2018.

[15] ABB (2018) ABB Ability. URI: https://new.abb.com/abb-ability. Cited 12.6.2018.

[16] GE (2018) Predix Platform | GE Digital. URI: https://www.ge.com/digital/predix-platform-foundation-digital-industrial-applications. Cited 12.6.2018.

[17] GE (2018) Exelon Optimizes Wind Forecasting Accuracy with GE's Predix Platform. URI: https://www.ge.com/digital/stories/exelon-optimizes-wind-forecasting-accuracy-ge-s-predix-platform. Cited 12.6.2018.

[18] Bosch Software Innovations (2018) IoT platform. URI: https://www.bosch-si.com/iot-platform/bosch-iot-suite/homepage-bosch-iot-suite.html. Cited 12.6.2018.

[19] Distence (2018) Condence – IIoT platform. URI: https://www.distence.fi/en/condence-iiot-platform/. Cited 25.10.2018.

[20] Siemens (2018) MindSphere - open IoT operating system. URI: https://www.siemens.com/global/en/home/products/software/mindsphere.html. Cited 25.10.2018.

[21] IBM (2018) Watson IoT Platform - IBM Watson IoT. URI: https://www.ibm.com/internet-of-things/solutions/iot-platform/watson-iot-platform. Cited 25.10.2018.

[22] Amazon (2018) Amazon Web Services (AWS) - Cloud Computing Services. URI: https://aws.amazon.com/. Cited 25.10.2018.

[23] White J (2018) Microsoft will invest $5 billion in IoT. Here’s why.. URI: https://blogs.microsoft.com/iot/2018/04/04/microsoft-will-invest-5-billion-in-iot-heres-why/. Cited 12.6.2018.

[24] George S (2018) Hannover Messe 2018: Manufacturers Put Their Trust in Microsoft’s Industrial IoT platform. URI: https://blogs.microsoft.com/iot/2018/04/23/hannover-messe-2018-manufacturers-put-their-trust-in-microsofts-industrial-iot-platform/. Cited 12.6.2018.

[25] Ericsson (2018) Audi and Ericsson to pioneer 5G for automotive manufacturing. URI: https://www.ericsson.com/en/press-releases/2018/8/audi-and-ericsson-to-pioneer-5g-for-automotive-manufacturing. Cited 30.8.2018.

[26] Brillinger DR (2000) Time series: general. Int.Encyc.Social and Behavioral Sciences .

[27] Kulkarni A (2017) What the heck is time-series data (and why do I need a time-series database)? . URI: https://blog.timescale.com/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-database-dcf3b1b18563. Cited 7.6.2018.

[28] Leighton B, Cox SJ, Car NJ, Stenson MP, Vleeshouwer J & Hodge J (2015) A best of both worlds approach to complex, efficient, time series data delivery. International Symposium on Environmental Software Systems. , Springer: 371-379.

[29] Dunning T & Friedman BE (2014) Time Series Databases: New Ways to Store and Access Data. , O'Reilly Media.

[30] InfluxData (2018) Time Series Database (TSDB) Explained. URI: https://www.influxdata.com/time-series-database/. Cited 7.6.2018.

[31] Basho (2018) Time Series Databases. URI: http://basho.com/resources/time-series-databases/. Cited 7.6.2018.

[32] DB-Engines (2018) DB-Engines Ranking per database model category. URI: https://db-engines.com/en/ranking/time+series+dbms. Cited 28.6.2018.

[33] InfluxData (2018) InfluxDB 1.5 documentation. URI: https://docs.influxdata.com/influxdb/v1.5/. Cited 8.6.2017.

[34] Sirosh J (2017) Announcing Azure Time Series Insights. URI: https://azure.microsoft.com/en-us/blog/announcing-azure-time-series-insights/. Cited 12.6.2018.

[35] Microsoft (2018) Azure Time Series Insights Documentation. URI: https://docs.microsoft.com/en-us/azure/time-series-insights/. Cited 12.6.2018.

[36] Vertica (2018) Vertica Overview. URI: https://www.vertica.com/overview/. Cited 13.6.2018.

[37] OpenTSDB (2018) OpenTSDB - A Distributed, Scalable Monitoring Sytem. URI: http://opentsdb.net/overview.html. Cited 12.6.2018.

[38] InfluxData, Hajek V, Klapka T & Kudibal I (2018) Benchmarking InfluxDB vs. OpenTSDB for Time Series Data, Metrics & Management. URI: https://www.influxdata.com/resources/benchmarking-influxdb-vs-opentsdb-for-time-series-data-metrics-and-management/. Cited 12.6.2018.

[39] KairosDB (2018) Getting Started - KairosDB 1.0.1 documentation. URI: https://kairosdb.github.io/docs/build/html/GettingStarted.html. Cited 13.6.2018.

[40] AWS (2018) What is Elasticsearch ? URI: https://aws.amazon.com/elasticsearch-service/what-is-elasticsearch. Cited 13.6.2018.

[41] elastic (2018) Basic Concepts | Elasticsearch Reference [6.4]. URI: https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html. Cited 13.6.2018.

