Application of Time Series Database for IoT Smart City Platform

Petr John, Jiří Hynek, Tomáš Hruška, Michal Valný

Abstract—**IoT devices are becoming more prevalent yearly due to their relatively low cost and high maintainability. Their use cases are often varied–from industry (e.g., devices intended for quality control) to their application in smart cities, where they can help supervise day-to-day operations. With the increase in popularity comes the need to effectively store, query and process the data produced by such devices. This problem can seem relatively easy at first glance, but often, the solutions are not as straightforward. In real-world applications, multiple device types are needed to accomplish each task. This typically results in large streams of incoming data, each with its own structure and data representation. Due to the large amounts of devices, it may be necessary to distribute the workload or use serverless computing to tackle fluctuations in incoming requests. This paper provides a study of some options that can be used to achieve this goal, focusing on the efficiency and ease of use of both SQL and NoSQL databases currently available regarding deployment on both local devices and serverless functions like AWS Lambda. Document or relational databases are often used to store data from such devices. However, features of Time Series databases could be used to improve the entire system's performance and reduce storage requirements. One of the systems that could benefit from this improvement is a Smart City system developed by Logimic. The cost of the proposed smart city platform is essential because a high running cost can result in the system being more expensive than the cost reduction from using efficient IoT devices.**

Index Terms— **IoT, Smart City, Database, NoSQL, Time Series, AWS, Serverless computing**

I. Introduction

IoT devices have been rising in popularity in recent years. Currently, there are estimated 10 – 11 million IoT devices [1, 2] in use. Their low cost and high practicality can be considered the main reason for this rise in popularity. One of the examples can be seen in Smart Cities, where such devices can significantly improve the day-to-day experience of their inhabitants. This can range from smart bin management [3] to measure the amounts of dangerous oxides in urban areas [4]. IoT devices in cities might become necessary to keep up with the rise in world population and urbanization. Currently, 55% of the population lives in urban areas, and the United Nations estimates the rise to be 68% [5, 6]. Both India and China are investing heavily to sustain their rising population, with China currently funding more than 200 smart city pilots [7] and India announced 100 smart cities [8]. Both trends result in large amounts of data that must be stored and effectively queried. Due to the amount of incoming data, choosing a database system that can store them efficiently is necessary to create a solution with sustainable storage requirements. Sometimes, it may be possible to delete older samples and reduce the storage space. However, there may be situations where downsampling or entirely deleting old measurements may not be possible – for example, due to non–functional legal requirements or SLAs [9].

On the other hand, in many cases, the user may want to get aggregated data. An example of this situation is when the user wants to display a large segment of data. Loading the raw data without aggregating them can result in the user UI taking too long to load, which can cause the user to stop using the page entirely [10]. A correctly selected aggregation function can reduce the number of points with no or acceptable accuracy loss [9]. This can create a significant requirement for the storage used to handle the data and the API providing access to this data. Some database types provide tools to run aggregation functions directly in the database, but in other cases, the API needs to be able to run the appropriate function efficiently. This can be achieved by third-party software like Apache Spark [11], or any other distributed map-reduce method [12].

Another concern is the performance of the storage, which needs to be able to handle the stream of incoming data and the option to distribute the same database between multiple network nodes, to reduce the load on each node. This creates a problem for relation databases, which must keep all of the data stored consistent [13]. NoSQL and NewSQL databases often allow the creation of multiple nodes by either not requiring a strong consistency (often NoSQL) or by not sharing data between the nodes (some NewSQL databases). This depends on the selected database system, but attaining both simultaneously is impossible due to the CAP theorem [14]. The data from smart cities can be classified as time series data,

First submitted 17. 2. 2023.

This work was supported by project Smart information technology for a resilient society, FIT-S-23-8209, funded by Brno University of Technology.

Petr John is a doctoral student at Brno University of Technology, Brno, 612 00 Czechia, (e-mail: ijohn@fit.vutbr.cz)

Jiří Hynek is with the Brno University of Technology (e-mail: hynek@fit.vutbr.cz)

Tomáš Hruška is with the Brno University of Technology (e-mail: hruska@fit.vutbr.cz)

Michal Valný is a CEO of Logimic, Jiříkovice, 664 51 Czechia (e-mail: michal.valny@logimic.com)

as they are repeated measurements over time, with regular or irregular time intervals. Optimizations and special database systems currently exist to handle such data, but other database types are often preferred for data storage.

