A Comparison of SQL, NoSQL and Distributed SQL Database Performance within the Internet of Things

Jack Fletcher
Department of Computer Science
Staffordshire University
Stoke-on-Trent, United Kingdom
f021280l@student.staffs.ac.uk

***Abstract*— When considering potential data storage solutions for business developments, SQL has been the standard database type considered since the late 1970s, storing data using a structured tabular format. However, with the rise of the Internet of Things (IoT), vast quantities of unstructured sensor data have been collected as part of the Big Data movement, becoming difficult to work with SQL's rigid schema. Database paradigms including the CAP Theorem, BASE (Basically available, Soft State and Eventual Consistency) and ACID properties (Atomicity, Consistency, Isolation and Durability) are explored in relation to database approaches. Several promising alternatives to SQL have arisen over the years which have their performance compared within this paper using a desk-based research methodology.**

Keywords—NoSQL,SQL,Relational Databases,MySQL,NewSQL,Distributed Systems,Distributed SQL

I. Introduction

The Internet of Things (IoT) are physical devices that contain a multitude of sensors that can for example, sense changes in their environment, e.g., a temperature sensor. For each event these devices go through, data points can be created and sent back to the controlling enterprise to support research and development [1]. With Cisco predicting that IoT home applications will grow by 48% and car applications by 30% by 2023[2], an easily scalable database solution should be selected. Additionally, due to constant datapoints from time sensitive data, high concurrent read and write speeds should be employed. [3].

This serves as the rationale behind this paper and its focus on performance. It's important to assess current database approaches to ensure current and future requirements can be met, in addition to allowing for business growth.

While relational databases have been the standard since the 1970s using SQL or Structured Query Language, their compliance to the ACID principles has resulted in a focus on consistency, not latency, a core requirement of IoT devices. Relational Database Management Systems (RDBMS) also have a strict schema which is unsuitable for the IoT due to its varied data. [4]. Additionally, SQL databases scale better vertically, by upgrading the server hardware currently in use. This has resulted in growth of alternative approaches such as NoSQL using Cassandra and MongoDB.

NoSQL or Not-Only-SQL is a flexible and schema free database that does not belong to the relational database model and conforms to BASE principles. This flexibility may be useful for the IoT due to the large quantities of likely varied data. [5]. It has been classified several times, with some overlap. The key categories include:

Type	Examples
Document store	Apache CouchDB,MongoDB
Key Value Pair	Amazon DynamoDB
Graph Store	Neo4J,OrientDB,RedisGraph
Wide Column	Apache Cassandra,Scylla,Apache HBase

Table 1. Types of NoSQL Databases

II. Backgrounds and Related Works

A. Cap Theorem, ACID and BASE

The transaction paradigm, defined by Reuter and Härder [6] builds upon work by Gray and describes the major highlights of the transaction paradigm. This aims to guarantee data validity using the concept of transactions, which can be considered a single unit of work; all actions affecting the database contained within a single transaction must occur or no changes are committed to the database. To achieve this; they defined four core traits known as the ACID properties.

- Atomicity, by considering all related actions within a transaction a single unit, with either complete success or the database is not changed, it provides safety against only partial changes
- Consistency guarantees that by only allowing successful transactions to be committed to the database, only valid results can affect the database
- Isolation, or the idea that concurrent transactions are not aware of each other.
- Durability guarantees that transactions that have been committed, stay committed.

The trade-off for maintaining consistency is latency. To maintain ACID compliance, servers lock the objects affected by a transaction until the transaction is completed. In large scale databases where there may be millions of concurrent transactions, this can cause noticeable latency. This can be acceptable for applications that require reliable data, such as accounting software, however for IoE applications such as a temperature monitoring system, where consistency is not highly sought after, an alternative database that focuses on availability may be suitable.

The CAP Theorem, or Brewers Theorem [7], dictates that distributed databases can only provide two of three characteristics – *Consistency*, *Availability* and *Partition Tolerance*, forming CAP.

