DATABASE SYSTEMS PERFORMANCE EVALUATION FOR IOT APPLICATIONS

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

[1]Laboratory team of Distributed Microcomputer systems, Department of Mathematics, University of Ioannina, Ioannina, Greece
[2]Department of Business Administration, TEI of Western Macedonia, Grevena, Greece

ABSTRACT

The amount of data stored in IoT databases increases as the IoT applications extend throughout smart city appliances, industry and agriculture. Contemporary database systems must process huge amounts of sensory and actuator data in real-time or interactively. Facing this first wave of IoT revolution, database vendors struggle day-by-day in order to gain more market share, develop new capabilities and attempt to overcome the disadvantages of previous releases, while providing features for the IoT.

There are two popular database types: The Relational Database Management Systems and NoSQL databases, with NoSQL gaining ground on IoT data storage. In the context of this paper these two types are examined. Focusing on open source databases, the authors experiment on IoT data sets and pose an answer to the question which one performs better than the other. It is a comparative study on the performance of the commonly market used open source databases, presenting results for 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) refers to services that are able to sense, communicate and share data. There is a vast amount of IoT data during such exchange processes, of small in length data objects. The primary tasks of IoT services are to acquire, filter and analyze data objects, so as to initiate specifications and measurements. Thus, databases performance capabilities are crucial and significant for the storage and management of IoT data. The variety of today's databases management systems has put users in a big dilemma on which one is the most suitable for each offered IoT service.

Database systems started gaining ground in the 60's. Different types have been developed, each one using its own data representation schema. Initially set as navigational databases based on linked-lists, transformed later on to relational databases with joins, triggers, functions, stored procedures and object-oriented capabilities. In the late 2000s NoSQL emerged and became a popular trend. The most commonly used database implementations today are based on the relational model which uses SQL as its query language. However, NoSQL database solutions are becoming more popular as big amounts of rapidly growing unstructured data are being deposited, overlapping strict relational databases performance and scalability constraints. That brought up the question if the relational model came to its dawn. On the one hand, relational databases use normality forms on the idea of data separated into field's records and tables, following

DOI : 10.5121/ijdms.2018.10601

normalization rules. On the other hand NoSQL databases escape for normality and re-design of scalable services manages to offer a robust solution in terms of performance.

The normalization procedure is basically based on the concepts of normal forms. A relation table is said to be in a normal form if it fulfills a certain set of constraints. There are 6 defined normal forms: 1NF, 2NF, 3NF, BCNF, 4NF and 5NF. Normalization should get rid of whatever is not needed but not at the cost of integrity. De-normalization is the inverse process of normalization, where the normalized schema is converted into a schema which has redundant information. Relational databases performance improved by using redundancy and keeping the redundant data consistent.

De-normalization can also be defined as the tactic of saving the join of superior normal form relations as base lower normal form relations. That way it decreases the number of tables and complicated table joins because a bigger number of joins can delay the process. There are various de-normalization methods such as: Storing derivable values, pre-joining tables, hard-coded values, keeping details with master, aggregations and views functions. Specifically for relational databases, although views functionality has restrained the performance problem with the pre-calculation of tables' aggregation functions of many tables; significant performance counter parts still remain unsolved. De-normalized schema can greatly improve performance under extreme read-loads but the updates and inserts become perplexing as the data is duplicated and hence have to be updated/inserted in more than one places

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 raised 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 [9], 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].

2. RELATIONAL AND DOCUMENT DATABASES IOT CAPABILITIES

According to the Aboutorabi's literature which has tested the performance evaluation on big e-commerce data, strongly concentrated 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 concerning distributed database functionalities and replication, storage limits, asynchronous notification capabilities, triggers and stored procedures support, JSON data type support and transactions [1].

Table 1. Functionalities required by an IoT 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

On the one hand, MySQL database supports various types of replication services and its distributed database engine which is more robust than the PostgreSQL. In addition, MySQL shows bigger storage limits than PostgreSQL. MongoDB collections have the storage capabilities of the OS; however enforce diverse limitations in respect of capacity to the documents' sizes inserted to each collection.

