On-Premise Internet of Things (IoT) Data Storage: Comparison of Database Management Systems

Anna Wolters[a], Mevludin Blazevic[b] and Dennis M. Riehle[c]
Institute for IS Research, University of Koblenz, Universitätsstraße 1, Koblenz, Germany

Keywords: Internet of Things, Big Data, Database Management System, Time-series Database.

Abstract: The Internet of Things (IoT) connects millions of devices, leading to the production of vast amounts of data. For such data to be of value, efficient and effective data storage is of utmost importance. In this paper, we present a comparison of on-premise database management systems in the context of the IoT. We perform a market analysis on relational, Not Only SQL (NoSQL), and time-series database systems as well as a requirement analysis in order to comprehensively compare database systems based on functional and non-functional criteria. After an initial selection, we compare MySQL, PostgreSQL, Cassandra, MongoDB, InfluxDB, and QuestDB. As a result, we provide a best practice guide to support the decision-making on which database to select for an IoT use case.

1 INTRODUCTION

Connected objects in the Internet of Things (IoT) produce vast amounts of data, making the data itself one of the essential elements in the IoT (Nasar & Kausar, 2019). The increasing amounts of data generated in the IoT create the challenge of storing the data efficiently, which is essential to ensure better decision-making based on the data. IoT data is generated mainly by IoT sensors which collect environmental data (Rayes & Salam, 2017). These data sets are too large to be analyzed with classic data-processing software and "have entered the common language with the term Big Data" (Sestino et al., 2020, p. 1). Moreover, IoT data and Big Data are equal in their emphasis on information that is rich in volume, velocity, and variety and therefore requires innovative forms of processing (Lycett, 2013; Sestino et al., 2020).

The characteristics that Big Data and IoT have in common, impose requirements for the database system. Those include technical requirements to, for instance, support heterogeneous sensor infrastructures (Amghar et al., 2018), as well as other factors such as data security, query speed, reliability, pricing, maintainer, business model (on-premise, cloud), licensing model, support, API, and interfaces (Beynon-Davies, 2017; Rautmare and Bhalerao, 2016). This makes the selection of the right database system solution challenging, as different requirements and contextual information are needed to find the right balance between costs and benefits.

In this paper, we present a best practice guide for selecting a suitable database system for storing IoT data on-premise. We perform a comprehensive comparison of available database systems by identifying functional and non-functional criteria to guide our selection process. In total, we compare six database systems, two of which are relational, two Not Only SQL (NoSQL), and two time-series databases.

The remainder of the paper is structured as follows. In Section 2 we provide background information on the IoT and Big Data, as well as database systems. Section 3 presents the research method applied. Next, we perform a market and requirement analysis in Section 4. The final set of selected databases is compared in more detail in Section 5. We discuss and conclude our findings in Section 6.

2 BACKGROUND

2.1 Internet of Things and Big Data

Since its first appearance, many definitions for IoT were established which can be found in scientific and non-scientific literature (e.g., Baiyere et al., 2020; Chui et al., 2020; ITU-T, 2011; Miorandi et al., 2012; Rayes & Salam, 2017). Mostly, these definitions have the interrelationship between sensors and physical objects in common. IoT can be defined as a "system

[a] https://orcid.org/0000-0002-4075-5737
[b] https://orcid.org/0000-0003-0347-7392
[c] https://orcid.org/0000-0002-5071-2589

Wolters, A., Blazevic, M. and Riehle, D.
On-Premise Internet of Things (IoT) Data Storage: Comparison of Database Management Systems.
DOI: 10.5220/0011851200003482
In *Proceedings of the 8th International Conference on Internet of Things, Big Data and Security (IoTBDS 2023)*, pages 140-149
ISBN: 978-989-758-643-9; ISSN: 2184-4976

of interconnections between digital technologies and physical objects that enable such (traditionally mundane) objects to exhibit computing properties and interact with one another with or without human intervention" (Baiyere et al., 2020, p. 557). Another definition focuses on IoT applications and information technology, whereupon IoT is the "interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications" (Gubbi et al., 2013). This can be achieved by pervasive sensing and data processing using cloud infrastructures as a unifying architecture. IoT sensors generate a variety of different data that differs from data from traditional Information Systems (IS). One approach in describing IoT data is the distinction between the origins of data that is processed by computers. IoT data is generated mainly by IoT sensors which collect environmental data (Rayes & Salam, 2017). Moreover, IoT enables different physical devices to connect to the Internet and engage in data exchange. These datasets are too large to be analyzed with classic data-processing software and can be seen as a subcategory of "Big Data" (Sestino et al., 2020).

The term Big Data refers to the generation of large amounts of structured and unstructured data. Big Data is usually associated with real-time analysis, which distinguishes it from traditional datasets (Chen et al., 2014). Originally, Big Data was defined by the *3 V's*: volume, velocity, and variety (Strohbach et al., 2015), while further characteristics such as value, veracity, and visualization can be considered as well (Chen et al., 2014; Strohbach et al., 2015). IoT is considered to be one of the most promising drivers of Big Data expansion, due to the increasing availability of connected sensor devices in companies that generate a vast amount of data (De Mauro et al., 2015).