[42] Timescale (2018) TimescaleDB Docs. URI: https://docs.timescale.com/v0.9/main. Cited 13.6.2018.

[43] Timescale (2017) Github - timescale/timescaledb. URI: https://github.com/timescale/timescaledb. Cited 13.6.2018.

[44] Crate.io (2013) CrateDB Reference - CrateDB Documentation. URI: https://crate.io/docs/crate/reference/en/latest. Cited 17.10.2018.

[45] Lilja DJ (2005) Measuring computer performance: a practitioner's guide. , Cambridge university press.

[46] AgileData (2002) Database Testing: How to Regression Test a Relation Database. URI: http://www.agiledata.org/essays/databaseTesting.html. Cited 26.6.2018.

[47] TPC (2018) Actice TPC Benchmarks. URI: http://www.tpc.org/information/benchmarks.asp. Cited 12.6.2018.

[48] Cooper BF, Silberstein A, Tam E, Ramakrishnan R & Sears R (2010) Benchmarking cloud serving systems with YCSB. Proceedings of the 1st ACM symposium on Cloud computing. , ACM: 143-154.

[49] Bader A (2016) Comparison of time series databases.

[50] STAC (2010) STAC-M3 Central. URI: https://stacresearch.com/m3. Cited 12.6.2018.

[51] Schmakeit M, Stinner F, Ziegeldorf DJH, Wehrle IK & Müller ID (2017) Performance Evaluation of Low-Overhead Messaging Protocols and Time Series Databases via a Common Middleware.

[52] Wlodarczyk TW (2012) Overview of time series storage and processing in a cloud environment. Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. , IEEE: 625-628.

[53] Goldschmidt T, Jansen A, Koziolek H, Doppelhamer J & Breivold HP (2014) Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on. , IEEE: 602-609.

[54] InfluxData (2018) Comparing InfluxData Products | InfluxDB versus Other Platforms. URI: https://www.influxdata.com/products/compare/. Cited 11.6.2017.

[55] Mathe Z, Ramo AC, Stagni F & Tomassetti L (2015) Evaluation of NoSQL databases for DIRAC monitoring and beyond. Journal of Physics: Conference Series. , IOP Publishing 664: 042036.

[56] Rudolf C (2017) SQL, noSQL or newSQL–comparison and applicability for Smart Spaces. Network Architectures and Services .

[57] Lautenschlager F, Kumlehn A, Adersberger J & Philippsen M (2015) Fast and efficient operational time series storage: The missing link in dynamic software analysis. Proceedings of the Symposium on Software Performance (SSP 2015). Edited by G. eV. Softwaretechnik-Trends. 3: 46.

[58] Vaughan-Nichols SJ (2018) What is Docker and why is it so darn popular?. URI: https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/. Cited 15.6.2018.

[59] Lustosa H, Porto F, Valduriez P & Blanco P (2016) Database system support of simulation data. Proceedings of the VLDB Endowment 9(13): 1329-1340.

[60] Masson C (2018) Why the OPC UA Standard – and What's Next? - Internet of Things. URI: https://blogs.microsoft.com/iot/2018/04/11/why-the-opc-ua-standard-and-whats-next. Cited 15.6.2018.

[61] Boyer SA (2009) SCADA: supervisory control and data acquisition. , International Society of Automation.

[62] Schroeder GN, Steinmetz C, Pereira CE & Espindola DB (2016) Digital twin data modeling with automationML and a communication methodology for data exchange. IFAC-PapersOnLine 49(30): 12-17.

[63] Bolton D (2016) What Are Digital Twins And Why Will They Be Integral To The Internet Of Things?. URI: https://www.applause.com/blog/digital-twins-iot-faq/. Cited 15.6.2018.

[64] Smith A (2013) The Triple-A Approach to Enterprise IT Security. URI: https://www.ecommercetimes.com/story/76987.html. Cited 19.6.2018.

[65] Anderson JM (2003) Why we need a new definition of information security. Comput Secur 22(4): 308-313.

[66] Welander P (2010) Control System Lifespan: How Long is Long Enough?. URI: https://www.controleng.com/single-article/control-system-lifespan-how-long-is-long-enough/95e56d4b766003cc3022e12ffe7d9044.html. Cited 18.6.2018.

[67] Schroll M (2018) Towards 3B time-series data points per day: Why DNSFilter replaced InfluxDB with TimescaleDB. URI: https://blog.dnsfilter.com/3-billion-time-series-data-points-dnsfilter-replaced-influxdb-with-timescaledb-d9f827702f8b. Cited 23.8.2018.

[68] Andreev I (2017) Time Series Databases for IoT (On-premises and Azure). URI: https://www.slideshare.net/ivoandreev/time-series-databases-for-iot-onpremises-and-azure. Cited 12.9.2018.

[69] Kiefer R (2017) TimescaleDB vs. PostgreSQL for time-series data – Timescale. URI: https://blog.timescale.com/timescaledb-vs-6a696248104e. Cited 12.9.2018.