Database Systems Performance Evaluation for Cloud Applications

Suhas Shetty, Sapthagiri College of Engineering Bangalore & India, suhasshetty570@gmail.com

Monica S, The Oxford College of Engineering Bangalore & India, preethureddy98@gmail.com

Shiva Ranjani J, The Oxford College of Engineering Bangalore & India, shivaranjani4848@gmail.com

Abstract **As Cloud applications spread throughout smart city appliances, industry, and agriculture, the volume of data saved in Cloud databases grows. Large volumes of actuator and sensory data must be processed in real-time or interactively by modern database systems. Database providers are battling this first wave of the Cloud revolution on a daily basis to expand their market share, create new capabilities, and try to fix the flaws in earlier releases all while offering features for the Cloud:**

Keywords —Cloud, Data, Database, NoSQL, Performance, Storage, SQL.

I. INTRODUCTION

In the 1960s, database systems began to acquire popularity. There have been numerous types created, each with its unique data representation format. originally designed as linked-list-based navigational databases, subsequently evolving into relational databases with joins, triggers, functions, stored procedures, and object-oriented features. NoSQL first appeared in the late 2000s and quickly gained popularity. The relational model, which makes use of SQL as its query language, is the foundation for the majority of today's database implementations. As large volumes of unstructured data being deposited, however, and strict relational databases' performance and scalability limits are exceeded, NoSQL database systems are rising in popularity. This raised the question of whether the relational model had just begun. Relational databases, on the other hand, make use of normalcy forms based on the notion of data divided into field records and tables while adhering to normalization standards. On the other hand, NoSQL databases manage to offer a reliable performance-related answer by escaping from normalcy and redesigning scalable services.

Cloud services' main responsibilities are to collect, filter, analyze, and mine Cloud data items in order to spot patterns and respond appropriately via notifications or triggers. Therefore, the collection and storage of Cloud data depend on the performance capabilities of databases. Which database management system is best for Cloud services is a major conundrum caused by the range of systems available today, Cloud services that need to store large amounts of data in databases need storage devices and rapid insertion queries, whereas agents that use database stored procedures and aggregation functions to utilize data-mining and deep learning algorithms to Cloud data need huge storage chunks and fast CPU processing for selection queries.

The most popular relational databases and the most widely used open source document databases—MongoDB [9], which is utilized by many Cloud services—are tested in this article. All of the scenarios that were looked at included Cloud datasets of Cloud sensory data, and the literature research that was conducted included an analysis of BLOB data utilized by Cloud streaming services. Since the authors are interested in databases that gather Cloud data, they have also done an experimental evaluation using MongoDB [9], MySQL [6, 10], and PostgreSQL [8]. The findings of the experimental evaluation are provided, evaluated, and debated. Based on ranking reports on the utilization of open source databases, authors chose the databases described above [3].

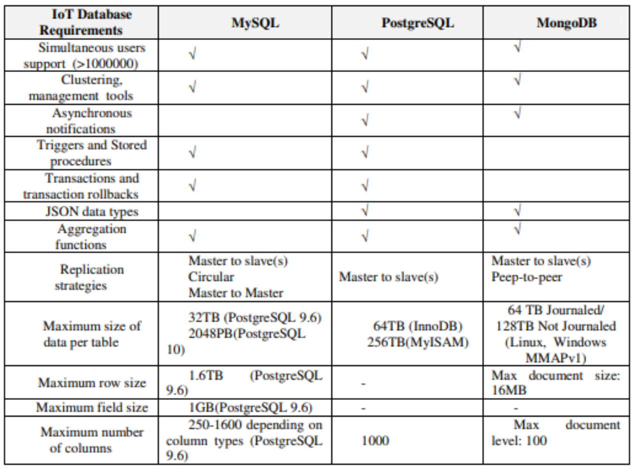
II. CLOUD CAPABILITIES FOR RELATIONAL AND DOCUMENT DATABASES

The primary functional and service differences between MySQL [6], PostgreSQL [8], and MongoDB [9] have received a lot of attention in the Aboutorabi literature, which has evaluated the performance evaluation on large-scale e-commerce data. The features of MySQL, PostgreSQL, and MongoDB with regard to distributed database functionality and replication, storage constraints, asynchronous notification capabilities, support for triggers and stored procedures, JSON data type support, and transactions are shown in Table 1 below [1].

On one hand, the distributed database engine of the MySQL database, which is more reliable than the PostgreSQL, allows a variety of replication services. Additionally, compared to PostgreSQL, MySQL displays higher storage restrictions. MongoDB collections have access to the OS's storage capabilities, but they impose different restrictions on the capacity and size of the documents added to each collection.

On the other hand, PostgreSQL and MySQL are the two databases that support all of the necessary features for an Cloud data storage system. Asynchronous notifications and JSON fields are not supported by MySQL. Asynchronous events can be transferred to other services at the database level using PostgreSQL notifications (PaaS). Similar to the MongoDB database, PostgreSQL JSON and its performance-improved JSONB features give the database the ability to store and process documents [5].

