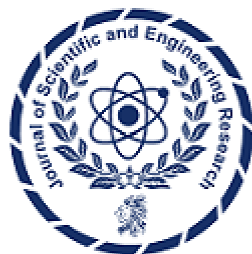
Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2019, 6(1):125-133



Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Comparing Relational and NoSQL Databases for carrying IoT data

Sotirios Kontogiannis[1], Christodoulos Asiminidis[1], George Kokkonis[2]

[1]Laboratory of Distributed Microcomputer Systems, Department of Mathematics, University of Ioannina, University campus, 45100 Ioannina, Greece
[2]Department of Business Administration, TEI of Western Macedonia, 65110 Grevena, Greece

Abstract Data stored in Internet of Things (IoT) storage repositories radically increases. Database vendors struggle in order to gain more market share, develop new capabilities and overcome the disadvantages of previous releases, while providing new features for the IoT industry. Taking into consideration the vast amount of database capacity and processing needed, as well as the exponential increase and use of IoT devices, storage and retrieval of sensory data is the main bottleneck and sets the boundary requirements for IoT services.
This paper compares open source relational databases and document databases, trying to pose an answer to the question which one performs better than the other over IoT datasets, carrying either binary large objects or small-size IoT data records or documents. It is a comparative study on the performance of the most commonly used Database Management Systems of the NoSQL MongoDB database and SQL databases of MySQL and PostgreSQL.

Keywords Database systems performance evaluation, document databases, relational database systems, IoT, IoT Data

1. Introduction

Internet of Things (IoT) relies on services that are able to sense, communicate and share sensory data. There is a huge amount of IoT data during such exchange processes, of either small in length data objects that carry sensory measurements or large binary data objects carrying multimedia streams or real-time haptic streams of robotic actuators control and feedback responses. IoT usage in everyday life has become easier and smarter in a sense that technologically evolved IoT devices equipped with sensors and transponders are used in houses, cities, transportation and agriculture.
The primary tasks of IoT services are to acquire, filter, analyze and mine IoT data objects, so as to identify patterns and take appropriate actions accordingly via notifications or triggers. Thus, databases performance capabilities are crucial and significant for the storage and retrieval of IoT data. The variety of today’s databases management systems has arisen a big dilemma on which one is the most suitable for IoT services. The amount of data that need to be stored by IoT services into databases requires disk storage and fast insertion queries, while agents that apply data-mining and deep learning algorithms on IoT data require big memory chunks and CPU processing capabilities for selection queries, since they use database stored procedures and aggregation functions.
In this paper the most commonly used open source document database of MongoDB [9] used by many IoT services and the most commonly used relational databases are put to test. All the examined scenarios include IoT datasets of IoT sensory data, while the performed literature review includes evaluation of BLOB data used by IoT streaming services. Since the authors’ interest is targeted onto databases that collect IoT data, an experimental evaluation has been also conducted by the authors, using MongoDB (MongoDB, 2014), MySQL

[6, 10] and PostgreSQL [8] and the experimental results are presented, analyzed and discussed. Authors' database selection described above was based on ranking reports on use of open source databases [3].

Relational and document databases IoT capabilities

According to the literature Aboutorabi presents a performance evaluation on big e-commerce data, focusing also on the main differences in functionalities and services between MySQL [6], PostgreSQL [8], MongoDB [9]. Table 1 below presents the MySQL, PostgreSQL and MongoDB capabilities in terms of distributed database functionalities and replication, storage limits, asynchronous notification capabilities, triggers and stored procedures support, JSON datata type support and transactions [1].

Table 1: MySQL, PostgreSQL and MongoDB cross comparison of supported functionalities required by an IoT database system

IoT Database Requirements	PostgreSQL	MongoDB	MySQL
Simultaneous users support (>1000000)	√	√	√
Clustering, management tools	√	√	√
Asynchronous notifications	√	√	
Triggers and Stored procedures	√		√
Transactions and transaction rollbacks	√		√
JSON data types	√	√	
Aggregation functions	√	√	√
Maximum size of data per table	256TB(MyISAM)	128TB	2048PB
Maximum row size	-	Max document size: 16MB	1.6TB
Maximum number of columns	1000	Max document level: 100	1600
Maximum field size	-	-	1GB
Replication strategies	Master to slave(s)	Master to slave(s) Peep-to-peer	Master to slave(s) Circular Master to Master

From Table 1, PostgreSQL supports all of the required functionalities for an IoT data storage system, followed by MySQL. MySQL lacks support of asynchronous notifications and has no JSON field support. PostgreSQL notifications can be used to transmit asynchronous incidents to other services at the database level (PaaS). PostgreSQL JSON and improved version in terms of performance JSONB fields add to the database the functionality to store and process documents similarly to MogoDB database [5].