On the other hand, PostgreSQL supports all of the required functionalities for an IoT data storage system, followed by MySQL. MySQL lacks of support of asynchronous notifications and has no JSON field support. PostgreSQL notifications can be used to transfer asynchronous events to other services at the database level (PaaS). PostgreSQL JSON and improved version regarding performance JSONB fields add to the database the functionality to store and process documents similarly to MongoDB database [5].

3. RELATED WORK ON IOT DATA

Benchmarks of the leading commercial and open-source databases on Binary Large Objects have been examined by Starcu-Mara and Baumann's. [13]. Experimental scenarios include the open-source databases of PostgreSQL and MySQL. PostgreSQL version used 8.2.3 and MySQL version was 5.0.45. This survey has shown that PostgreSQL had much better select queries performance than MySQL on BLOB sizes bellow 5MB.

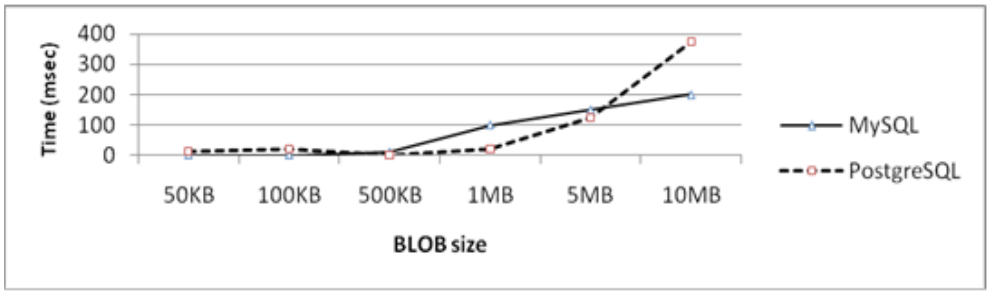

Figure 1. Big data insert queries performance of MySQL and PostgreSQL

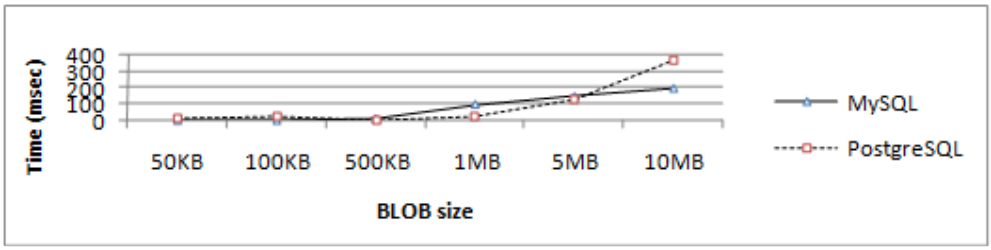

Figure 2. Big data select queries performance of MySQL and PostgreSQL

PostgreSQL compared to MySQL was not much more efficient during insert queries for BLOB sizes above 100KB. It turned out that MySQL outperformed PostgreSQL in select queries of BLOB sizes above 5MB. For big BLOB sizes, MySQL and PostgreSQL showed similar Master-slave scalability performances. The MySQL and PostgreSQL read (select) and write (insert) performance results are shown in Figures 1 and 2 correspondingly [13].

Considering the study that has been conducted by the [14], authors used a big number of records(>100,000) of maximum 1KB in record size. They made the research on MySQL and PostgreSQL databases and from the collected results they concluded that MySQL is faster than PostgreSQL. However, PostgreSQL is faster in case of concurrency and contention increase for small servicing requests rates (up to 100req/sec).

An e-shop web application analysis using MySQL and MongoDB databases accordingly has been carried out by [2] and has shown that the performance of MongoDB was better when compared to that of MySQL [2]. Figures 3, 4, show the execution time difference between 100 numbers of returned records and 25.000 numbers of returned records during a single query for MySQL and MongoDB. The performance evaluation has been also shown throughput (queries/sec) proportional to the records stored or returned.