2.2 Database Systems

When discussing and comparing database systems in more detail, the *CAP* theorem and *ACID* properties must be considered. Both ideas describe characteristics of distributed systems, and the ACID properties particularly focus on transactional databases. The CAP theorem contains the characteristics of consistency, availability, and partition tolerance. According to the theorem, a distributed system is only able to fulfill two of these properties . The ACID properties are atomicity, consistency, isolation, and durability (Elmasri & Navathe, 2016, pp. 754-755).

In recent years, database or storage model technologies have evolved and improved significantly. In our research, we focus on relational, NoSQL, and time-series database systems.

The idea of a relational data model was introduced by Codd (1970) in 1970. In a relational database, data is structured in the form of tables that are eventually connected to each other using unique identifiers as reference keys (Elmasri & Navathe, 2016, p. 73), and the data is accessed using Structured Query Language (SQL) (Elmasri & Navathe, 2016, p. 59). Relational databases provide a very simple data structure as well as data access. With the concept of data normalization, data is structured clearly and data consistency is improved (Elmasri & Navathe, 2016, pp. 501 ff.). However, relational databases are not able to store more complex data formats, such as documents.

Compared to relational database systems, NoSQL databases provide flexible and schema-less data models (Fatima & Wasnik, 2016), but do not provide the ACID characteristics characteristics (Chauhan & Bansal, 2017). However, it adheres to *BASE*, which means *basically available*, *soft state*, and *eventually consistent* (Chauhan & Bansal, 2017). There exist four categories of NoSQL databases: key-value store, column-oriented database, document database, and graph database. Besides a more flexible data model, NoSQL databases also support horizontal scaling, while relational databases can only be scaled vertically (Chauhan & Bansal, 2017).

In IoT, time-series databases are very prominent, since IoT data usually contain a timestamp for every data point, which might be used to index the data. Time-series databases are *insert-heavy*, while there is no update operation as it is the case for relational and NoSQL databases (Mostafa et al., 2022). The time-based storage enables easier analysis of trends and changes in data over time. Additionally, querying and fetching data from a time-series database is usually very efficient (Musa et al., 2019). Time-series databases do not necessarily adhere to the ACID properties. Because of the continuous collection of new data, durability, and strong consistency do not have to be ensured. InfluxDB, a prominent time-series database, for instance, supports eventual consistency (Musa et al., 2019).

3 RESEARCH METHOD

In this research, we apply Design Science Research (DSR) based on the Design Science Research Methodology (DSRM) proposed by Peffers et al. (2007). As given by Peffers et al. (2007), artifacts can be "constructs, models, methods, or instantiations" (Peffers et al., 2007), making the knowledge contribu-

tion of DSR versatile (Gregor & Hevner, 2013). In our research, the final artifact is a best practice guide to assist decision-makers in selecting the most appropriate database system for IoT use cases. Aligned with the understanding of an artifact in DSR, we present a more abstract artifact similar to design principles or implicit technological rules (Gregor & Hevner, 2013).

Our research follows the six phases proposed in the DSRM: Based on these phases, our research is structured as follows:

Problem Identification. Storing IoT data is a challenging task since IoT devices produce vast amounts of data – *Big Data* – which must be stored and processed efficiently. Choosing the right database is just as challenging as there exists a large variety of database systems. Deciding on which system to use requires a comprehensive overview of available systems and their strengths and weaknesses.

Objectives. The aim of this research is a comparison of available database systems regarding their functionality in the context of IoT. Requirements are collected based on related research, market research, and given restrictions on the scope of the study.

Design and Development. Initially, a market and requirement analysis is performed to identify database systems for a detailed comparison. The selected database systems are installed and tested on-premise to collect data on their performance.

Demonstration. Results of the tests are consolidated, compared, and displayed in an understandable format.

Evaluation. Resulting data is evaluated regarding the predefined criteria representing the systems' suitability for IoT use cases.

Communication. The research findings on the suitability of specific database systems for storing and processing IoT data are provided.

4 MARKET AND REQUIREMENT ANALYSIS

4.1 Market Analysis

For the initial selection of databases, we performed a market analysis. Since there exists a plethora of database systems, a collection of the information on all database systems seems impossible. Still, we aimed for a comprehensive overview of available systems and collected information on 50 systems in total. At this stage of our research, we first tried to gather all popular database systems without applying exclusion criteria. For each selected database system we collected information on its name, type (relational, NoSQL, or time-series), the maintainer, license, as well as stable release, latest version, and first release. Additionally, we state the operating systems that the system can be installed on (i.e., Windows, Linux, and MacOS). Since our research focuses on on-premise database systems, we also check the business model, i.e., if the systems can be installed on-premise or if they are only hosted as a cloud service. To be able to properly compare non-functional requirements of the database systems, we decided to use Docker[1] to ensure a comparable setup. Although a native installation might be slightly faster compared to using Docker, trough the lack of virtualization, we used Docker to ensure that the technical conditions are uniform across all experiments. This allows us to assess the performance of the analyzed database systems. We selected Docker as it is one of the state-of-the-art container-based technologies, providing an efficient and scalable foundation for distributed applications. As such it is widely applied for IoT applications in practice (Alam et al., 2018). Hence, the last criterion included in the market analysis is the availability of a Docker image. We only considered Docker images that are marked as official on Docker hub or are provided by a verified publisher. To limit the scope of our research, we solely focus on relational, NoSQL, and time-series databases and created a comprehensive long list.