All of those requirements combined can result in situations where a custom server can be inefficient from a monetary and practical standpoint. Due to this, the solution should be ready to run in a cloud or serverless environment. Differences between deployment options can restrict us from using some database technologies, as many database systems created by cloud providers are either only available on their respective cloud solutions or behave differently when used locally.

Logimic is one of the companies attempting to overcome all the mentioned obstacles and create a platform that would not only allow the efficient storage and querying of the data from IoT devices but also allow the user to display and analyze both the current state and historical data. Specialized time series databases or databases with time series optimizations could bring both performance and monetary improvements, thanks to the possible reduction in storage requirements or increased performance due to specialized queries. Time series optimizations are commonly implemented in NoSQL systems rather than traditional relational databases.

The use of such databases could result in the reduction of costs, electricity, and resource requirements. In the case of cloud installations, systems are often charged for using the software as a service (SaaS) model. Due to this, using a more efficient database system does not directly translate to cost reduction, as software providers often charge for different metrics (e.g., amount of data stored, number of queries, transferred data). While cities can reduce their electricity spending by installing IoT devices (e. g. smart lamp posts), an inefficient platform can result in the city paying more than before the transition. Due to this, the effectiveness of the smart city platform is paramount to its success.

This paper outlines available database options, the practicality of their use for storing the data from IoT devices, the performance of the selected database systems, the pros and cons of using time series databases, and the monetary difference between DynamoDB and InfluxDB. The proposed solution was tested on a real smart city platform developed by Logimic and resulted in a 40% savings in running costs, increased storage efficiency, and added support for new functionality that was not previously supported.

II. Current Solutions

Database systems are typically used to store data efficiently. Database systems evolved from older file system databases to commonly used relational databases, lately very popular NoSQL databases to new NewSQL database systems that combine features of both NoSQL and relational databases [15].

Relational databases currently represent the most often used solution [16] for effectively and persistently storing data. Relational databases use the relational algebra outlined by Edgar F. Codd [17] to describe the structure and relations. Most of the time, the data is also normalized [18] to increase the querying speed and reduce the size of stored data by removing duplicates, which is necessary to keep the data consistent without propagating the changes manually.

While relational databases represent an often and easy solution to storing traditional data, the scalability and speed of modern NoSQL databases proved to be a more than adequate alternative in situations, where the performance of relational databases isn't sufficient, or the data need to be distributed between multiple nodes. Another key difference is the ability of NoSQL databases to store data with loose or no predefined structure. This can be beneficial when working with Big Data or data with an unknown structure, as can be the case in many IoT applications [19, 20]. Due to those limitations, relational databases are typically used to store and manage metadata [21] or require a use of architecture, allowing the data to be split into multiple database instances [22]. One of the most popular [16] relational databases, often famed for its performance, is PostgreSQL, which was used during testing.

Key-value databases may be considered the simplest NoSQL database type. DynamoDB, developed by Amazon and heavily integrated into AWS. The main advantage of key-value databases is their ability to quickly store elements without the need to keep or define the data structure. However, they often do not allow querying by non-primary fields and do not provide support for native aggregation functions. Due to the availability and scalability, the database can be used for storing metadata [23] or as the primary data storage used [24] to store information about railway tickets. DynamoDB is further analyzed since it was previously used by Logimic.

A very similar type to key-value databases are document databases. Document databases often allow users to create indexes over more fields or support more robust querying languages. MongoDB is a good representative of this database type. It can be considered the most popular NoSQL database [16]. It is often used in smart city systems such as the system presented in [25], which uses MongoDB time series mode (introduced in version 5) or other Big Data situations [26]. Not all document databases support querying languages like MongoDB, but they are still considered and used in Smart Cities. An example could be CouchDB coupled with Apache Spark over a distributed file system [27]. MongoDB was further analyzed due to its popularity.

Not all NoSQL databases can be used for general purposes. Some of them are directly specialized for one task only. In this case, time series databases are directly optimized for work with time series, like the data from IoT sensors. The most popular of these databases is InfluxDB [28]. InfluxDB, similarly to many time series databases, uses delta encoding [29], which significantly reduces the size of numerical series, that contain intervals where the parameter value is similar. InfluxDB is aimed at both cloud deployments, where it can be used as SaaS or local deployment with its Open-Source version. It was previously used in Smart City pilot projects related to Water quality monitoring [30, 31].

III. Testing Methodology

Using time series databases to replace relational and document NoSQL databases must first be tested in a controlled environment. While data from IoT devices can be generally classified as time series data (always dependent on time and being stored in rising order), general-use databases could handle the data more efficiently.