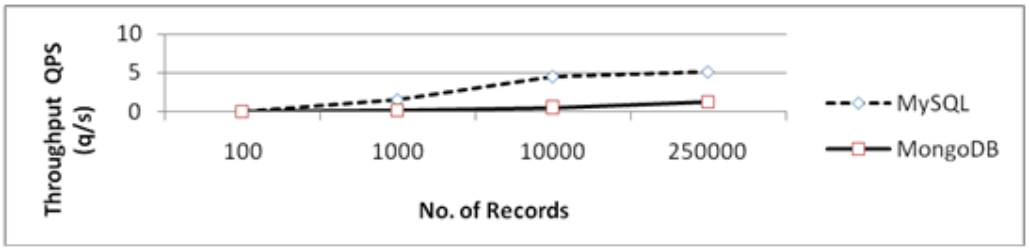

Figure 3. Select-find queries per second over number of returned records

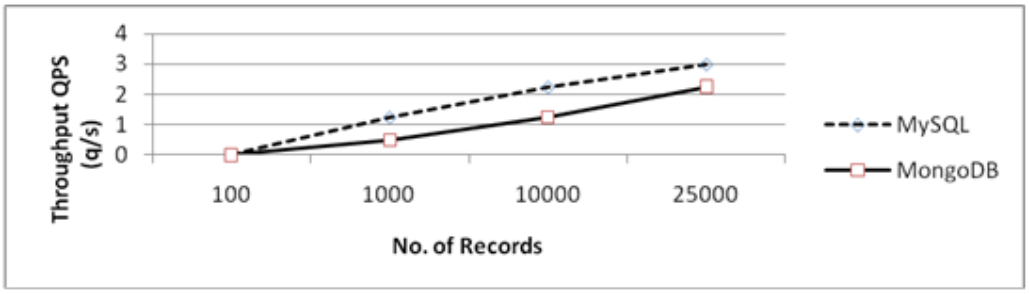

Figure 4. Insert queries over number of stored records

Table 2. Measurements of throughput (MBits/sec) over back to back insert-select queries/sec and record size

Insert/Select Operations	Record size (KB)	PostgreSQL	Mongo DB
4 q/s	5	0 .1Mbit/s	0.2Mbit/s
	10	0 .9Mbit/s	1Mbit/s
	20	2 Mbit/s	2 .2Mbit/s
	40	5 Mbit/s	5.4Mbit/s
	80	9 Mbit/s	9 .2Mbit/s
	160	17 Mbit/s	18Mbit/s
	320	18 Mbit/s	24 Mbit/s
	640	19 Mbit/s	40 Mbit/s
16 q/s	5	2.5Mbit/s	2.7Mbit/s
	10	5 Mbit/s	5.5Mbit/s
	20	10 Mbit/s	11Mbit/s
	40	17 Mbit/s	18Mbit/s
	80	18 Mbit/s	22 Mbit/s
	160	19 Mbit/s	36 Mbit/s
	320	19 Mbit/s	38 Mbit/s
	640	19 Mbit/s	42 Mbit/s
64 q/s	5	8 Mbit/s	9Mbit/s
	10	15 Mbit/s	16 Mbit/s
	20	18 Mbit/s	25 Mbit/s
	40	20 Mbit/s	40 Mbit/s
	80	20 Mbit/s	58 Mbit/s
	160	18 Mbit/s	65 Mbit/s
	320	22 Mbit/s	75 Mbit/s
	640	35 Mbit/s	82 Mbit/s

According to the study that has been carried out by [11], they are using modest-sized structured database sizes (100,000 records) to compare the performance of the MySQL database with the MongoDB database.

The results present that at the burst insert queries experiment, the MySQL outperforms MongoDB in queries less than 1MB. MySQL and MongoDB perform similarly, in queries above 1MB both of them have almost the same insert response time. For select queries experimentation, as record sizes increase (more than 700Kbytes of records sizes data per transaction) then MongoDB and MySQL present similar execution time. That phenomenon happens for low size transactions (less than 100Kbyte records sizes. For records of mean size 100KByte-700Kbyte), MongoDB performs much better than MySQL. In conclusion, the select experiment shows that the MySQL database

performance is worse than MongoDB. The results on the average time show that select all records query and select 10000 records query on a modest size database (100,000 records).
Research that has been carried out by Fiannaca [4], throughput between the MongoDB and PostgreSQL databases has been evaluated in the matter to decide which is the best database store for a future application embedded in the current Robot Operation System (ROS system) [12]. PostgreSQL performed importantly worse than MongoDB and results are shown at Table 2. This is not especially concerning since MongoDB is made for handling JSON data while PostgreSQL is designed to manage relational data only with extensions for JSON document data. The transformations from relational data to JSON document data are time consuming when referred to performance.