We collected information on 15 relational, 15 NoSQL, and 20 time-series databases and ordered them by their popularity. We applied a popularity ranking published and maintained by solid IT gmbh (2022), who define a popularity score for roughly 400 different database systems. The underlying method is based on six parameters, which are the number of mentions on websites based on the query results from Google and Bing, general interest as measured by search frequency based on Google Trends, frequency of online discussions about the system based on primarily IT-related forums such as Stack Overflow, number of job offers that mention the database systems based on the job search engines Indeed and Simply Hired, number of LinkedIn profiles that state the database system as a skill, and the systems' relevance on social media as represented by Twitter tweets mentioning the system.

The resulting long list of database systems is de-

[1] https://www.docker.com/

picted in appendix A.1. Next, we apply the collected information to create a short list of six database systems that are then used for our detailed comparison. For our research, we make the initial restrictions, that we only consider two systems per category and additionally, only consider open-source and on-premise databases that can be installed via Docker on an Ubuntu[2] system. We considered the criteria in the following order: business model (on-premise or cloud solution), license, availability of Docker image, further criteria, and popularity.

From the set of relational databases, five were excluded because they are only hosted as a cloud service (Microsoft Azure, Snowflake, Databricks, Google Big Query, and Amazon Redshift). Six further databases could not be considered because they are not open-source systems (Oracle, Microsoft SQL Server, IBMDB2, Microsoft Access, Teradata, and FileMaker). From the remaining four, one system (SQLite) does not provide an official Docker image, thus the set of databases was reduced to the systems MySQL, PostrgreSQL, and Maria DB. Since we only wanted to select two systems from each category, we excluded Maria DB in the final step because it is less popular.

For the set of NoSQL database systems, we were able to exclude seven systems because they are cloud-only, and thus do not meet the scope of our research (Amazon Dynamo DB, Microsoft Azure Cosmos DB, Firebase Realtime Database, CouchBase, Google Cloud FireStore, Microsoft Azure Table Storage, and Google Cloud Bigtable). One of the remaining eight systems could also not be considered due to a commercial license (Datastax Enterprise). Based on the popularity, we selected MongoDB and Cassandra as final candidates for our requirement analysis. Not chosen because of their lower popularity were Redis, Hbase, Memcached, CouchDB, and Neo4j. Besides, Redis and Memcached aim at different targets, i.e., they are primarily designed for key-value storage, which does not fit our research goal.

From the set of 20 time-series databases, 3 were excluded because they cannot be installed on-premise (Fauna, TD Engine, and Amazon Timestream). An additional set of 5 database systems only have a commercial license, and thus could not be considered here (Kdb+, Dolphin DB, eXtremeDB, IBM Db2 Event Store, and Raima Database Manager). In the next step, we identified that 9 systems did not provide an official Docker image (Graphite, Timescale DB, Apache Druid, RRD Tool, Open TSDB, Grid DB, KairosDB, Victoria Metrics, and Apache IOTDB). The final set of tables consisted of InfluxDB, Prometheus, and QuestDB, but since Prometheus is rather used for monitoring purposes, we excluded it here, so we selected InfluxDB and QuestDB as time-series databases for our requirements analysis. InfluxDB is already a very well-established database in the field of IoT, while QuestDB is still fairly new since it was first released in 2016.

[2]https://ubuntu.com/

4.2 Requirement Analysis

For the comparison of our final set of databases, we identified functional and non-functional requirements that are important criteria to consider when selecting a suitable database for a given context. As functional requirements, we consider the following criteria proposed by Bader et al. (2017): (i) distribution or clusterability, (ii) availability of (a) basic and aggregate and (b) advanced functions, (iii) granularity and downsampling, (iv) interfaces and extensibility and (v) support and license.

For the distribution and clusterability criteria, we consider the CAP theorem, scalability, and load balancing. In the IoT, consistency is the least important characteristic from the CAP theorem since eventual consistency is sufficient as data is collected continuously. The focus is therefore on high availability and partition tolerance. Due to the large amounts of data, storage and performance should be scalable to meet the criteria of an IoT application. For further efficiency, load balancing should be included to evenly distribute the workload in a distributed system (Bader et al., 2017).