MySQL database on the other hand has support of various types of replication services and its distributed database engine is more robust than the postgreSQL. Furthermore MySQL presents higher capacities storage limits than postgreSQL. MongoDB collections have the storage capabilities of the OS used; however enforce separate limitations in terms of capacity to the documents' sizes inserted to each collection.

Performance evaluation survey on IoT Blob data

Starcu-Mara and Baumann's examined benchmarks of the leading commercial and open-source databases on Binary Large Objects (BLOB) [13]. Their experimental scenarios also included the open-source databases of PostgreSQL and MySQL. PostgreSQL version used 8.2.3 and MySQL version was 5.0.45. In that study turned out that PostgreSQL had a better select queries performance than MySQL specifically for BLOB sizes bellow 5MB.

MySQL was also more efficient during insert queries compared to PostgreSQL for BLOB sizes above 100KB. Finally, MySQL outperformed PostgreSQL in select queries of BLOB sizes above 5MB. Both, for large BLOB sizes, MySQL and PostgreSQL presented similar Master-slave scalability performances. The MySQL and PostgreSQL read (select) and write (insert) performance results are shown in Figures 1 and 2 accordingly [13].

According to the authors of [14], used a big number of records(>100,000) of maximum 1KB in record size has been added into MySQL and PostgreSQL databases and from the collected results it turned out that MySQL is faster than PostgreSQL. However, PostgreSQL responds faster in cases of concurrency and contention increase for small servicing requests rates (up to 100req/sec).

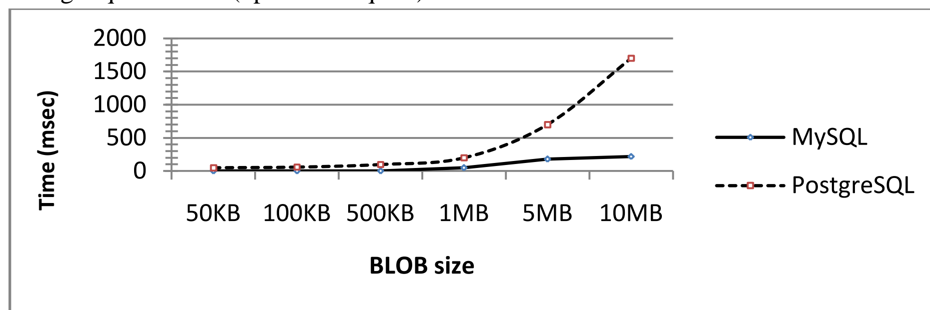


Figure 1: Large BLOB insert queries performance of MySQL and PostgreSQL

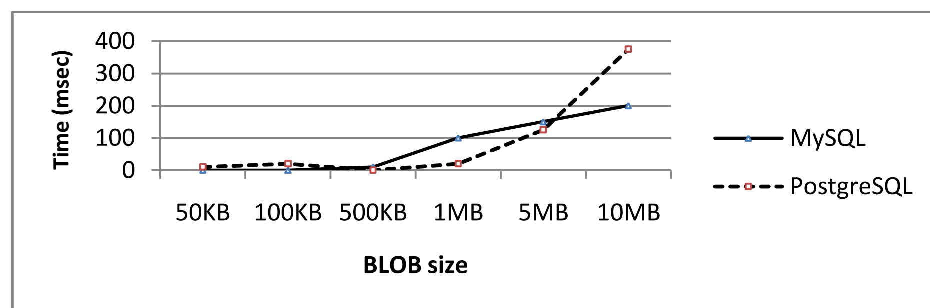


Figure 2: Large BLOB select queries performance of MySQL and PostgreSQL

Based on an analysis conducted over an e-shop web application using MySQL and MongoDB databases accordingly, the performance of MongoDB was better when compared to that of MySQL [2]. The following two Figures 3, 4, show the big time difference execution between 100 numbers of returned records and 25.000 numbers of returned records during a single query for MySQL and MongoDB. Performance has been measured in terms of throughput (queries/sec) over the records stored or returned.