4. EXPERIMENTS AND RESULTS ON IOT DATA

Performance measurements have been conducted by the authors of this paper between relational databases (MySQL 5.6.3 and PostgreSQL 9.6) and NoSQL (MongoDB 2.6.10) database. For the purpose of this paper, 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 this configuration, because 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).

The experimental database server performed locally using Python scripts because authors wanted to minimize network delays and jitter. The amount of concurrent database connections is set to 2,000 for MySQL, PostgreSQL and for MongoDB. 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 terms of other services. The OS system and services use up to 500-700MB of resident memory; during experimentation, the file memory mappings of MongoDB did not exceed at all the 1.3GB of memory RAM.

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

NoSQL database has been evaluated using MongoDB stored data coming from an IoT agricultural service. A collection of documents consists of 7 moisture sensors, a temperature sensor and a servo valve actuator status (on|off decision). Sensors-actuator systems have been 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. Each of the experiment has been performed 10 times. The average response time query values have been calculated as well.

4.1. Performance Evaluation Metrics and Measures

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

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

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).

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

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:

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

where the sums $\sum R_1^{insert|update|}, \sum R_2^{insert|update|}$ are the number of records returned from queries 1 and 2 respectfully 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) and is calculated using a zero result query time estimate.

4.2. Experimental Scenario 1, Select Queries Experimentation on IoT Data

The first scenario of the authors was to evaluate the select-find queries, since the IoT applications or agents use usually this type of queries, to interrogate the databases and acquire records for further evaluation. A fixed number of records are being returned for the purpose of this experiment which measures the queries execution time amongst MySQL, PostgreSQL and MongoDB.

The total execution time up to 500.000 (500K) returned records in the IoT database is being presented in the Figure 5 below. The returned records average data size can be estimated to 64Bytes multiplied by the value of x axis number of returned query records.