As basic functions we consider inserting, reading, scanning, updating, and deleting the data, while averaging, summing, counting, and identifying the maximal or minimal value are the aggregate functions we tested. Further, to compare advanced functions we compare continuous calculation, tags, long-term storage, and matrix time series. Matrix time series describes that multiple timestamps can be entered per item (Bader et al., 2017).

In order to compare granularity and down-sampling, four criteria are considered: support of down-sampling, the smallest sample interval, the smallest granularity for storage, and the smallest guaranteed granularity for storage. Most databases support down-sampling, i.e., when executing queries the results can be fitted for outputs with longer time frames. Two timestamps must be defined, as well as data for short time intervals within the wider time intervals. The shorter time interval is also referred to as the sampling interval. Granularity describes the smallest gap feasible between two timestamps, i.e., the degree of detail in data that can be stored. If

a system stores data in finer granularity than is securely guaranteed, data could be aggregated or might get lost. For our comparison, we consider the smallest sample interval, the smallest granularity for storage, and the smallest guaranteed granularity for storage (Bader et al., 2017).

In the next category, interfaces and extensibility, we consider collecting consisting interfaces and checking for client libraries and plugins. As interfaces, we consider both, graphical and non-graphical interfaces that are provided for a specific database system. Client libraries support handling network management and accessing the data. The functionality of the database system may be increased with plugins (Bader et al., 2017).

To study support and license, we check for long-term support (LTS) of stable versions, commercial support, and free licenses (Bader et al., 2017).

Next to functional requirements, we also consider non-functional requirements for our comparison. First, we study the databases' efficiencies when loading data. We load the data in batches and increase the batch size in powers of 10. The resulting metric represents the average number of data instances loaded per unit of time (Hao et al., 2021).

Second, the query latency, i.e., the amount of time needed to execute a query, is tested. We rely on the research by Liu and Yuan (2019) as we apply their defined query types. The resulting metric per query is the average response time. We slightly adapted the ten queries presented by Liu and Yuan (2019) to better fit our research scope and data structure, since Liu and Yuan (2019) solely focus on time-series databases.

5 COMPARISON OF DATABASES

5.1 Data Collection and Installation

To adequately compare the functionalities of the database systems, we rely on large datasets. For our research, we utilize three different data sources. First, the majority of the data is fetched from *The Things Network (TTN)*[3], collecting data that is generated by end devices installed and configured by the research group. Using a workflow created in *Node-RED*[4] we track every new data point from the collection of devices from TTN and store the data in a MySQL database, which is only used to easily store the data during the data collection. Second, we use further data that is generated by weather sensors installed at the University of Koblenz. This data can be accessed by an Application Programming Interface (API) endpoint and is periodically saved in the MySQL database as well. Last, we artificially create data from the two previous data sources to further extend the size of the dataset. In total, we were able to collect and generate around 10 million data points.

The data consists of five attributes: a unique identifier for each data point, a timestamp, an identifier referencing the sensor that generated the data, the type of measurement, and the measured value.

In order to test the database systems, we install the corresponding Docker image of each system on an Ubuntu machine. Using a unified setup for each database system ensures a fair comparison of the non-functional requirements of the database systems. For each database system, we used the latest version of the Docker image that was available at the time the paper was written. Our evaluation is written in Python, using Python client libraries for each database system.

5.2 Comparison

5.2.1 Functional Requirements

The results of the comparison of functional requirements can be found in appendix A.2. First, we studied three criteria related to distributed systems. We were able to conclude that both relational databases favored consistency and availability over partition tolerance. The NoSQL database Cassandra, however, favors partition tolerance over consistency, while MongoDB guarantees consistency over availability. Similar to Cassandra, InfluxDB focuses on availability over consistency, while QuestDB favors consistency over availability. Both relational databases do not support scalability or load balancing. Additionally, compared to other time-series databases, the open-source version of InfluxDB is not scalable.

All databases support the basic and aggregate functions that were checked in our research. Concluding, since all the databases provide aggregate functions, they also fulfill the continuous calculation criteria. Only InfluxDB supports tags, and only the relational databases do not support long-term storage. As time-series databases use the timestamp as the primary key, they do not allow for matrix time series, while the other database systems do.

All databases support down-sampling without restrictions. Additionally, all support a minimum time range of one millisecond.

Each database provides an API to access the data. In addition, they all provide client libraries and plugins to connect to the database via multiple program-

[3]https://www.thethingsnetwork.org/

[4]https://nodered.org

Table 1: Comparison of Data Loading (in sec.).

Database	10^0	10^1	10^2	10^3	10^4	10^5	10^6	10^7
MySQL	0.311	2.359	24.570	253.74	2476.4	-	-	-
PostgreSQL	0.102	**0.083**	**0.158**	0.859	15.436	37.727	276.273	3086.24
Cassandra	**0.034**	0.626	3.131	29.807	247.68	2421.68	-	-
MongoDB	1.110	1.123	4.479	40.033	450.10	3844.97	24127.58	-
InfluxDB	1.714	1.479	12.222	119.83	1263.70	12273.91	-	-
QuestDB	0.251	0.190	0.285	**0.315**	**0.316**	**2.0347**	**19.161**	**272.30**

ming languages. Most of them also support a free open-source interface for querying and visualizing data. Detailed information on the provided APIs can be found in appendix A.2.