- Consistency, being slightly different from the ACID characteristic, requires that any read request to any specific node, shows the same data at any one time. In other words, this shows the most up to date result at any one time. This requires each node forwards and write request to all further nodes to ensure parity between each node
- Availability requires that any request receives a response, regardless of whether all servers are online. This means that should a node go down; the request is forwarded to an available node
- Partition tolerance implies that even if there is a break in communication between nodes, the cluster of nodes would continue to function; either sending old data (In a CP database) or not responding to requests (in an AP database)

This would theoretically, give three combinations, denoted by the initials of each characteristic – CA, CP, or AP. Unfortunately, however, CA distributed databases are not able to be practically created, as distributed computing will always have the potential for network partitions to be created, as it is impossible to guarantee no messages are dropped and the nodes do not fail. [8] Arguably, if a CA database is desired, it would be more applicable to use a traditional relational database management system (RDMS) such as PostgreSQL instead of a distributed system. Regarding this system, further databases will be described using their acronym.

In practice, the CAP theorem is taken as more of a guideline, with the sacrificed characteristic typically being compensated for using either physical hardware or software solutions. Databases that require a weaker consistency favouring the BASE model [9]. The inverse of the ACID model, it is comprised of *Basically Available*, *Soft-State* and *Eventual Consistency*.

- Basically available guarantees a response to a request, however that response could be a failure
- Soft-State suggests that the state of the system may change over time, even without user input. This can be due to the eventual consistency guarantee
- Eventual Consistency guarantees that when an update is made to a node, for example a write query, the update will eventually be updated for all nodes in the cluster, however they may not all be consistent at any given time

Returning to the idea of temperature management, a BASE database, most likely using the AP classification, would be more applicable due to its reliance on availability over instantaneous consistency.

B. *Non-Relattional Database Management Systems*

1) NoSQL

Due to NoSQL's various classifications, two popular database management systems for IoT (DBMS) from different classifications were selected for comparison, MongoDB, and Cassandra.

a) MongoDB

MongoDB [10] is a CP based open-source document database that aims to provide high performance databases with flexible schemas. It considers each set of data a document, which can be part of a collection, e.g., John Doe's data may be part of a collection called Users. These documents use JSON, a standard storage format that has good integration with many programming languages. Figure 2 shows a typical JSON object.

Fig. 1 An Example of JSON Data

b) Cassandra

Cassandra [11] is an AP based wide-column store, similar in design to a traditional RDBMS, developed by Facebook in 2008. Typical of all NoSQL solutions, it has an easily distributable architecture and supports flexible schemas. It provides no single point of failure, with every node containing identical data. Using a custom query language known as Cassandra Query Language (CQL) with similar syntax to SQL, it provides an easier learning curve and experienced SQL developers should be able to migrate easily. [12]

C. *Relational Database Management Systems*

1) MySQL

MySQL [13] is one of the most popular database management solutions of 2021[14] and offers an open-source solution to database needs. As a relational database system, it stores data in a tabular format and uses the relational model [15] to link tables together.

2) Distributed SQL and NewSQL

With the arrival of the Big Data Movement and IoT, databases were required to scale out dramatically to deal with the increased data intake. For traditional RDBMS, this required upgrading the server's hardware for increased performance. This becomes expensive quickly,

with higher end components reaching diminishing returns of performance. Alternatively, a NoSQL based solution could be implemented, however this would likely mean giving up ACID properties. [16] As a result of this, NewSQL and Distributed SQL was developed to benefit from distributed computing and the ability to scale horizontally, while keeping ACID properties. [17]

Distributed SQL is often considered a subset of NewSQL with several DBMS providers using the terms interchangeably. For this purpose, we will consider them separate using IBM's definitions. [18].