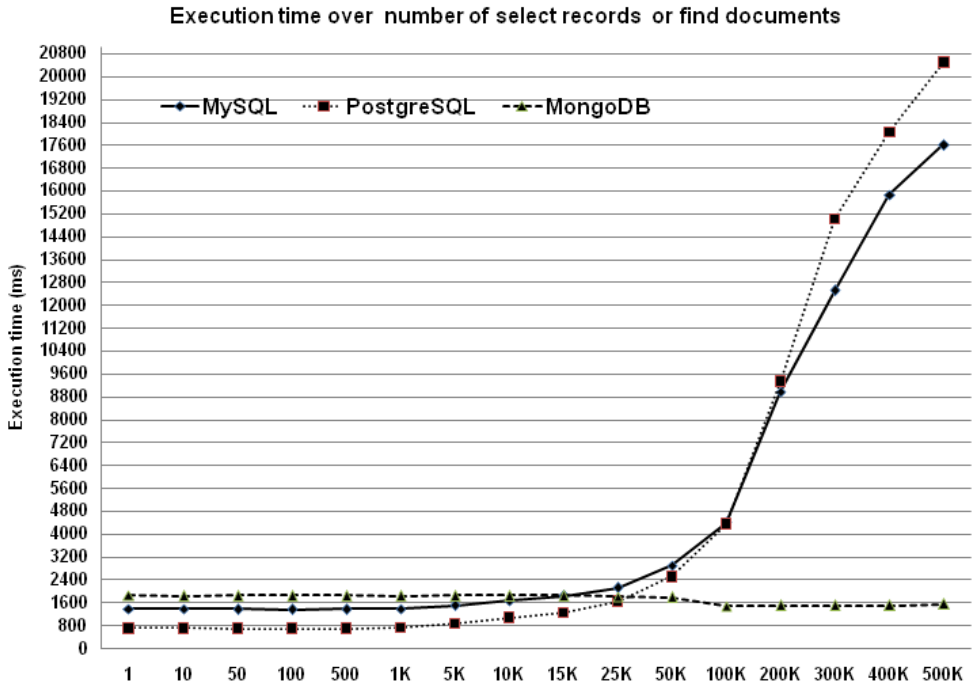

Figure 5. Total Execution time in IoT databases over number of records

For small record sizes of up to 100,000 IoT records returned by a select-find query, which eis equal to data transfer of up to 6.1MByte, PostgreSQL is much more efficient than MySQL from 48% for one returned record query down to 0.08% for a 100K records query. PostgreSQL outperforms for small (<100K records) number of records returned from an IoT database presenting an average of 36.5% more throughput than MySQL (see Table 3, 1-100,000 records). For big data transactions (above 7MB of returned data >100,000), MySQL performs better than PostgreSQL at an average of 18% based on execution time. In respect to throughput, MySQL, for big data transactions outperforms 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 Table 3).

For big data queries, PostgreSQL performance has been evaluated to be similar to MySQL one with the restriction that the queries in a transaction are clustered to small returned record queries executed back to back. In such cases PostgreSQL shows 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).

For small queries (up to query size of 1.52 MB per transaction -25,000 records) MongoDB performance is 51% worse than PostgreSQL and 20% (on average) MySQL. Regarding execution time all MongoDB measurements keep pacing similarly depending on 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 performs better than MySQL as regards

throughput by 69% on average for returned records above 20,000 and performs better by 72% on average PostgreSQL for returned data records above 30,000 (see Table 3).

Table 3. Measurements of Throughput over query number of records

	Throughput (QPS KB/s)			% Throughput - QPS Performance		
Records per query	**MySQL**	**PostgreSQL**	**MongoDB**	**PostgreSQ Lover MySQL**	**MongoDB over MySQL**	**MongoDB over PostgreSQL**
10	0.44	0.87	0.33	48.62	-24.84	-61.38
50	2.25	4.46	1.68	49.40	-25.50	-62.30
100	4.54	8.92	3.36	49.13	-25.92	-62.31
500	22.58	43.68	16.84	48.30	-25.41	-61.44
1000	44.70	85.45	33.86	47.69	-24.24	-60.37
5000	207.87	360.39	168.44	42.32	-18.97	-53.26
10000	371.95	574.82	335.94	35.29	-9.68	-41.56
15000	510.21	749.93	503.58	31.96	-1.30	-32.85
25000	731.39	950.68	857.55	23.07	14.71	-9.80
50000	1071.60	1238.35	1739.25	13.47	38.39	28.80
100000	1417.95	1435.03	4173.12	1.19	66.02	65.61
200000	1391.43	1339.65	8325.11	-3.72	83.29	83.91
300000	1498.54	1248.98	12428.08	-16.65	87.94	89.95
400000	1576.46	1385.68	16462.96	-12.10	90.42	91.58
500000	1774.68	1524.49	20181.99	-14.10	91.21	92.45

According to the table 3, the throughput and %throughput comparative results of MySQL, PostgreSQL and MongoDB have been shown. PostgreSQL is the best database system for up to medium sized IoT select queries, while performed better than the rest of the databases. MongoDB keeps a stable execution time performance. For big data transfers and for the relational databases, MySQL performs better than PostgreSQL. Nonetheless, MongoDB importantly performed better than MySQL. At last, for medium size transactions (from 25.000-100.000 returned records, which applies to an average of 3MB of total data transfers), MongoDB followed by PostgreSQL manage to keep lower throughput results.

4.3. Experimental Scenario 2, Insert Queries Experimentation on IoT Data

In this case, authors perform a number of insert queries within a transaction. Also, they measure the total transaction execution time as well as queries jitter (based on Equation 3). Queries jitter emphasizes the queries execution consistency in terms of execution time. The records used are of 128Bytes size, formatted as JSON strings, which is the maximum data size used by IoT applications. Regarding this scenario authors examine how many IoT devices can continuously transfer data to the database system. The databases used for examining the burst insert queries have set a dataset of 500,000 records of IoT data. In order to simulate concurrent IoT insert queries, within a transaction, the queries are back to back executed, using delay intervals of 2ms. Figure 6 shows the results of MySQL, PostgreSQL and MongoDB.