The authors of [11] are using modest-sized structured database sizes (100,000 records) in order to compare the performance of the MySQL database with the MongoDB database. The results show that at the burst insert queries experiment, the MySQL performs better than MongoDB in queries less than 1MB. Despite the fact that is not clear enough that MySQL and MongoDB perform similarly, in queries above 1MB both of them have almost the same insert response time. For select queries experimentation in [11], as record sizes increase (more than 700Kbytes of records sizes data per transaction) then MongoDB and MySQL present similar execution time. The same occurs for low size transactions (less than 100Kbyte records sizes. For records of mean size 100KByte-700Kbyte), MongoDB outperforms MySQL. Concluding, the overall select experiment shows that the MySQL database performance is worse than MongoDB. The results on the average time to perform select all records query and select 10000 records query on a modest size database (100,000 records).

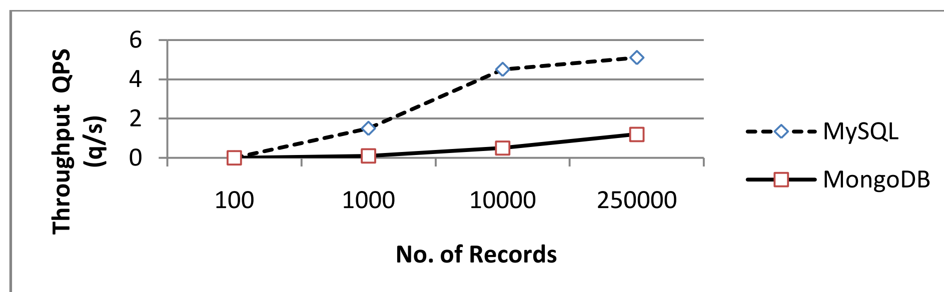


Figure 3: Select-find queries per second over number of returned records

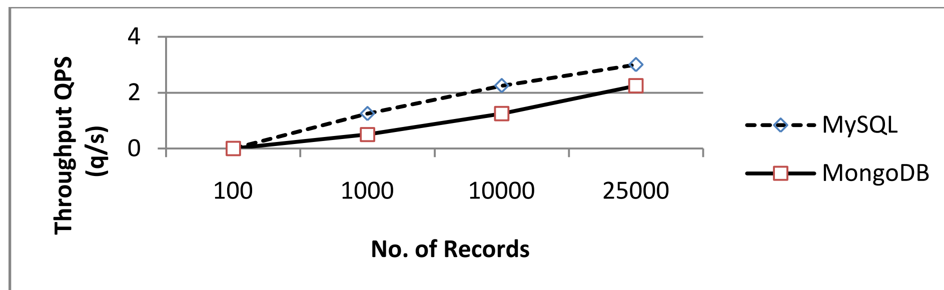


Figure 4: Insert queries over number of stored records

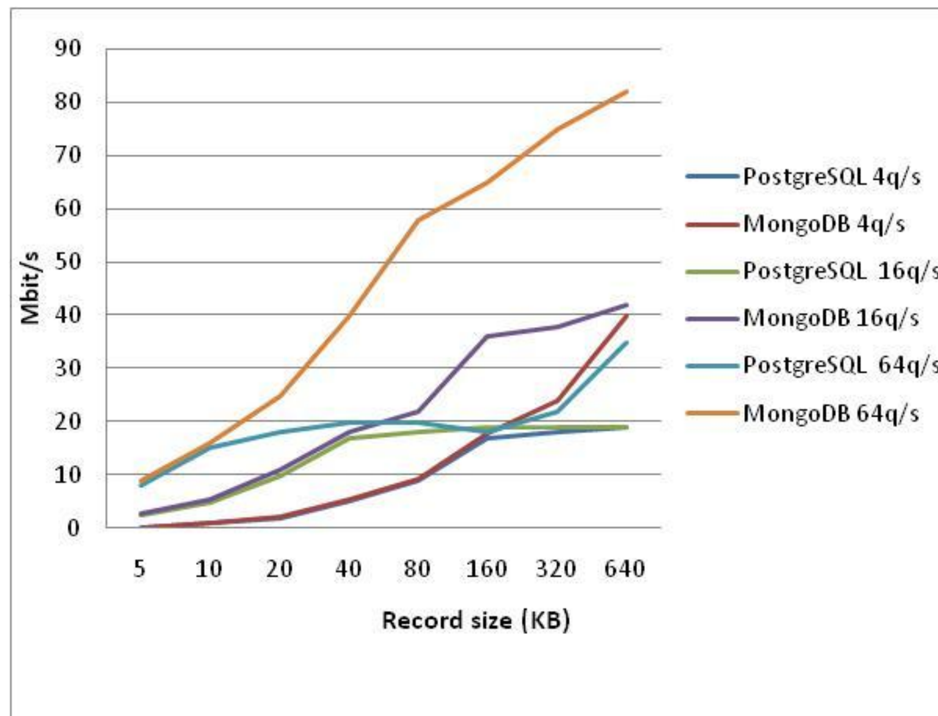


Figure 5: Network Throughput (MBits/sec) VS Record size (KB)

According to Fiannca study [4], the potential throughput of the MongoDB and PostgreSQL databases has been evaluated in order to determine the best database store for a future application embedded in the current Robot Operation System (ROS system) [12]. PostgreSQL performed significantly worse than MongoDB and the results are presented at Figure 5. This is perhaps not particularly surprising because MongoDB is specifically designed for handling JSON data whereas PostgreSQL is designed to manage relational data only with extensions for JSON document data. In addition, the transformations from relational data to JSON document data are time consuming in terms of performance [4].

Experimental scenarios and results on IoT data

Authors' experimental scenarios include performance measurements of relational databases (MySQL 5.6.3 and PostgreSQL 9.6) and NoSQL (MongoDB 2.6.10) database. For this purpose the server used is a P4 at 3.2GHz single core PC with 2GB of RAM and a RAID 1 disk array of 120GB. The authors deliberately used such an old fashioned server configuration, since it is the minimum monthly price SaaS configuration offered by the Microsoft Azure cloud, for small companies ($50/month for a virtual machine running on Ubuntu Linux, with 1 core, 2GB RAM, 128GB storage and redundancy and 100,000 storage transactions per month.

To minimize network delays and jitter, database queries have been performed locally in the experimental database server using python scripts. The number of concurrent database connections is set to 2,000 for MySQL, PostgreSQL and for MongoDB. Specifically for MongoDB the number of OS open file descriptors is set to 150,000. During the experimentation, only the tested service (MySQL, PostgreSQL or MongoDB) is the active