- NewSQL: Converts or adds distributed logic to existing SQL databases
- Distributed SQL: Builds the distributed database from the ground up

a) CockroachDB

CockroachDB [19] is a CP based open source distributed SQL system inspired by Google's Spanner [20]. It loosely supports ACID properties and allows for clustering to scale horizontally, supporting deployment across both on premises and on the cloud, known as hybrid deployment. CockroachDB stores user data in a map of key-value pairs which is then mapped to a chunk known as a 'range' which can initially be considered a table in traditional SQL. When this range reaches its max size (default 512 MiB), it splits into two ranges. [21]

III. A Comparison of Database Management Systems

A. Performance Measurements

In [22] using a tabular data structure where data had a unique ID and two attributes MongoDB had consistently fast read and write times compared to MySQL and Cassandra, however using the built-in update function reduced its overall ranking considerably. The author found using the delete and write commands to update values instead was considerably faster, however this may have unintended effects and is potentially not a viable long-term solution. Additionally, it is unknown whether they used data that is indicative of real-world data, therefore while this is a good artificial test, it may need to be reran with different data using a standardised tool such as YCSB [23]. In [24] they found that while Redis performed significantly better than MongoDB in read and write operations, MongoDB still performed significantly faster than Cassandra due to its use of volatile memory. This supports the previous paper by Reichardt et al and uses YCSB, a commonly used tool to compare performance in NoSQL database management systems. In comparison Swaminathan & Elmasri found that Cassandra had consistently better throughput than MongoDB, disregarding blind read comparisons. [25] There does not appear to be any clear reason for this conflict, so it's possible that improvements have been made to Cassandra since the original experiment was performed. Conversely in [26] they found that MySQL slowed dramatically when there were more than 7000 read and write operations a second, however for lower throughput and read intensive environments the performance was comparable to Cassandra. This is compounded by work by Keshavarz who found that MySQL write and read duration increased exponentially when testing a higher number of records, however, was roughly twice as performant as MongoDB at update and delete requests. [27]

In [28] they found that CockroachDB had significantly worse throughput and latency than comparable NewSQL solutions when benchmarking using YCSB and suggest this may be due to its use of disk based primary storage, rather than memory-based storage. As CockroachDB utilizes a variation of the ANSI SERIALIZABLE SQL isolation level [29], this may also account for its low performance. [30], [31] If a lower isolation level is desired, it may be more applicable to use a different technology as CockroachDB does not currently offer the ability to change this. Additionally, in [33] it was found that NoSQL databases such as MongoDB and Cassandra are significantly faster at manipulating unstructured data when compared to SQL and NewSQL solutions and Mongo also had typically faster query time than other solutions. This study also tests using JSON data, which is well suited for handling the unstructured data of the Internet of Things and is closely linked to MongoDB's database type.

B. Discussion

The performance metrics shown above show some conflicting information. This may be due to the age of the study, however there was no standardised testing approach used throughout all these studies, although some of them used YCSB. This is most likely due to each of the studies targeting a different type of data within the IoT, which makes it difficult to suggest a database that is clearly better.

IV. Conclusion

Based on the research within this paper, it's clear no single database format can be used definitively within the IoT; however, a NoSQL candidate, such as MongoDB or Cassandra, is likely the preferred choice due to its low cost for initial creation, and ability to scale horizontally using cheaper commodity hardware as nodes. As noted within this paper, improvements in each DBMS can result in previous work being made redundant, therefore it's important for updated research to be done on database performance metrics after any significant update. Within this paper it was found that MongoDB was typically faster than both Cassandra and MySQL in read and write operations, however its update operations were slower than alternative systems. As of MongoDB 4.2 it also supports multi-document transactions and distributed transactions. [32] Additionally, MySQL scored consistently low in high throughput environments yet was comparable to

Cassandra for low throughput scenarios, which would make it a viable solution if low throughput was expected. CockroachDB is likely not currently suitable for the IoT due to its low throughput performance. Extraneous factors must also be considered, e.g., for a company that already has a highly integrated SQL database, converting to NewSQL may be a more viable solution. The chosen connection method, e.g., the Cassandra Python client, may also have a negative impact on performance. IoT sensors have no standardised way of sending data and therefore a flexible schema, such as those found in NoSQL solutions, would be beneficial. A potential solution to this, could be to implement a standardised format for data, such as that described by the Open Data Format. [34] Additionally, this paper focused on throughput performance and not the ease of scaling, which is one of the major features of NoSQL and Distributed SQL systems, a complementing paper evaluating the elasticity of scaling would help develop an informed decision. Additionally, time series databases were not considered in this paper and should be explored further.