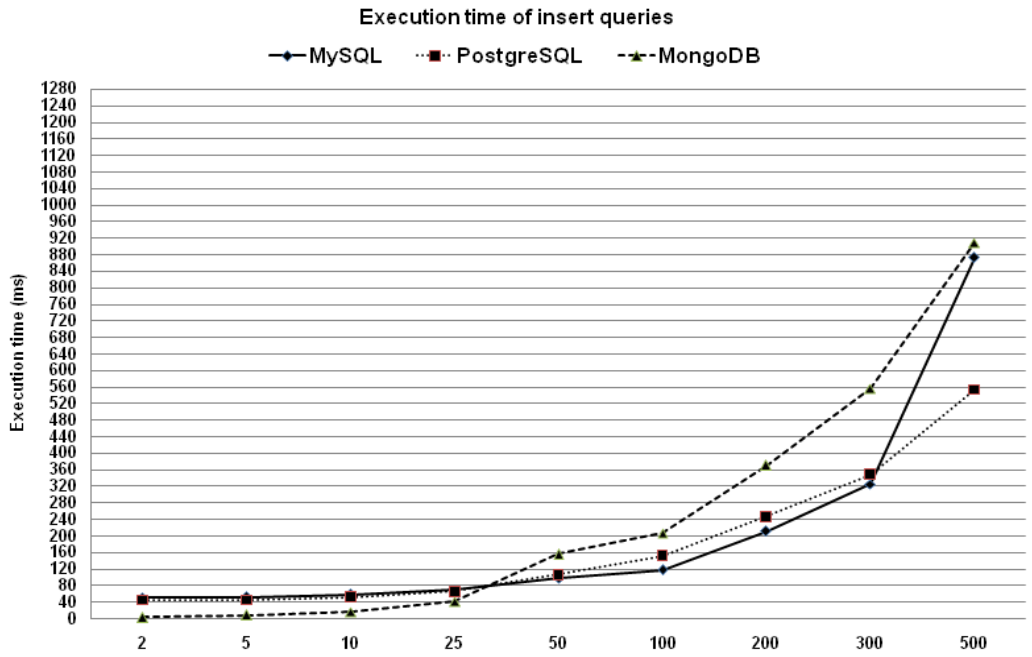

Figure 6. Evaluation of Insert Queries execution time in IoT databases. In the x-axis is the number of queries within an insert transaction.

According to the results, MongoDB shows the best execution time, 68% in average less time than PostgreSQL and 72% in average less time than MySQL for burst insert queries up to 30 queries per transaction. For burst insert queries up to 50 queries per transaction MySQL and PostgreSQL perform the same as expressed by execution with less time than MongoDB.

For transactions that include more than 30 insert queries, MongoDB performs poorly, starting with a 3.6-3.9% more execution time than PostgreSQL and MySQL respectfully, reaching a maximum of 72% in comparison to PostgreSQL at 500 insert queries and 75% compared with MySQL at 300 insert queries. The average execution time is shown at Table 4, divided into: Small number of insert queries (up to 10 queries), medium number of insert queries (from 10 up to 100 queries) and big number of insert queries (from 100 up to 500 queries). Table 4 shows the average jitter (from Equation 3) that corresponds to the three types of insert transactions (low, medium, big). MySQL outperforms all the rest two of them for above 50 insert queries and up to 350 insert queries which are 15% better than PostgreSQL regarding average execution time. From 350 insert queries and above MySQL performance decreases radically reaching close to that of MongoDB, with the exception of big number of inserted records (above 300 insert queries), where PostgreSQL shows the least execution time close to 98% much better than the execution time of MySQL and MongoDB.