A three-step testing methodology was selected. In the first step, all of the selected database systems were subjected to some everyday situations: insertion of a new value, querying a time segment without an aggregation, querying a time segment with an aggregation, and querying the last element currently stored. These operations represent some of the commonly used queries. For example, aggregation and selecting raw data are needed when the data needs to be visualized for the user. Insertion is needed to store the incoming data. Querying the last inserted element can be used in cases where the current state of the connected IoT devices needs to be visualized or KPIs need to be re-evaluated.

A small test file (with a raw size of around 30 MB) was selected to filter out database systems that are very slow and thus unsuitable. All tests were run 20 times, and individual results were averaged. All tests were run on an Intel i5 7500HQ CPU with 16 GB of RAM.

The second step used the same scenarios described in the previous step but with a more extensive test file (more than 500 thousand records with a total size of around 350 MB). The space required to store the database was also measured to filter out databases with large storage requirements. While this dataset does not represent the most common structure, it can be used to emulate a worst-case scenario, as it contains many fields with much larger sample sizes than is typical for IoT devices in smart cities.

In the last step, a different dataset was used, this time more representative of the traditional data from smart cities. The dataset was created from multiple RHF1S001 sensors obtained over a year. The dataset was supplied by Logimic with a total size of 163 MB with around 670 thousand items.

IV. Results

The results of the first step show the disadvantages of DynamoDB and PostgreSQL. The performance of DynamoDB suffers due to the lack of a query language and the need to query the database repeatedly to get all of the queried items, resulting in low performance. On the other hand, DynamoDB excelled at querying and inserting a single item. MongoDB and InfluxDB fared much better, providing significantly more performance than both DynamoDB and PostgreSQL. While the speed of inserting a single item is essential due to the number of incoming items, computing the average can be a more significant obstacle as it directly impacts the user experience. The results of the first step are mentioned in **Figure 1**.

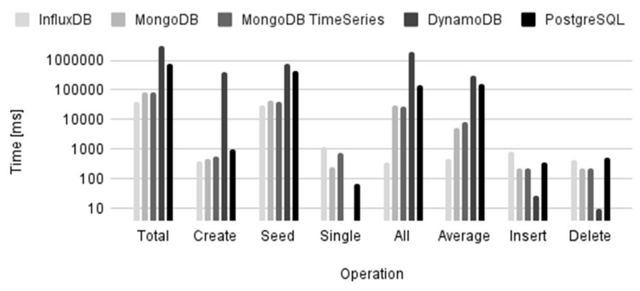

Figure 1: Results of the first step

This left only two viable database systems-InfluxDB and MongoDB. Both of those systems performed adequately. Generally, MongoDB performed better when the new time series mode was not enabled. The time series mode increased the time needed to perform each operation. For example, in the case of selecting a 5-minute average, the time needed to complete the query changed from 2699 ms to 4338 ms in the case of Mongo DB version 5 without and with time series mode, respectively. InfluxDB performed less favorably than MongoDB, completing the same 5-minute average in 4165 ms. InfluxDB only proved to be faster in the 5-minute average of 60 days. See **Figure 2** for more details.

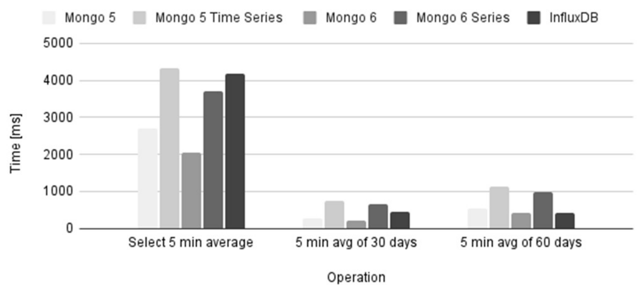

Figure 2: Results of the second step

MongoDB, in default document mode, often offered better performance than InfluxDB. The MongoDB time series mode performed similarly to the previous dataset. MongoDB version 6 often performed worse than MongoDB version 5. This could be due to a different default caching policy between versions. The performance of InfluxDB is often similar to or very close to MongoDB. The difference is often in the tens of milliseconds. In the case of 60-minute average of the last 120 days this difference was 12 ms between the InfluxDB and MongoDB in both versions.