REFERENCES

[1] O. Vermesan and P. Friess, Internet of things - from research and innovation to market de-ployment. Aalborg: River Publishers, 2015, p. 297.

[2] E. Perspectives and C. Report, "Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper", Cisco, 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed: 09- Jan- 2022].

[3] P. Thi Anh Mai, J. Nurminen and M. Francesco, 2014 IEEE International Conference on Internet of Things (iThings), and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom), 1st ed. Taipei, Taiwan: IEEE, 2014, pp. 117-124.

[4] B. Diène, J. Rodrigues, O. Diallo, E. Ndoye and V. Korotaev, "Data management tech-niques for Internet of Things", Mechanical Systems and Signal Processing, vol. 138, p. 106564, 2020. Available: https://www.sciencedirect.com/science/article/abs/pii/S088832701930785X. [Accessed 9 Jan-uary 2022].

[5] J. Cooper and A. James, "Challenges for Database Management in the Internet of Things", IETE Technical Review, vol. 26, no. 5, p. 320, 2009. Available: https://www.tandfonline.com/doi/pdf/10.4103/0256-4602.55275?needAccess=true. [Ac-cessed 9 January 2022].

[6] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery", ACM Computing Surveys, vol. 15, no. 4, pp. 287-317, 1983. Available: 10.1145/289.291 [Accessed 9 January 2022].

[7] E. Brewer, "Symposium on Principles of Distributed Computing (PODC)", 2000.

[8] C. Hale, "You Can't Sacrifice Partition Tolerance", codahale.com, 2010. [Online]. Availa-ble: https://codahale.com/you-cant-sacrifice-partition-tolerance/. [Accessed: 09- Jan- 2022].

[9] D. Ganesh Chandra, "BASE analysis of NoSQL database", 2015. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0167739X15001788?via%3Dihub. [Accessed: 09- Jan- 2022].

[10] "MongoDB: the application data platform", MongoDB, 2009. [Online]. Available: https://www.mongodb.com/. [Accessed: 09- Jan- 2022].

[11] "Apache Cassandra | Apache Cassandra Documentation", Apache Cassandra, 2008. [Online]. Available: https://cassandra.apache.org/_/index.html. [Accessed: 09- Jan- 2022].

[12] V. Abramova and J. Bernardino, "NoSQL databases | Proceedings of the International C* Conference on Computer Science and Software Engineering", Doi.org, 2013. [Online]. Available: https://doi.org/10.1145/2494444.2494447. [Accessed: 09- Jan- 2022].

[13] "MySQL", Mysql.com, 1995. [Online]. Available: https://www.mysql.com/. [Accessed: 09- Jan- 2022].

[14] M. Kamaruzzaman, "Top 10 Databases to Use in 2021", Medium, 2021. [Online]. Availa-ble: https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba. [Ac-cessed: 09- Jan- 2022].

[15] E. Codd, "A relational model of data for large shared data banks", Communications of the ACM, vol. 13, no. 6, pp. 377-387, 1970. Available: 10.1145/362384.362685 [Accessed 9 January 2022].

[16] M. Stonebraker, "SQL databases v. NoSQL databases", Communications of the ACM, vol. 53, no. 4, pp. 10-11, 2010. Available: 10.1145/1721654.1721659 [Accessed 9 January 2022].

[17] K. Kaur and M. Sachdeva, "Performance evaluation of NewSQL databases", 2017 Inter-national Conference on Inventive Systems and Control (ICISC), 2017. Available: 10.1109/icisc.2017.8068585 [Accessed 9 January 2022].