service running. All database services use the same amount of memory for a 2,000 max_connections configuration value. MySQL database configuration uses InnoDB storage engine, with a pool buffer size of 1,3GB (65% of the available memory) to reduce I/O transactions, using 512KB of total read and sort buffer sizes and 128MB of key buffer size. PostgreSQL uses 1,3GB of shared_buffers. MongoDB has no memory size restriction configuration parameter and uses the whole memory in respect to other services. Since the OS system and services use up to 500-700MB of resident memory, authors confirm that the file memory mappings of MongoDB do not exceed the 1.3GB of memory, during experimentation.

For the MySQL and PostgreSQL databases authors used a medium content-size IoT data content of a meteorological station that includes 1-year minute measurements (up to 570,000 records). Each database record contains fields of sensory measurements of time, temperature, humidity, pressure, dew point, rainfall, wind speed and wind direction. Since all data are stored as variable char fields, each record size varies from 48-128Bytes of data. The original meteorological station database was a MySQL database, which the authors also migrated to PostgreSQL using the pgloader tool [7].

For the process of evaluating a NoSQL database, authors used MongoDB stored data coming from an IoT agricultural service. This service includes a collection of documents coming from 7 moisture sensors, a temperature sensor and a servo valve actuator status (on|off decision). Such sensors-actuator systems are placed in a small greenhouse and transmit periodically (every 30s) data to the server. The MongoDB dataset has a total of 770,000 records of similar size to the relational databases experimental dataset. The following experiments have been performed by authors using IoT data: 1. A select-find query experiment, 2. a burst insert query experiment and 3.an aggregation function query experiment. For each one measurement has been performed 10 times and average response time query values have been calculated.

Performance evaluation metrics

In order to measure databases performance using IoT application data, authors present the metrics used in their experimentation scenarios below. The most important metric for the application layer protocol that performs database transactions, is the time required for completing a task, which is translated to the time required for the database service to complete a transaction (series of prepared SQL queries). Then the average query execution time is derived from the average number of queries per transaction and the average transactions execution time. Queries execution time calculations are based on Equation 1.