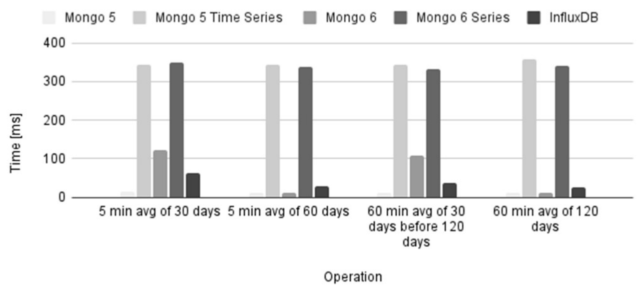

Figure 3: Results of the additional benchmark.

Another important metric is the size of the resulting database. The total size of the bucket data stored on disc[1] was selected because the InfluxDB Open-Source version does not currently support estimating the database size. Few of the available metrics were selected for MongoDB, as many of them (mainly the MognoDB 6 Series) differ from the actual disc space used. A comparison of the space required to store both datasets can be seen in **Figure 4** and **Figure 5**.

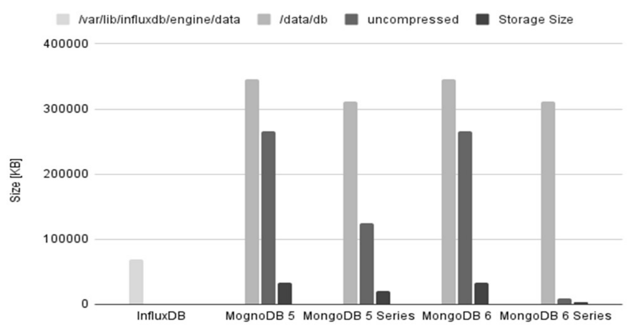

Figure 4: Size of the 350 MB data set in different databases.

The combination of performance and storage efficiency makes InfluxDB an interesting choice for smart city installations. Both MongoDB and InfluxDB show remarkable storage efficiency when compared to PostgreSQL. In the case of the 350 MB dataset, the total database size was 787 MB in the case of PostgreSQL, which is more than double the size of uncompressed MongoDB version 5 at 265 MB. InfluxDB was able to store the dataset on 69 MB. This reduction highlights the advantages of using specialized compression.

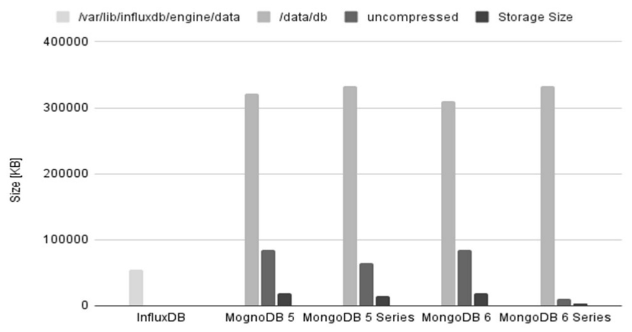

Figure 5: Size of the temperature data set in different databases.

V. CASE STUDY: LOGIMIC

The effectiveness of InfluxDB was demonstrated in a case study that was done in cooperation with Logimic, where InfluxDB replaced the existing DynamoDB database. While MongoDB was strongly considered, InfluxDB was chosen in the end for the combination of good performance and storage efficiency.

Logimic develops several IoT applications, many of them aimed at smart cities. These platforms range from general-use smart cities to specific platforms aimed at smart lamp posts, temperature sensors used in freezers in vaccination centers, water retention, and others. The AWS cloud solution, combined with The Things Network, is used for such solutions. This architecture was chosen because the company mainly uses MQTT-based devices. Data from these devices are sent to TTN, decoded from vendor-specific formats to more standardized ones, and then sent to AWS, where lambda handlers store the incoming data in databases. Logimic used a combination of DynamoDB for time series and metadata that needs to be accessed quickly and PostgreSQL for relational data and sensor metadata. Stored data are then used to evaluate any relevant KPIs. KPIs and raw data can be displayed in one of the frontend applications developed by Logimic.

Before the switch from DynamoDB to InfluxDB Logimic used a single AWS Lambda function to handle all the interaction with statistics stored in DynamoDB. During the switch, a new RESTful API was developed to accommodate the new database system and multiple new operations offered.

The switch resulted in a positive change in both performance and costs. While the operating costs of DynamoDB were constantly rising (due to the increasing amount of data that needed to be stored), InfluxDB doesn't display such behavior. The spike at 10. 3. 2022 in **Figure 6** was caused by copying data stored in DynamoDB to InfluxDB. DynamoDB currently continues to be used for managing configurations and other metadata. The cost of InfluxDB and DynamoDB combined was reduced to around 40% of the cost of DynamoDB in at 1. January 2022.