Table 1. Functionalities required by an Cloud database system amongst MySQL, PostgreSQL and MongoDB



IoT Database Requirements	MySQL	PostgreSQL	MongoDB
Simultaneous users support (>1000000)	√	√	√
Clustering, management tools	√	√	√
Asynchronous notifications		√	√
Triggers and Stored procedures	√	√	
Transactions and transaction rollbacks	√	√	
JSON data types		√	√
Aggregation functions	√	√	√
Replication strategies	Master to slave(s) Circular Master to Master	Master to slave(s)	Master to slave(s) Peep-to-peer
Maximum size of data per table	32TB (PostgreSQL 9.6) 2048PB(PostgreSQL 10)	64TB (InnoDB) 256TB(MyISAM)	64 TB Journaled/ 128TB Not Journaled (Linux, Windows MMAPv1)
Maximum row size	1.6TB (PostgreSQL 9.6)	-	Max document size: 16MB
Maximum field size	1GB(PostgreSQL 9.6)	-	-
Maximum number of columns	250-1600 depending on column types (PostgreSQL 9.6)	1000	Max document level: 100

Table 1

III. CLOUD DATA EXPERIMENTS AND RESULTS

The authors of this work have compared the performance of relational databases (MySQL 5.6.3 and PostgreSQL 9.6) and NoSQL databases (MongoDB 2.6.10). For the purposes of this article, a P4 at 3.2GHz single core PC with 2GB of RAM and a 120GB RAID 1 disc array is employed as the server. This configuration was chosen by the authors because it is the lowest cost SaaS configuration for small businesses on the Microsoft Azure cloud ($50/month for a virtual machine running Ubuntu Linux with a single core, 2GB of RAM, 128GB of storage, redundancy, and 100,000 storage transactions per month).

Because the authors sought to reduce network jitter and delays, the experimental database server was run locally using Python scripts. For MySQL, PostgreSQL, and MongoDB, the concurrent database connections limit is set to 2,000. 150,000 OS open file descriptors are allocated for MongoDB. Only the tested service (MySQL, PostgreSQL, or MongoDB) is running actively during the experiment. For a configuration value of 2,000 max connections, all database services need the same amount of memory. In order to minimize I/O transactions, the MySQL database setup uses the InnoDB storage engine, which has a pool buffer size of 1,3 GB (65 percent of the RAM that is available), using 512 KB for the total read and sort buffer sizes and 128 MB for the key buffer size.

Utilizing saved MongoDB data from an Cloud agriculture service, NoSQL databases have been assessed. Seven moisture sensors, a temperature sensor, and a servo valve actuator status (on off decision) are included in a group of documents. A small greenhouse has been equipped with sensor-actuator systems that provide data to the server on a regular basis (every 30 seconds). Similar in size to the experimental dataset for relational databases, the MongoDB dataset comprises 770,000 records overall. Using Cloud data, writers conducted the following experiments: Three query experiments: a select-find query, a burst insert query, and an aggregation function query. Ten different experiments have each been run. The values for query average response time have also been computed.

A. Performance Evaluation Metrics and Measures

The metrics utilised in the authors' testing scenarios are shown below in order to evaluate the performance of databases using data from Cloud applications. The time needed to complete a job, which translates to the time needed for the database service to complete a transaction, is the most crucial metric for the application layer protocol that executes database transactions (series of prepared SQL queries). The average number of queries per transaction and the average transaction execution time are then used to calculate the average query execution time. Calculations of query execution time are based on Equation 1.

$$T_{SQL} = T^{Q}_{STOP} - T^{q}_{START} \text{ (ms)} \tag{1}$$

Throughput is another statistic that is used to indicate the number of transactions-queries over time. In order to more correctly extrapolate how well the database handles varying loads and changing numbers of connections, database throughput assessments are typically made using the total number of queries per second rather than transactions. The most popular Equation 2 is used to measure inquiries per second and is used to calculate queries per second (QPS).

$$QPS = \frac{No\_queries\_per\_thread * No\_threads}{Total\_query\_time} \quad \text{(req/s)} \tag{2}$$

Authors suggest the query jitter measure (Qj), which reflects the fluctuation in database requests across time and is derived from Equation 3, for the process of estimating scalability:

$$Qj = TDB_{init} + \left| \frac{(dT_1 - dT_2)}{\sum R_1^{insert|update|} - \sum R_2^{insert|update|}} \right| \quad \text{(ms)} \tag{3}$$

If dT1 and dT2 represent the time needed to complete queries 1 and 2, respectively, and the sums represent the number of records returned from each query. The average startup and setup time for each query, TDB init, is calculated using a zero result query time estimate and is taken into account for each query type (insert, update, delete, and select).