For the last subgroup of functional requirements, we collected information on the current stable version, if they have commercial support, and the license. We provide that information to give an understanding of how active the development of all the selected databases is.

5.2.2 Non-functional Requirements

The test of the non-functional requirements is divided into two parts. First, we measured the data ingestion time using the data that was collected from the different sensor sources. This measurement should represent the behavior of the databases when faced with inserting large amounts of IoT data. From the results in table 1, we can conclude that QuestDB is by far the fastest database when inserting data. During the test, we noticed that some databases crash when the amount of data points inserted is too large. Only QuestDB and PostgreSQL were able to cope with the entire set of ten million data instances. MySQL was the slowest database system and additionally also crashed first after inserting ten thousand data points. To our surprise, InfluxDB showed a low performance.

Second, we studied the query latency using ten thousand data points from the dataset. We needed to reduce the entire dataset to ten thousand because MySQL was not able to cope with more data. Thus, to ensure a fair comparison, we reduced the dataset. The results are depicted in table 2. In total, we executed 10 queries (Q1 - Q10) according to Liu and Yuan (2019):

Q1: Exact Point Query. A particular data instance was fetched from the databases based on specified matching criteria. The relational databases performed well, which can be explained by their indexed primary key. Overall, Cassandra showed the best performance followed by PostgreSQL and QuestDB.

Q2: Time Range Query. For this query, a time range for the data that needs to be fetched was specified. Similarly, as for the first query, Cassandra showed the best performance followed by QuestDB. Here MongoDB showed the worst performance, i.e., being the slowest database the fetch the data.

Q3: Time Range Query with Limit. The third query is a modification of the second query since a limit was added, reducing the fetched data to a pre-defined amount. Here Cassandra and PostgreSQL showed the best results.

Q4: Time Range Query with Multiple Filters. For this query, we included multiple filter criteria to the query. Cassandra was again the fastest database to execute the query, while MongoDB showed the slowest performance.

Q5: Time Range Query with Multiple Filters with Limit. As an adaptation to the fourth query, we added a limit to restrict the number of results. Cassandra again showed the best performance, followed by the two relational database systems.

Q6: Query with Aggregation Function (AVG). From this query onward we test several aggregation functions. Starting with the average function, our results show that QuestDB executed the query first. The two relational databases also show decent results.

Q7: Query with Aggregation Function (COUNT). When counting data points, PostgreSQL showed the best performance. Due to an error, the results of the Cassandra database could not be considered. The Python client was not able to execute the query, since the aggregation of data was grouped by the devices, which is not the primary key. Cassandra, however, only permits an aggregation grouped by the primary key, which in our case is the timestamp.

Q8: Query with Aggregation Function (SUM/ MEAN). For the SUM or MEAN function, MongoDB showed the best performance. Similarly to the previous query, the result of the Cassandra database could not be considered due to an error when executing the query.

Q9: Query with Aggregation Function (MAX/ MIN). QuestDB, MySQL, and InfluxDB showed

Table 2: Comparison of Query Latency (in sec.).

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
MySQL	0.095	0.102	0.836	0.091	0.028	0.083	0.107	0.186	0.067	0.281
PostgreSQL	0.031	0.162	0.016	0.12	0.018	0.136	**0.036**	0.117	0.147	0.218
Cassandra	**0.0004**	**0.0009**	**0.001**	**0.0005**	**0.001**	0.485	-	-	0.337	-
MongoDB	0.851	0.3	0.041	0.334	0.313	0.166	0.074	**0.116**	0.103	0.236
InfluxDB	0.203	0.197	0.131	0.071	0.087	0.213	0.213	1.584	0.087	0.213
QuestDB	0.037	0.037	0.045	0.058	0.044	**0.0419**	0.121	0.135	**0.055**	**0.141**

the best results when applying the MAX or MIN aggregation function. The database Cassandra showed rather weak performance.

Q10: Query with Aggregation Function (MAX) and Group-By Clause. Due to the same execution error as before, the result of the Cassandra database could not be considered. QuestDB showed the best performance among the remaining database systems.

Despite the reoccurring error, the NoSQL database Cassandra shows the best performance for five queries as compared to the other database systems. When calculating the average, however, QuestDB shows the best results, while InfluxDB comes in last due to a significantly higher value for Q8 as compared to results from other queries. On average, PostgreSQL and MySQL also show decent results overall.

6 DISCUSSION AND CONCLUSION

In this research, we demonstrated a comparison of database systems regarding their functionalities required to meet the demands for IoT applications. We initially performed a market analysis to identify a large set of databases. For our market study, we focused on relational, NoSQL, and time-series databases. After having collected information on 50 database systems, we refined the set based on predefined criteria. We solely focused on open-source, on-premise database systems that can be set up using an official Docker image. After excluding systems that did not meet our criteria, we applied a popularity measure to identify the two most popular systems per category from the systems that were still remaining.