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

$$T^{Estimated}_{SQL} = \sum_{all\ steps} Step\ Estimated\ Execution\ time \qquad \text{(ms)} \qquad \textbf{(1.a)}$$

Another metric used that expresses the number transactions-queries over time is throughput. Database throughput measurements are performed using mainly the total number of queries per second rather than transactions, as it extrapolates more accurately how well the database copes with different loads and different numbers of connections. To calculate the queries per second the following most widely known Equation 1.a is used that measures Queries Per Second (QPS).

$$QPS = \frac{No_queries\ _per\ _thread * No_threads}{Total\ _query\ _time} \qquad \text{(req/s)} \qquad \textbf{(1.b)}$$

Authors used a slight variation of the throughput metric (QPS - Equation 1.a), where No. records are the number of records inserted or updated or selected (returned) or deleted from a query, TDB_init is the time spend on a query that inserts or updates or selects or deletes zero records and query_time is the average calculated query time of a transaction that is performed by a single thread (Equation 1.d).

$$QPS = \frac{No_queries\ _per\ _thread * No_threads}{Total\ _query\ _time} \rightarrow \frac{No_{records}\ [insert\ |update\ |select\ |delete\ ]}{\overline{query\ _time} - T_{DB\ _init}} \text{(rec|q/s)} \ \textbf{(1.c)}$$

Since throughput is measured using queries and not transactions, authors transformed Equation 1.d, similarly to QPS metric, that expresses the total queries per second, in order to quantify databases processing efforts, using the Equation 2:

$$QPS' = \frac{Total\ number\ \ of\ transactions\ \ |thread\ queries}{T_{Transaction\ \ |Thread}} \rightarrow \frac{\sum_{i=1}^{k} Q_i}{T} \rightarrow \frac{\bar{Q}}{\bar{T}} \text{(trans|q/s)} \qquad \textbf{(2)}$$

For the process of scalability estimation authors propose the query jitter metric (Qj) which is calculated using Equation 3 and expresses database queries variation over time:

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

where the sums $\sum R_1^{\text{insert |update |}}, \sum R_2^{\text{insert |update |}}$ are the number of records returned from queries 1 and 2 accordingly and dT1, dT2 is the time required completing the queries. TDB_init is the average initialization and setup time for each query which is assumed as a constant coefficient parameter for each query type (insert, update, delete, select) accordingly and is calculated experimentally using a zero result query time estimate. Similarly to Equation (3), database Transactions jitter (Tj) can also be measured using number of Queries instead of number of records returned according to Equation 4:

$$Tj = \left| \frac{(dT_1 - dT_2)}{\sum_{i=1}^{k} Q_i^{Tr\,1} - \sum_{i=1}^{l} Q_i^{Tr\,2}} \right| \text{ (ms)} \quad \textbf{(4)}$$

where $\sum_{i=1}^{k} Q_i^{Tr\,1}$ is the sum of (1..k) queries of transaction 1 and dT1 is the transaction 1 execution time and where $\sum_{i=1}^{l} Q_i^{Tr\,2}$ is the sum of (1..l) queries of transaction 2 and dT2 is the transaction 2 execution time. Transactions jitter is database scalability metric and it can identify queries or transactions variations over a distributed-partitioned database system. Qj scalability measurements are performed using a standard number of Query records selected, inserted or updated. Tj scalability measurements are performed on a fixed number of queries per transaction.

In cases where it is impractical to calculate the total number of queries between two transactions or the number of records returned from two consecutive queries, authors propose a more practical measurable metric of the normalized query jitter |Qj| or normalized transaction jitter |Tj|, corresponding to the metrics denoted by Equations 3, 4 for scalability estimation. Normalized jitter is measured by executing the same transaction or query k times with a per query-transaction interval set for IoT databases between 500ms-2s and calculate jitter time according to the following equation:

$$Tj^2 = \frac{\sum_{i=1}^{k} (dT_i^{Query\ |Transaction} - \overline{dT})^2}{k-1} \text{ (ms)} \quad \textbf{(5)}$$

where $\overline{dT}$ is the mean execution time for the k sample queries or transactions.

Experimental Scenario

The experimental scenario involves select-find queries, since the IoT applications or agents use frequently this type of queries, to interrogate the databases and acquire records for further evaluation. In this experiment authors perform queries that return a fixed number of records and measure the queries execution time in MySQL, PostgreSQL and MongoDB. The IoT database contains data that have been recorded from a meteorological station.