Concluding, the results presented at Table 3, MongoDB outperform for transactions of small numbers of insert queries, followed by PostgreSQL. For medium number of insert queries MySQL outperforms followed by PostgreSQL and for big number of queries PostgreSQL outperforms followed by MySQL. Furthermore, for very big number of inserted queries (1,000-2,000 inserted queries), MongoDB presents the best performance out of MySQL in an 7% average execution time, while PostgreSQL performs better than MongoDB in a 50-80% average execution time.

Table 4. Measurements of total transaction execution time and transaction jitter on insert queries

	Average Execution time (ms)			**Average |Tj| scalability (ms)**		
Queries per transaction	MySQL	PostgreSQL	MongoDB	MySQL	PostgreSQL	MongoDB
Small number of insert queries [2..10]	53.89	47.38	9.82	0.12	1.15	0.17
Medium number of insert records (10, 100]	95.43	108.27	123.47	0.06	0.39	1.69
Big number of insert records (100,500]	470.25	383.22	612.56	6.40	0.17	1.9

Keeping a low performance profile is an indication of strong scale service. PostgreSQL does that on almost constant queries jitter values, followed by MySQL and MongoDB. This indicates better scalability capabilities of PostgreSQL. MongoDB jitter values are similar to PostgreSQL values. In spite of low performance for medium and big number queries, results show it is a fair scalable service. MySQL shows that the worst scalability attributes because of the sudden increment of queries jitter values for big number of inserted records.

4.4. Experimental scenario 3, aggregation functions experimentation on IoT data

In this case, authors perform a select query over a constant number of records. Each time the embedded MAX aggregation function of PostgreSQL, MySQL and MongoDB is called. The scenario is performed using a transaction of 10 MAX queries and the transaction total execution time. The queries jitter time (derived from Equation 3) is being measured and the results are shown at Table 5.

According to the Table 5, it is clear that for small record sizes PostgreSQL aggregation function execution time performs better, followed by MySQL and then MongoDB. MongoDB aggregation function measurement rates are stagnant compared with relational databases. Nevertheless, for medium record sizes MySQL and PostgreSQL do not perform very fast leaving space for MongoDB to perform faster and better. For big record sizes MySQL is better than the PostgreSQL and MongoDB.

Regarding databases scalability as expressed by transaction jitter (Equation 3), MongoDB shows important jitter for small number of queries. Also, it should be mentioned that it fails to keep a low jitter profile for both medium and big queries. This is an indication that MongoDB stored procedures should be better executed on records found on a single database rather than a distributed one. PostgreSQL keeps the lowest jitter profile for small number of queries. This is an indication that PostgreSQL can be distributed only if the applications perform internal aggregations on small clustered data chunks. MySQL shows the least jitter profile for both medium and big record sizes aggregations, which indicates that its internal engine for procedural execution is the most suitable for clustered databases.

Table 5. Measurements of total transaction execution time and transaction jitter on MAX stored procedure on a float field, over a number of records.

	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

*1 low record sizes, *2 medium record sizes, *3 big record sizes

5. CONCLUSIONS

Authors evaluate the performance between open source relational databases and NoSQL databases. Relational databases disadvantages relay on the unease design, normalization forms and types for IoT services, their limitations on maximum storage records, and their corruption are procumbent to big data that mostly requires the use of special type. In this case, successfully repair software is not always a solution.

NoSQL databases are new and become popular especially designed for IoT, as they provide horizontal schema-less collections, tremendously useful for IoT data originated from different sources of different structure, sensory hardware and transmission protocols. The relational databases tested through this paper are MySQL and PostgreSQL, as well as the MongoDB non-relational database. At first a short literature review has been performed focusing on database IoT capabilities and BLOB data storage evaluation. Then Experimental scenarios took place using IoT sensory data in three different experimental cases: 1) IoT data insertion time, 2) IoT agent select queries execution time and 3) IoT agent database aggregation function execution time.

According to the authors' experiments and results, for small number of selected records PostgreSQL outperforms MySQL and MongoDB. MongoDB performs better in respect to MySQL and PostgreSQL for big number of selected records. MySQL outperforms better than PostgreSQL for big number of selected records (>20000) but still cannot perform better than MongoDB.