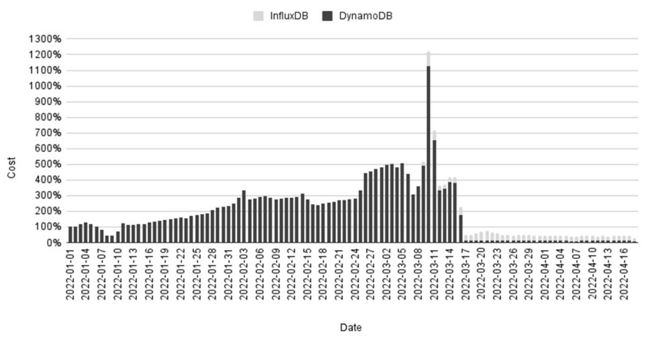

Figure 6: Cost of database systems used by Logimic during the transition.

VI. DISCUSION

In the final decision, InfluxDB was selected as a storage of the incoming data from IoT devices. This choice was a compromise between performance and used storage space. This choice can be a limiting factor, when compared to MongoDB. For example, InfluxDB does not allow mixing numerical and string data types in a single field or storing nested documents and provides no optimizations for accessing the last inserted element. MongoDB is better suited for applications where this proves to be a typical operation. While accessing the last element can be sped up by keeping the item in a different database, likely the same as the one used to keep sensor metadata, this creates added complexity. On the other hand, InfluxDB provides tools to easily request downsample

[1] This metric was selected because the InfluxDB Open-Source version 2.1 does not provide any other metric to measure the bucket size. This size corresponds to the total size of */var/lib/influxdb/engine/data* on traditional Linux distributions.

data, such as Telegraf, Tasks, and MQTT plugins.

The combination of AWS Lambda functions and RESTful API can result in a sub-optimal performance caused by cold starts. This could be mitigated by reducing the number of endpoints. On the other hand, this solution is better from the scalability and practical standpoints as it is easier to manage due to other aspects like caching and authorization. Cold starts can be an obstacle, primarily on systems and endpoints with low traffic. In these cases, deploying the application using a different solution, like self-hosting a server or deploying a docker container instead of serverless functions, may be beneficial.

Logimic currently stores metadata of the sensors in PostgreSQL. Moving them to a NoSQL database like DynamoDB or MongoDB could be beneficial to improve performance. Another solution to this problem may be caching or using the CQRS pattern. The current architecture may cause the UI to be slow as the sensor device eui needs to be queried from the relational database before querying the time series data. On the other hand, it may be possible to include the relational data directly in the response from InfluxDB thanks to InfluxDB's SQL integration options.

Both optimizations mentioned above can be explored in the future, as they represent a significant performance improvement when the number of connected devices and users rises. Another point that should be considered is data locality. While storing the data on the cloud is a convenient solution, it may be beneficial to distribute the data between data centers or self-hosted instances. Another possible disadvantage of a cloud-based solution may be internet outages in remote installations.

VII. Conclusion

This study outlines the practicality of using time series databases, especially InfluxDB, in smart city platforms. This approach was first tested in a controlled environment and then in a smart city platform developed by Logimic. InfluxDB performed similarly to MongoDB, which is often used in smart city platforms. Both databases often provide comparable performance and outperform traditional relational databases. The use of DynamoDB as primary storage is very limited.

On the other hand, it is an excellent choice for storing metadata or as a cache due to its high performance when inserting or retrieving a single item. The solution was deployed to a smart city platform, which resulted in reduced costs and increased performance. This development can aid Logimic provide an efficient platform by reducing both runtime and storage costs. Coupled with smart devices like smart street lightning, the platform can provide a simple and convenient way to save electricity by utilizing it only when needed.

References

[1] State of iot 2021: Number of connected iot devices growing 9% to 12.3 billion globally, cellular iot [online] Available: https://iot-analytics.com/number-connected-iot-devices

[2] A. Holst. Iot connected devices worldwide 2019-2030, Oct 2021.

[3] F. Annie Lincy and T. Sasikala. Smart dustbin management using iot and blynk application. In 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), pages 429–434, 2021.

[4] M. Schiavon, M. Redivo, G. Antonacci, E. C. Rada, M. Ragazzi, D. Zardi, and L. Giovannini. Assessing the air quality impact of nitrogen oxides and benzene from road traffic and domestic heating and the associated cancer risk in an urban area of verona (italy). Atmospheric Environment, 120:234–243, 2015.