[18] B. Anderson, "SQL vs. NoSQL Databases: What's the Difference?", Ibm.com, 2021. [Online]. Available: https://www.ibm.com/cloud/blog/sql-vs-nosql. [Accessed: 09- Jan- 2022].

[19] "Cockroach Labs, the company building CockroachDB", Cockroach Labs, 2015. [Online]. Available: https://www.cockroachlabs.com/. [Accessed: 09- Jan- 2022].

[20] "Frequently Asked Questions | CockroachDB Docs", Cockroachlabs.com, 2022. [Online]. Available: https://www.cockroachlabs.com/docs/stable/frequently-asked-questions.html. [Accessed: 09- Jan- 2022].

[21] "Architecture Overview | CockroachDB Docs", Cockroachlabs.com, 2022. [Online]. Available: https://www.cockroachlabs.com/docs/stable/architecture/overview.html. [Accessed: 09- Jan- 2022].

[22] M. Reichardt, M. Gundall and H. Schotten, "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients", IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society, 2021. Available: 10.1109/iecon48115.2021.9589382 [Accessed 9 January 2022].

[23] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, "Benchmarking cloud serving systems with YCSB", Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10, 2010. Available: 10.1145/1807128.1807152 [Accessed 9 January 2022

[24] N. Seghier and O. Kazar, "Performance Benchmarking and Comparison of NoSQL Da-tabases: Redis vs MongoDB vs Cassandra Using YCSB Tool", 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI), 2021. Available: 10.1109/icrami52622.2021.9585956 [Accessed 9 January 2022].

[25] S. Swaminathan and R. Elmasri, "Quantitative Analysis of Scalable NoSQL Databases", 2016 IEEE International Congress on Big Data (BigData Congress), 2016. Available:

10.1109/bigdatacongress.2016.49 [Accessed 9 January 2022].

[26] B. Tudorica and C. Bucur, "A comparison between several NoSQL databases with com-ments and notes", Doi.org, 2011. [Online]. Available: https://doi.org/10.1109/RoEduNet.2011.5993686. [Accessed: 09- Jan- 2022].

[27] S. Keshavarz, "Analyzing Performance Differences Between MySQL and MongoDB", Researchgate, 2021. [Online]. Available: https://www.researchgate.net/publication/349764039_Analyzing_Performance_Differences_Between_MySQL_and_MongoDB. [Accessed: 09- Jan- 2022].

[28] G. Schreiner, R. Knob, D. Duarte, P. Vilain and R. Mello, "NewSQL Through the Look-ing Glass", Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services, 2019. Available: https://dl.acm.org/doi/pdf/10.1145/3366030.3366080 [Accessed 9 January 2022].

[29] "International Standard ISO/IEC 9075:1992", Contrib.andrew.cmu.edu, 1992. [Online]. Available: https://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt. [Accessed: 09- Jan- 2022].

[30] "CockroachDB Performance | CockroachDB Docs", Cockroachlabs.com, 2022. [Online]. Available: https://www.cockroachlabs.com/docs/stable/performance.html. [Accessed: 09- Jan- 2022].

[31] "Understanding isolation levels - JDBC Driver for SQL Server", Docs.microsoft.com, 2021. [Online]. Available: https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels?view=sql-server-ver15. [Accessed: 09- Jan- 2022].

[32] Docs.mongodb.com, 2022. [Online]. Available: https://docs.mongodb.com/manual/core/write-operations-atomicity/. [Accessed: 09- Jan- 2022].

[33] P. Ribarski, B. Ilijoski, and B. Tojtovska, “Comparing Databases for Inserting and Querying Jsons for Big Data.”

[34] “Open Data Format (O-DF), an Open Group Internet of Things (IoT) Standard – Introduction.” http://www.opengroup.org/iot/odf/p1.htm (accessed Jan. 22, 2022).