For insert queries and small amount of IoT records, MongoDB outperforms MySQL and PostgreSQL, whereas for big number of records PostgreSQL presents the least execution time in comparison to MySQL and MongoDB.

Aggregation functions execution experiments has shown that PostgreSQL is the most suitable database system for performing aggregation functions on a small number of IoT data records. On the opposite edge, for an aggregation function applied on a big number of IoT records, MySQL presents the best performance results in terms of execution time. MongoDB is not a good option for aggregation functions execution on IoT data.

REFERENCES

[1] Aboutorabi S., Rezapour M., Moradi, M., and Ghadiri, N. (2015), "Performance evaluation of SQL and MongoDB databases for big e-commerce data ", In proc. of CSICSSE conf., DOI: 10.1109/CSICSSE.2015.7369245

[2] Damodaran D. B., Salim S. and Vargese M. V. (2016), "Performance evaluation of MySQL and MongoDB databases ", International Journal of Cybernetics & Informatics IJCI, Vol. 5, No. 2, ISSN:2320-8430

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

[4] 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]

[5] 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]

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

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

[8] PostgreSQL (1996)., "PostgreSQL: The World's Most Advanced Open Source Relational Database", Internet: https://www.PostgreSQL.org [Apr. 2010]

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

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

[11] 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

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

[13] Stancu-Mara, S., and Baumann, P. (2008)., "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

[14] Sullivan P. (2012), "Comparing PostgreSQL 9.1 vs MySQL 5.6 using Drupal 7.x." Internet: http://posulliv.github.io/2012/06/29/mysql-postgres-bench/ , [Feb, 2017]

AUTHORS

Christodoulos Asiminidis graduated from the University of Ioannina, Department of Mathematics. He received his Bachelor of Science in Mathematics specialized in computing science. During his university studies, he managed to carry out three projects through university internships. The first project focused on application development, the second one on building machine learning systems and the last one on web developing. He is currently doing his master's degree in Data Mechanics and Computational Systems in the Department of Computer Science and Engineering of the University of Ioannina. His e-mail is: chasiminidis at cs.uoi.gr.

George Kokkonis received his PhD diploma from the Dept. of Applied Informatics, University of Macedonia, Greece. He received a five-year Eng. Diploma from the Dept. of Electrical and Computer Engineering, Aristotle University of Thessaloniki and his MSc in Information Systems from the University of Macedonia. He has been working as a Lecturer since 2006 at the Dept. of Computer Applications in Management and Economics, Western Macedonia University of Applied Studies, Greece. He has been teaching Multimedia Systems, Pc Programming and Data Bases. Since 2009 is responsible of the Network Operations Center (NOC) of the branch of the Western Macedonia University of Applied Studies in the city of Grevena, Greece. He is currently a post doctoral researcher in the University of Macedonia in the fields of IoT and supermedia communications. He is also a researcher at the laboratory of Renewable Energy Sources of the Western Macedonia University of Applied Studies, Greece. He has several publications in international Conferences, books chapters and peer reviewed journals. His professional interests are: Multimodal Data Communications Systems, Haptic Communication between Humans and Robots, Future Media- Internet of Things.

Sotirios Kontogiannis graduated from Democritus University of Thrace, Department of Electrical and Computer Engineering. He received an MSc in Software Engineering and Ph.D. from the same department, in the research area of algorithms and network protocols for distributed systems. He worked as a software developer for more than ten years in the private sector and participated into SME research and development projects. He also worked as an adjoint assistant professor at the Dept. of Business Administration, Technological Educational Institution of Western Macedonia, for six years and for two years as a contract lecturer at the Dept. of Informatics & Telecommunications Eng., University of Western Macedonia. His research interests focus on the areas of distributed systems, artificial intelligence, AI algorithms, sensor networks, middleware protocols and computer networks. He is currently a scientific staff member and director of the Distributed micro-computers laboratory (http://kalipso.math.uoi.gr/microlab), at the Applied Mathematics and Engineering research section of the Department of Mathematics, University of Ioannina. His personal web-page is at http://spooky.math.uoi.gr/~skontog and his e-mail is: skontog at cc.uoi.gr.