Next, we performed a requirements analysis based on related literature to identify functional and non-functional requirements for a database system in an IoT context. Based on these requirements we performed a variety of tests. As a result, we were able to demonstrate the weaknesses and strengths of the database systems, which helps practitioners and researchers in their decision on which database to use.

From our comparison of the functional requirements, we conclude the following aspects. The selected relational database systems focus on consistency and availability, while both, the NoSQL and time-series databases definitely focus on partition tolerance, but then either aim for high availability or high consistency. Only the relational database systems are not scalable and do not provide load balancing. Additionally, the open-source version of InfluxDB is also not scalable. Since scalability is an important aspect to handle the vast amounts of data created by IoT devices, our results regarding this criterion show that relational databases are less suitable. Additionally, the relational databases also do not support long-term storage, which again makes them less appropriate for IoT use cases. Tags are only supported by InfluxDB, while matrix time series is only possible in the relational and NoSQL databases. For the remaining functional criteria, we were not able to observe significant differences between the systems.

Our test of the non-functional requirements demonstrated inclusive results. First, the results of the data ingestion test showed that QuestDB, one of the time-series databases tested, shows the best results overall, while InfluxDB performed very poorly. Based on this observation, we cannot make the generalized conclusion that time series databases are best suited for storing large amounts of data quickly. Interestingly, one of the relational databases, PostgreSQL, showed very decent results for each test as well. Both NoSQL databases performed moderately.

Second, our comparison of the query latencies demonstrated that Cassandra shows by far the best results. Additionally, the relational databases show decent performance for basic queries since they used a primary key on the timestamp improving the performance of queries on the timestamp. On average, QuestDB performs the best. Similar to our test on loading the data into the databases, InfluxDB shows inefficient performance. As expected, the relational databases performed only decently.

While based on the results of our test on functional and non-functional requirements, we are not able to identify a clear winner, we still conclude that QuestDB appears to be most suitable to handle the

large amounts of data generated from IoT devices. We additionally observed that there exist large differences between databases of the same type since InfluxDB indicated a weaker performance than QuestDB, for instance.

Our study has limitations since we restricted the set of tested systems to open-source and on-premise databases, while commercial and/or systems hosted on clouds make up a large majority of the database systems available as demonstrated in our market research. We did not consider further requirements for using database systems in an IoT application, such as security. Still, we demonstrated the strengths and weaknesses of six popular databases regarding their suitability for IoT use cases, providing a comprehensive overview to support decision-making.

ACKNOWLEDGEMENTS

This research has been supported by the Deutsche Forschungsgemeinschaft (DFG) under Research Grant No. 432399058.

REFERENCES

Alam, M., Rufino, J., Ferreira, J., Ahmed, S. H., Shah, N., & Chen, Y. (2018). Orchestration of microservices for iot using docker and edge computing. *IEEE Commun*. Mag., 56(9), 118–123.

Amghar, S., Cherdal, S., & Mouline, S. (2018). Which NoSQL database for IoT Applications? 2018 *Proc. of MoWNeT*, 131–137.

Bader, A., Kopp, O., & Falkenthal, M. (2017). Survey and comparison of open source time series databases. *Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband*, 249–268.

Baiyere, A., Topi, H., Wyatt, J., Venkatesh, V., & Donnellan, B. (2020). Internet of things (IoT) – a research agenda for information systems. *Commun. Assoc. Inf. Syst.*, 47(1), 21.

Beynon-Davies, P. (2017). *Database systems*. Bloomsbury Publishing.

Chauhan, D., & Bansal, K. L. (2017). Using the advantages of nosql: A case study on mongodb. *Int. J. Recent Innov. Trends Comput. Commun.*, 5, 90–93.

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mob. Netw. Appl.*, 19, 171–209.

Chui, M., Löffler, M., & Roberts, R. (2020). The Internet of Things. *McKinsey Quarterly*.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13, 377–387.

De Mauro, A., Greco, M., & Grimaldi, M. (2015). What is big data? A consensual definition and a review of key research topics. *AIP Conference Proceedings, 1644*, 97–104.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems (7th ed.)*. Pearson.

Fatima, H., & Wasnik, K. (2016). Comparison of sql, nosql and newsql databases for internet of things. 2016 *IEEE Bombay Section Symposium (IBSS)*, 1–6.

Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. *MIS Quarterly*, 37, 337–355.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7), 1645–1660.

Hao, Y., Qin, X., Chen, Y., Li, Y., Sun, X., Tao, Y., Zhang, X., & Du, X. (2021). Ts-benchmark: A benchmark for time series databases. *Proc. of 37th IEEE ICDE*, 588–599.

ITU-T, T. S. S. o. I. (2011). *Overview of the Internet of Things* (tech. rep.) [Publication Title: International Telecommunication Union].