[5] H. Ritchie and M. Roser. Urbanization. Our world in data, 2018.

[6] United Nations Publications. World Urbanization Prospects: The 2018 Revision. UN, 2019.

[7] P. Liu and P. Zhenghong. China's smart city pilots: A progress report. Computer, 47:72–81, 10 2014.

[8] S. Madakam and R. Ramaswamy. 100 new smart cities (india's smart vision). In 2015 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW), pages 1–6, 2015.

[9] J. Poncela, P. Vlacheas, R. Giaffreda, S. De, M. Vecchio, S. Nechifor, R. Barco, M. C. Aguayo-Torres, V. Stavroulaki, K. Moessner, et al. Smart cities via data aggregation. Wireless personal communications, 76(2):149–168, 2014.

[10] K. Eaton. How one second could cost amazon $1.6 billion in sales - fast company, Mar 2012 [online] Available: https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales.

[11] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, DB Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. The Journal of Machine Learning Research, 17(1):1235–1241, 2016.

[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan 2008.

[13] M. Rys. Scalable sql: How do large-scale sites and applications remain sql-based? Queue, 9(4):30–37, apr 2011.

[14] S. Gilbert and N. Lynch. Perspectives on the cap theorem. Computer, 45(2):30–36, 2012.

[15] C. Coronel and S. Morris. Database Systems: Design, Implementation, & Management. Cengage Learning, 2016.

[16] Db-engines ranking [online] Available: https://db-engines.com/en/ranking_trend.

[17] E. F. Codd. A Relational Model of Data for Large Shared Data Banks, pages 263–294. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[18] H. Köhler and S. Link. Sql schema design: foundations, normal forms, and normalization. Information Systems, 76:88–113, 2018.

[19] B. Silva, M. Khan, C. Jung, J. Seo, M. Diyan, J. Han, Y. Yoon, and K. Han. Urban planning and smart city decision management empowered by real-time data processing using big data analytics. Sensors, 18:2994, 09 2018.

[20] M. Ahmed, O. Shahat. A novel big data analytics framework for smart cities. Future Generation Computer Systems, 91:620–633, 2019.

[21] K. Takahashi, S. Yamamoto, A. Okushi, S. Matsumoto, and M. Nakamura. Design and implementation of service api for large-scale house log in smart city cloud. In 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, pages 815–820, 2012.

[22] F. G. Brundu, E. Patti, Anna O., M. Del Giudice, N. Rapetti, A. Krylovskiy, M. Jahn, V. Verda, E. Guelpa, L. Rietto, and A. Acquaviva. Iot software infrastructure for energy management and simulation in smart cities. IEEE Transactions on Industrial Informatics, 13(2):832–840, 2017.

[23] M. Krämer, S. Frese, and A. Kuijper. Implementing secure applications in smart city clouds using microservices. Future Generation Computer Systems, 99:308–320, 2019.

[24] G. M. D'silva, A. K. Scariah, L. R. Pannapara, and J. J. Joseph. Smart ticketing system for railways in smart cities using software as a service architecture. In 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), pages 828–833, 2017.

[25] M. Colosi, F. Martella, G. Parrino, A. Celesti, M. Fazio, and M. Villari. Time series data management optimized for smart city policy decision. In 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pages 585–594, 2022.

[26] Y., I. Park, J. Rhee, and Y. Lee. Mongodb-based repository design for iot-generated rfid/sensor big data. IEEE Sensors Journal, 16(2):485–497, 2016.

[27] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. Building a big data platform for smart cities: Experience and lessons from santander. In 2015 IEEE International Congress on Big Data, pages 592–599, 2015.

[28] Timeseries db-engines ranking [online] Available: https://db-engines.com/en/ranking_trend/time+series+dbms

[29] J. Xiao, Y. Huang, C. Hu, S. Song, X. Huang, and J. Wang. Time series data encoding for efficient storage: A comparative analysis in apache iotdb. Proc. VLDB Endow., 15(10):2148–2160, sep 2022.

[30] Y. Chen and D. Han. Water quality monitoring in smart city: A pilot project. Automation in Construction, 89:307–316, 2018.

[31] K. Shanmugam, M. E. Rana, D. T. Z. Xuen, and S. Aruljodey. Water quality monitoring system: A smart city application with iot innovation. In 2021 14th International Conference on Developments in eSystems Engineering (DeSE), pages 571–576, 2021.