The total execution time for 1 up to 500.000 (500K) returned records in the IoT database has been presented in the Figure 6 below. The returned records average data size can be estimated to 64Bytes multiplied by the value of x axis number of returned query records.

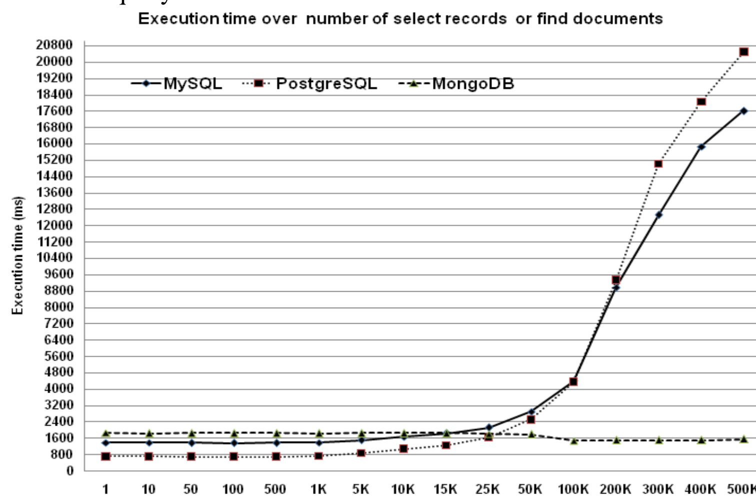


Figure 6: Total Execution time in IoT databases over number of records

For small record sizes of up to 100,000 IoT records returned by a select-find query, which equals to data transfer of up to 6.1MByte, PostgreSQL is faster than MySQL from 48% for one returned record query down to 0.08% for a 100K records query. PostgreSQL performs better for small (<100K records) number of records returned from an IoT database presenting an average of 36.5% more throughput than MySQL (see Figure 7, 1-100,000 records).

For big data transactions (above 7MB of transferred data >100,000 returned records), MySQL outperforms PostgreSQL at an average of 18% based on query execution time measurements. In terms of throughput, MySQL, for big data transactions performs better than PostgreSQL at an average of 12%, starting from 1.19% for 200,000 returned records up to 14.10% for 500,000 returned records (see Figure 7).

PostgreSQL performance is close to that of MySQL for big data queries, if the queries in a transaction are clustered to small returned record queries executed back to back. In such cases PostgreSQL presents a performance boost of 10% and reaches close to the MySQL performance (performs 0.5% worse than MySQL for >250,000 returned records and up to 4.1% for 500,000 records).

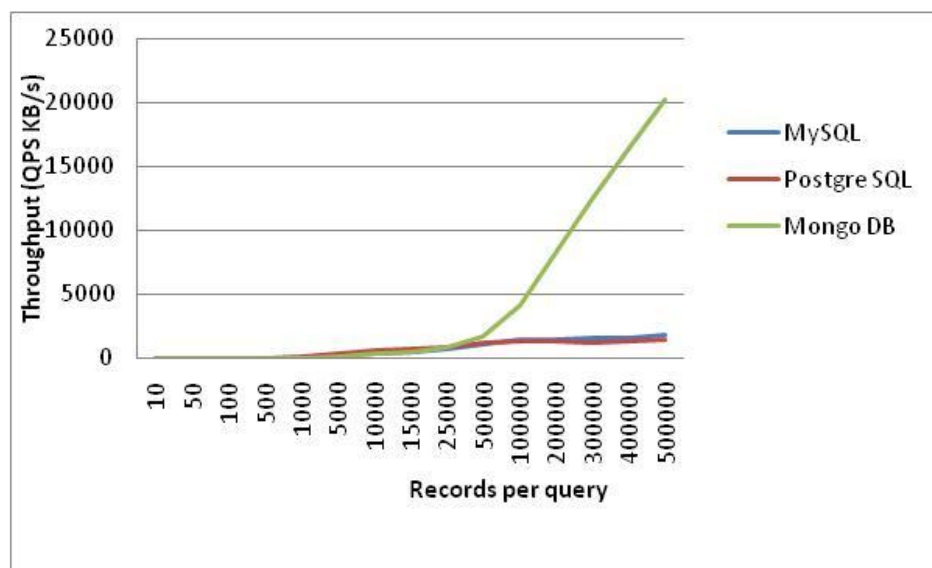


Figure 7: Throughput over query number of records

PostgreSQL performance is close to that of MySQL for big data queries, if the queries in a transaction are clustered to small returned record queries executed back to back. In such cases PostgreSQL presents a performance boost of 10% and reaches close to the MySQL performance (performs 0.5% worse than MySQL for >250,000 returned records and up to 4.1% for 500,000 records).

