COMPARATIVE PERFORMANCE ANALYSIS BETWEEN MYSQL AND MONGODB DATA STORAGE ENGINES TO SUPPORT DYNAMIC CONTENT OBJECTS

Análisis comparativo del rendimiento entre los motores de almacenamiento de datos MYSQL y MONGODB para soportar objetos de contenido dinámico

ROGER CALDERÓN MORENO[1], EDWIN DAVID RUBIANO BACCA[2], LUIS ALBERTO PARRA LINARES[3]

Recibido: 2 de junio de 2024 Aceptado: 5 de julio de 2024

DOI: https://doi.org/10.21017/rimci.1093

Abstract

Comparative performance analyses enable users to take informed decisions on technologies beyond current market trends and/or commercial information from vendors. In this paper we present a comparative performance analysis between MySQL and MongoDB, based on data from the *Dynamic Survey Manager* at the *Planes de Energización Rural Sostenible, Región Orinoquía* project. First, we proposed a document-based data storage model to support the current relational model. This model allows a flexible structure and storage of information for the organization in future. Then, we configured the document-based model on MongoDB. Subsequently, we measured and compared performance times by using selected test scenarios. Finally, we found MongoDB has at least 40% better response times, in addition to the flexibility in the information structure and storage with respect to MySQL. MongoDB's flexibility allows software developers to skip the object relational mapping within their data persistence layer, while supporting ACID-type transactions (Atomicity, Consistency, Isolation, Durability). Despite the positive results, we found a test scenario where MySQL outperformed MongoDB. For queries involving larger objects (e.g., > 100MB), MongoDB was 7.5% slower than its counterpart.

Key words: MongoDB; MySQL; BSON; JSON; performance analysis.

Resumen

Los análisis comparativos de desempeño permiten a los usuarios tomar decisiones informadas sobre tecnologías más allá de las tendencias actuales del mercado y/o la información comercial de los proveedores. En este artículo presentamos un análisis comparativo de desempeño entre MySQL y MongoDB, basado en datos del Dynamic Survey Manager en el proyecto Planes de Energización Rural Sostenible, Región Orinoquía. Primero, propusimos un modelo de almacenamiento de datos basado en documentos para soportar el modelo relacional actual. Este modelo permite una estructura y almacenamiento flexible de información para la organización en el futuro. Luego, configuramos el modelo basado en documentos en MongoDB. Posteriormente, medimos y comparamos los tiempos de desempeño utilizando escenarios de prueba seleccionados. Finalmente, encontramos que MongoDB tiene al menos un 40% mejores tiempos de respuesta, además de la flexibilidad en la estructura y almacenamiento de información con respecto a MySQL. La flexibilidad de MongoDB permite a los desarrolladores de software omitir el mapeo relacional de objetos dentro de su capa de persistencia de datos, al mismo tiempo que admite transacciones de tipo ACID (atomicidad, consistencia, aislamiento, durabilidad). A pesar de los resultados positivos, encontramos un escenario de prueba donde MySQL superó a MongoDB. En el caso de consultas que involucran objetos más grandes (p. ej., > 100 MB), MongoDB fue un 7,5 % más lento que su contraparte.

Palabras clave: MongoDB, MySQL, BSON, JSON, análisis de rendimiento.

1 Ingeniero de Sistemas, Especialista en Ingeniería de Software, Magister en Software Libre. Profesor Asistente Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos. Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos. Villavicencio, Colombia. ORCID: https://orcid.org/0000-0001-5923-8601 Correo electrónico: rcalderonmoreno@unillanos.edu.co

2 Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos. Villavicencio, Colombia. ORCID: https://orcid.org/0009-0000-0713-0578 Correo electrónico: edwin.rubiano@unillanos.edu.co

3 Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos. Villavicencio, Colombia. ORCID: https://orcid.org/0000-0003-1146-1841 Correo electrónico: luis.parra.linares@unillanos.edu.co

I. Introduction

In the frame of the research project *Planes de Energización Rural Sostenible, Región Orinoquía*[1], the Observatory of the Territory of the University of the Llanos generated an information system called *Dynamic Survey Manager* to collect information through complex surveys on its structure and size, which can change over time without affecting the data already collected. Analysts designed 32 survey forms, which were applied by users in the departments of Arauca, Casanare, Meta and Vichada through a mobile application that allowed them to send data to the server and update their survey forms without having to make additional software installations. To store the information during the first phase of project execution, the *Dynamic Survey Manager* used MySQL as storage manager, where 5929 surveys with an approximate total weight of 1.0 GB were stored. In addition, we evidenced that to store information in MySQL it was necessary to condition the way in which the information is treated on the side of the client application. Traditionally tables represents this information as the disaggregation of composite objects[2], in addition to the validations and integrity rules imposed in the relational model; the information transformation occurs through traditional coding or with the support of the *Object Relational Mapping* (ORM).