Liu, R., & Yuan, J. (2019). Benchmarking time series databases with iotdb-benchmark for iot scenarios. *arXiv preprint*. http://arxiv.org/abs/1901.08304

Lycett, M. (2013). 'Datafication': Making sense of (big) data in a complex world. *Eur. J. Inf. Syst.*, 22(4), 381–386.

Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7), 1497–1516.

Mostafa, J., Wehbi, S., Chilingaryan, S., & Kopmann, A. (2022). Scits: A benchmark for time-series databases in scientific experiments and industrial internet of things. *Proc. of the 34th Int. Conf. Sci. Stat. Database Manag.*, 1–11.

Musa, E., Delač, G., Šilić, M., & Vladimir, K. (2019). Comparison of relational and time-series databases for real-time massive datasets. *2019 Proc. of the 42th MIPRO*, 1065–1070.

Nasar, M., & Kausar, M. A. (2019). Suitability of influxdb database for iot applications. *Int. J. Eng. Innov. Technol.*, 8, 1850–1857.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *J. Manag. Inf. Syst.*, 24(3), 45–77.

Rautmare, S., & Bhalerao, D. M. (2016). MySQL and NoSQL database comparison for IoT application. 2016 Proc. of. IEEE ICACA, 235–238.

Rayes, A., & Salam, S. (2017). *Internet of things from hype to reality*. Springer Basel.

Sestino, A., Prete, M. I., Piper, L., & Guido, G. (2020). Internet of Things and Big Data as enablers for business digitalization strategies. *Technovation*, 98(July).

solid IT gmbh. (2022). Db-engines ranking. Retrieved December 7, 2022, from https://db-engines.com/en/ranking/relational+dbms

Strohbach, M., Ziekow, H., Gazis, V., & Akiva, N. (2015). Towards a big data analytics framework for iot and smart city applications. MOST, 4, 257–282.

A APPENDIX

A.1 Market Analysis

Database	Type	Operating Systems	Maintainer	License	Latest Stable Release	First Release	Cloud/ on-prem.	Official Docker Image
Oracle	relational, multi-model	all	Oracle Corp	commercial	in 2019: 19C	1980	both	no
MySQL	relational, multi-model	all	MySQL	open-source	in 2022: 8.0.29	1995	both	yes
Microsoft SQL Server	relational, multi-model	Linux, Windows	Microsoft	commercial	in 2019: SQL Server 2019	1989	both	no
PostgreSQL	relational, multi-model	all	PostgreSQL Global Development Group	open-source	in 2022: 15.1	1996	both	yes
IBMDB2	relational, multi-model	all	IBM	commercial	in 2022: 13.1	1983	both	yes
Microsoft Access	relational	Windows	Microsoft	commercial with trailware	in 2019: Microsoft Access 2019	1992	both	no
SQLite	relational	all	SQLite	public domain	in 2022: 3.39.0	2000	both	no
Maria DB	relational, multi-model	all	MariaDB Corporation Ab, MariaDB Foundation	open-source	in 2022: 10.9.3	2009	both	yes
Microsoft Azure	relational, multi-model	Windows	Microsoft	commercial	in 2022: V12	2010	cloud	no
Snowflake	relational	hosting on all	Snowflake Computing Inc.	commercial	in 2022: 6.15	2014	cloud	no
Databricks	multi-model	hosting on all	Databricks	open-source	in 2022: 11.2	2013	cloud	no
Google Big Query	relational	hosting on all	Google	commercial	in 2022: 3.3.3	2010	cloud	no
Teradata	relational	Linux	Teradata	commercial	in 2019: 1.0 MU2	1984	both	no
Amazon Redshift	relational, multi-model	hosting on all	Amazon	commercial	in 2022: 1.0.41465	2012	cloud	no
FileMaker	relational	all	FileMaker	commercial	in 2021: 19.4.1	1983	both	no
Mongo DB	Document Store	all	MongoDB, Inc	open-source Server-side public license	in 2022: 6.0.1	2009	both	yes
Cassandra	Wide-column store	all	Apache Software Foundation	open-source	in 2022: 4.0.6	2008	both	yes
Redis	Key-value store	all	Redis	open-source	in 2022: 7.0.4	2009	both	yes
Hbase	Wide-column store	all	Apache Software Foundation	open-source	in 2022: 2.5.0	2008	both	yes
Amazon Dynamo DB	Document store, Key-value store	all	Amazon	commercial	in 2022 (no version)	2012	cloud	no
Microsoft Azure Cosmos DB	Graph DBMS, Document Key-value, Wide-column	all	Microsoft	commercial	in 2022 (no version)	2014	cloud	no
Memcached	Key-value store	all	Danga Interactive	open-source	in 2022: 1.6.15	2003	both	yes
Firebase Realtime Database	Document store	all	Google	commercial	in 2022 (no version)	2012	cloud	no
Datastax Enterprise	Wide-column store	all	DataStax	commercial	in 2020: 6.8	2011	both	no
CouchDB	Document Store	all	Apache Software Foundation	open-source	in 2022: 3.2.2	2005	both	yes
CouchBase	Document Store	all	CouchBase, Inc	open-source	in 2021: 7.1.0	2010	cloud	yes
Google Cloud FireStore	Document store	all	Google	commercial	in 2022 (no version)	2017	cloud	no
Microsoft Azure Table Storage	Wide-column store	all	Microsoft	commercial	in 2022 (no version)	2012	cloud	no
Google Cloud Bigtable	Key-value, Wide-column store	all	Google	commercial	in 2022 (no version)	2015	cloud	no