For 500,000 returned records (30.5MB of transferred data), there is a curve-bend in MySQL execution time, which reaches the conclusion that above 500,000 records the performance difference in terms of throughput between MySQL and PostgreSQL is close to 14-18%(more than 50Mbyte search data per transaction). However, above 1,000,000 records PostgreSQL and MySQL might present similar performance results due to database server high CPU utilization.

MongoDB's throughput performance for small queries (up to query size of 1.52 MB per transaction -25,000 records) is 51% worse than PostgreSQL and 20% (on average) MySQL. In terms of execution time all MongoDB measurements keep a stable execution time profile close to 1600ms for queries returning records bellow 25,000, that drops to 1450-1500ms for queries returning records >25,000. That is, MongoDB outperforms MySQL in terms of throughput by 69% on average for returned records above 20,000 and outperforms by 72% on average PostgreSQL for returned data records above 30,000 (see Figure 7).

Figure 7 presents the throughput of MySQL, PostgreSQL and MongoDB. Based on these results, PostgreSQL is the best database system for up to medium sized IoT select queries, while outperformed by PostgreSQL MongoDB maintains a quite stable execution time performance. For big data transfers and for the relational databases, MySQL outperforms PostgreSQL. However, MongoDB significantly outperforms MySQL and in

some cases with double throughput data rates. Finally, for medium size transactions (from 25.000-100.000 returned records, which corresponds to an average of 3MB of total data transfers), MongoDB followed by PostgreSQL manage to maintain better throughput results.

Based on the bibliographic evaluation results the authors took their research one step further, by trying to evaluate the performance of MySQL, PostgreSQL and MongoDB on IoT data sets. The summary experimental results of authors' experimentation are presented at Table 3 for small, medium and big number of query records for insert, select and aggregation call queries.

According to authors' experimentation, experimental scenarios confirm that MySQL database is the best performing database for big number of records maintaining a good stored procedure execution performance and fair select and insert queries performance. Even if the best insert performance for big queries belongs to PostgreSQL and the best find to MongoDB, MySQL as a total maintains the best performance profile. Authors need to pinpoint here that MongoDB might perform as similar or even better than MySQL for very big number of insert queries (Table 3 '+' sign on MySQL and MongoDB insert performance for very big number of records).

Table 2: MySQL, PostgreSQL and MongoDB literature summary performance table for select, insert queries on Blob IoT data

Record Size	**Performance Summary (Best, Fair, Worst)**					
	MySQL		**PostgreSQL**		**MongoDB**	
Medium size queries	Insert	Select	Insert	Select	Insert	Find
	Fair	Worst	Worst	Fair-Best	Best	Best-Fair
Big size queries	Insert	Select	Insert	Select	Insert	Find
	Best	Fair	Worst	Worst	Fair	Best

Table 3: MySQL, PostgreSQL and MongoDB summary performance table for select, insert queries and aggregation functions on IoT data

Record Size	**Performance Summary (Best, Fair, Worst)**								
	MySQL			**PostgreSQL**			**MongoDB**		
Small number of records	Insert	Select	Call	Insert	Select	Call	Insert	Find	Call
	Worst	Fair	Fair	Fair	Best	Best	Best	Worst	Worst
Medium number of records	Insert	Select	Call	Insert	Select	Call	Insert	Find	Call
	Best	Worst	Fair	Fair	Fair	Best	Worst	Best	Best
Big number of records	Insert	Select	Call	Insert	Select	Call	Insert	Find	Call
	Fair+	Fair	Best	Best	Worst	Fair	Worst+	Best	Worst

Conclusions

In this paper, authors undergo at first a performance evaluation survey between relational databases and NoSQL databases. Their disadvantages however lay on the unease design for IoT services, their limitations on maximum storage records, and their breakage prone on big data that in most cases requires the use of special type and not always successful repair software and migration harshness from database to database. NoSQL databases are relatively new and become a popular trend for IoT, as they provide horizontal schema-less collections, extremely useful for IoT data coming from different sources of different structure, sensory hardware and transmission protocols.