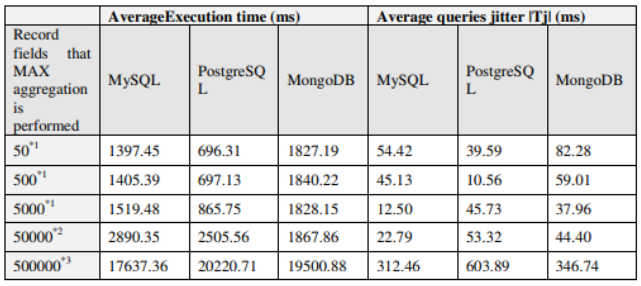
B. Experimental scenario, aggregation functions experimentation on Cloud data

The authors in this instance run a select query over a fixed number of records. Every time PostgreSQL, MySQL, and MongoDB's integrated MAX aggregating function is used. Using a transaction with 10 MAX queries and the transaction's overall execution duration, the scenario is run. The results of measuring the queries jitter time (taken from Equation 3) are displayed at Table 2.

Table 2 makes it evident that PostgreSQL aggregation function execution time works best for modest record sizes, followed by MySQL and MongoDB. When compared to relational databases, MongoDB's aggregation function measurement rates are stable. However, MySQL and PostgreSQL do not perform well for medium record sizes, providing room for MongoDB to perform more quickly and effectively. MySQL outperforms PostgreSQL and MongoDB for large record sizes.

MongoDB exhibits significant jitter for a limited number of queries when it comes to databases' scalability as defined by transaction jitter (Equation 3). It should be noted that for both medium and large queries, it fails to maintain a low jitter profile. This shows that records from a distributed database should not be used to execute MongoDB stored procedures. Instead, records from a single database should. For a small number of queries, PostgreSQL maintains the lowest jitter profile. This suggests that PostgreSQL can only be distributed if the applications carry out internal aggregations on tiny clustered data chunks. For both medium and large record size aggregations, MySQL exhibits the lowest jitter profile, indicating that its internal engine for procedural execution is the most suitable for clustered databases.

Table 2 shows measurements of the total transaction execution time and transaction jitter on a float field for a number of records using the MAX stored procedure.



	AverageExecution time (ms)			Average queries jitter |Tj| (ms)		
Record fields that MAX aggregation is performed	MySQL	PostgreSQL	MongoDB	MySQL	PostgreSQL	MongoDB
50^{*1}	1397.45	696.31	1827.19	54.42	39.59	82.28
500^{*1}	1405.39	697.13	1840.22	45.13	10.56	59.01
5000^{*1}	1519.48	865.75	1828.15	12.50	45.73	37.96
50000^{*2}	2890.35	2505.56	1867.86	22.79	53.32	44.40
500000^{*3}	17637.36	20220.71	19500.88	312.46	603.89	346.74

Table 2

IV. Conclusion

The authors compare the functionality of NoSQL databases to open source relational databases. Relational databases have a number of drawbacks, including an uncomfortable design, a lack of support for the normalisation forms and types used by Cloud services, limitations on the number of records that can be stored at once, and corruption that is more likely to occur with big data, which usually requires the use of special types. The solution in this case isn't necessarily a successful software repair.

PostgreSQL surpasses MySQL and MongoDB for a limited number of carefully chosen records, according to the trials and results of the authors. For a large number of selected records, MongoDB outperforms MySQL and PostgreSQL. For a large number of selected records (>20,000), MySQL surpasses PostgreSQL, but it still falls short of MongoDB in terms of performance.

The best database system for performing aggregating operations on a small number of Cloud data entries is PostgreSQL, according to tests with the execution of aggregation procedures. On the other hand, MySQL provides the best execution time results for an aggregating function applied on a large number of Cloud records. MongoDB is not a suggested option for the execution of aggregate procedures on Cloud data..

References

[1] Db-engines. (2018), "The DB-Engines Ranking ranks database management systems according to their popularity", Internet: https://db-engines.com/en/ranking [Oct. 2018]

[2] Fiannaca A. J. and Huang J. (2015), "Benchmarking of Relational and NoSQL Databases to Determine Constraints for Querying Robot Execution Logs." https://courses.cs.washington.edu/courses/cse544/15wi/projects/Fiannaca_Huang.pdf , Tech. Report [Feb 2017]

[3] Maksimov D. (2015)., "Performance Comparison of MongoDB and PostgreSQL with JSON types", Master Thesis, Tallin University of Technology, faculty of Information Technology, https://digi.lit.ttu.ee [May 2017]

[4] MariaDB foundation. (2015), "free MySQL database", Internet: https://mariadb.org [Jun. 2016]

[5] Fontaine D., (2017). "Pgloader tool." Internet: https://pgloader.io [Nov. 2017]

[6] MongoDB (2012). "MongoDB document database and documentation", Internet: https://docs.mongodb.com [Mar. 2015]

[7] Oracle Foundation. (2015).,"MySQL database", Internet: https://www.mysql.com [May. 2016]

[8] Parker Z., Scott P., Vrbsky V. Susan (2013), "Comparing NoSQL MongoDB to an SQL DB." Proceedings of the 51st ACM Southeast Conference. DOI: 10.1145/2498328.2500

[9] ROS-Open-Source Robotics Foundation. (2012). "Robot Operating system." Internet: http://www.ros.org/about-ros/ [Nov. 2016]

[10] Christodoulos Asiminidis., George Kokkonis.,(2018)," Database Systems Performance Evaluation for IoT Applications"