Neo4j	Grapf DBMS	all	Neo4j, Inc	open-source	in 2022: 4.4.11	2007	both	yes
InfluxDB	Time-series	all	InfluxData	open-source	in 2022: 2.4	2013	both	yes
Kdb+	Time-series	all	Kx Systems	commercial	in 2018: 3.6	2000	both	no
Prometheus	Time-series	all	Prometheus	open-source	in 2022: V2.3.0	2015	both	yes
Graphite	Time-series	Linux	Orbitz Worldwide, Inc	open-source	in 2022: 1.1.10	2006	both	no
Timescale DB	Time-series	all	Timescale Inc	open-source	in 2022: 2.6.0	2018	both	no
Apache Druid	Time-series	all	Apache Software Foundation	open-source	in 2022: 24.0.0	2012	both	no
RRD Tool	Time-series	Linux	Tobias Oetiker	open-source	in 2022: 1.8.0	1999	both	no
Open TSDB	Time-series	all	Yahoo	open-source	in 2021: 2.4.1	2011	both	no
Dolphin DB	Time-series	Linux, Windows	Dolphin Db, Inc	commercial	in 2022: V2.00.4	2018	both	no
Fauna	Time-series	all	Fauna, Inc	commercial	in 2021 (no version)	2014	cloud	no
Quest DB	Time-series	all	QuestDB Limited	open-source	in 2021: 6.0.3	2016	both	yes
Grid DB	Time-series	Linux	Toshiba Digital Solutions Corp.	open-source	in 2022: 5.0.0	2013	both	no
TD Engine	Time-series	Linux, Windows	TDEngine	open-source	in 2022: 3.0.1.3	2019	cloud	no
Amazon Timestream	Time-series	all	Amazon	commercial	in 2022 (no version)	2022	cloud	no
KairosDB	Time-series	all	KairosDB	open-source	in 2018: 1.2.2	2013	both	no
eXtremeDB	Time-series	all	McObject	commercial	in 2021: 8.2	2001	both	no
Victoria Metrics	Time-series	all	VictoriaMetrics	open-source	in 2022: V1.77	2018	both	no
IBM Db2 Event Store	Time-series	all	IBM	commercial	in 2021: 2	2017	both	no
Raima Database Manager	Time-series	all	Raima Inc	commercial	in 2021: 15.2	1984	both	no
Apache IOTDB	Time-series	all	Apache Software Foundation	open-source	in 2022: 0.13.2	2018	both	no

A.2 Comparison

			MySQL	**PostgreSQL**	**Cassandra**	**MongoDB**	**InfluxDB**	**QuestDB**
i)	CAP Theorem		CA	CA	AP	CP	AP	CP
	Scalability		✗	✗	✓	✓	✗	✓
	Load Balancing		✗	✗	✓	✓	✓	✓
ii)	a)	Basic Functions	✓	✓	✓	✓	✓	✓
		Aggregate Functions	✓	✓	✓	✓	✓	✓
	b)	Continuous Calculation	✓	✓	✓	✓	✓	✓
		Tags	✗	✗	✗	✗	✓	✗
		Long-term Storage	✗	✗	✓	✓	✓	✓
		Matrix Time Series	✓	✓	✓	✓	✗	✗
iii)	Down-Sampling		✓	✓	✓	✓	✓	✓
	Smallest Sample Interval		1ms	1ms	1ms	1ms	1ms	1ms
	Smallest Granularity for Storage		1ms	1ms	1ms	1ms	1ms	1ms
	Smallest Guaranteed Granularity for Storage		1ms	1ms	1ms	1ms	1ms	1ms
iv)	API and Interfaces		MySQL Rest API, MySQL Workbench and CLI	PostgREST for API, PgAdmin and CLI	Cassandra Restful API, Cassandra Query Language Shell (cqlsh)	MongoDB REST API, MongoDB Compass	Collect, CLI, Graphite, InfluxQL, OpenTSDB	QuestDB REST API, Web Console
	Client Libraries		✓	✓	✓	✓	✓	✓
	Plugins		✓	✓	✓	✓	✓	✓
v)	Version Used in Experiments and Release Date		8.0.30 Jul 2022	15.0 Oct 2022	4.0 Oct 2022	6.0 Aug 2022	2.4.0 Aug 2022	6.5.4 Oct 2022
	Commercial Support		✓	✓	✓	✓	✓	✓
	License		GNU GPL	PostgreSQL License	Apache Software Foundation	GNU AGPL	MIT	Apache-2.0