According to the survey results on databases performance evaluation, the examined relational databases were MySQL and PostgreSQL. For NoSQL databases the MongoDB has been examined. From these three selected databases, on most evaluation reports, MySQL presented good performance on big number of records for insert queries over MongoDB and MongoDB outperformed MySQL for big select-find queries. PostgreSQL presented the worst performance for both insert and select queries.

Experimental scenarios confirm that PostgreSQL is the best performing database for small number of record queries for insert, select and aggregation function queries. MongoDB presents the best performing profile for

medium size queries for find queries and aggregation calls. However, for insert queries MongoDB is the best performing database only for a small number of insert record queries back to back.
Comparison results between table 2, found from the literature and table 3, from the authors' experimentation, show that for select queries the results are similar. For the insert queries the literature promotes MySQL as the best candidate and MongoDB as a fair candidate. However, the authors' experimental results show that MongoDB is the worst candidate. This can be explained due to the different nature of BLOB records inserts and small chunked IoT data inserts.
Finally, from the experimental measurements of transactions jitter, that expresses databases scalability, it is obvious that PostgreSQL database presents the least time-deviations of query execution followed by MongoDB, with worst scalability candidate the MySQL database, for back to back insert queries. Nevertheless, for aggregation functions execution MySQL database presents the least time deviations (jitter), followed by MongoDB. This leaves PostgreSQL the worst scalable database implementation for performing aggregation calls.

References

[1]. Aboutorabi S., Rezapour M., Moradi, M., and Ghadiri, N., "Performance evaluation of SQL and MongoDB databases for big e-commerce data", In proc. of CSICSSE conf., DOI: 10.1109/CSICSSE.2015.7369245, (2015)
[2]. Damodaran D. B., Salim S. and Vargese M. V., "Performance evaluation of MySQL and MongoDB databases", International Journal of Cybernetics & Informatics IJCI, Vol. 5, No. 2, ISSN:2320-8430, (2016)
[3]. Db-engines., "The DB-Engines Ranking ranks database management systems according to their popularity", Internet: https://db-engines.com/en/ranking, (2018) [Oct. 2018]
[4]. Fiannaca A. J. and Huang J. (2015), "Benchmarking of Relational and NoSQL Databases to Determine Constraintsfor Querying Robot Execution Logs." https://courses.cs.washington.edu/courses/cse544/15wi/projects/Fiannaca_Huang.pdf , Tech. Report, (2015)
[5]. Maksimov D., "Performance Comparison of MongoDB and PostgreSQL with JSON types", Master Thesis, Tallin University of Technology, faculty of Information Technology, https://digi.lit.ttu.ee, (2015) [May 2017]
[6]. MariaDB foundation., "free MySQL database", Internet: https://mariadb.org, (2015) [Jun. 2016]
[7]. Fontaine D., "Pgloader tool." Internet: https://pgloader.io, (2017) [Nov. 2017]
[8]. PostgreSQL., "PostgreSQL: The World's Most Advanced Open Source Relational Database", Internet: https://www.postgresql.org, (1996) [Apr. 2010]
[9]. MongoDB ., "MongoDB document database and documentation", Internet: https://docs.mongodb.com, (2014) [Mar. 2015]
[10]. Oracle Foundation.,"MySQL database", Internet: https://www.mysql.com, (2015) [May. 2016]
[11]. Parker Z., Scott P., Vrbsky V. S., "Comparing NoSQL MongoDB to an SQL DB". Proceedings of the 51st ACM Southeast Conference. DOI: 10.1145/2498328.2500, (2013)
[12]. ROS-Open-Source Robotics Foundation., "Robot Operating system." Internet: http://www.ros.org/about-ros/, (2012) [Nov. 2016]
[13]. Stancu-Mara, S., and Baumann, P., "A Comparative Benchmark of large Objects in Relational Databases". In Proc. of the 2008 international symposium on Database engineering and applications, pp. 277-284, ACM, (2008)
[14]. Sullivan P., "Comparing PostgreSQL 9.1 vs MySQL 5.6 using Drupal 7.x." Internet: http://posulliv.github.io/2012/06/29/mysql-postgres-bench/, June, 29, [Feb, 2017], (2012)