One of the requirements of the *Dynamic Survey Manager* asked for time variable information structure, in terms of data types and organization. This requirement involved more complex designs — one could say, a small version control system for the relational data model. In addition, to support the changes that occurred when adding new data types it was introduced data redundancy, all of the above without losing the functionalities implemented already. These situations represent a potential risk in the software development process as suggested by[3]. Another difficulty was the latency for the delivery of queries, degrading the performance of the applications to generate the reports and causing dissatisfaction among the user community[4].

The situations previously expressed in the implementation of the *Dynamic Survey Manager* in the first phase of the project, allowed to raise the need to look for an alternative to store the information, for which a document-oriented NoSQL type database was proposed[5], specifically MongoDB[6], since when reviewing the ranking of documentary type databases it is observed that it has great acceptance and remains the most popular[7]. To evaluate the performance of the current and new database engine, we proposed multiple scenarios to measure the response time of each one. Reviewing related studies, we found several documents aiming to define some kind of performance comparison between the MySQL and MongoDB. These studies highlight different scenarios, for example some scenarios are based on the sending of data through local media or through the Internet, others focus on insertion operations only and data query, and those that verify all basic CRUD-type (Create, Read, Update, and Delete) operations. Besides of the executed operations, other studies repeat several times the measurement process to obtain better results[8]-[11]. In addition, other studies make use of different data types, however it is not evident the data size, which suggests that data is about few bytes. In another study[12] researchers compared the response times between MySQL and MongoDB, using standardized electronic health records with medical information. These records support the ISO/EN 13606 standard and included scenarios of 5000, 10000 and 20000 records.

Despite the abundance of related research we set apart by focusing on dynamic content objects execution and query rather than the computational complexity of querying relational and non-relational records. Also and for the proposed testing process, we define the size of the information contained by the objects that store information as a critical design factor. It is intended that the performance comparison from this study support the case for the technology migration of the *Dynamic Survey Manager*.

II. Methodology

2.1 Storage Engines

2.1.1 MySQL

MySQL is an open-source relational database manager released under *General Public License* (GPL), which is developed, distributed and

supported by Oracle Corporation. The structure definition that organizes the information and its respective storage is done through tables or entities that are composed by rows or tuples. In addition, fields form these entities with support for different data types[13]. MySQL works under different storage mechanisms such as: InnoDB, MyISAM, Memory, Federated, Merge and Archivie[14].

For the development of the *Dynamic Survey Manager*, they used the InnoDB storage mechanism from its inception. InnoDB allows applying relationships between tables and defining integrity restrictions. Also this mechanism supports the execution of ACID-type transactions. In addition, MySQL's SQL standard supports the information query language, the data manipulation language (DML) and the data definition language (DDL).

2.1.2 MongoDB

MongoDB is a document-oriented, open-source, cross-platform NoSQL database manager, released under *GNU AGPL* v3.0 license. It is distributed and owned by MongoDB Inc[15]. MongoDB organizes information through the definition of documents that may have a variable structure in their content. The structure of the documents is like JSON-type documents; MongoDB stores data as documents in a binary representation called BSON (Binary JSON), which extends the JSON model by generating additional data types, sorting fields and improving the coding and decoding of data for various programming languages ??[16]. Grouping information under a dynamic schema in JSON format allows software developers to generate schema relationships based on the objects defined in the class models and the application objects. In this way, developers send complete composite objects to storage and later query, without the need to disaggregate the information.

In contrast, relational databases lack this feature. Using MongoDB removes the complex layer of the ORM that translates objects from code into relational tables.

At present, there is a need to manage massive flows of data in a variety of formats: structured, semi-structured and polymorphic. Web, mobile, social and Internet applications produce data and provide the perfect venue to transform data into knowledge by developing software applications. MongoDB document data model is naturally assigned to the objects in the application's source code, which simplifies its learning and use for developers. In addition, MongoDB supports ACID-type transactions since version 4.0[17].

2.2.Test Scenario Definition

In similar studies, tests use various scenarios ranging from local tests through the execution of code by console, execution from web applications with PHP[18], to mobile applications as evidenced in[19]. For our case, we sent data from a mobile application developed with *Xamarin Forms*.

In the testing process, it is important to characterize the input. In Table I we show the size of the information contained by the objects that store information from the surveys. The larger the object the more complex storage structure is. We proposed to calculate the execution time of both insertion and query scenarios.

To perform the tests, we decided to use the structure of the object that has all the survey information as data for insertion and query operations. This object resides in the client application, for our case a mobile application. Out of the 32 available survey type forms, we selected the two most complex ones: *Residential Form* and *Economic Form*. Each form gave data matrices of 701400 and 136200 values respectively. Table 1 summarizes the selected instruments in detail.

We proposed three scenarios to measure running times during the test execution in MySQL and MongoDB: 1. Execution, insertion and query with 30 records; 2. Execution, insertion and query with 300 records; and 3. Execution, insertion and query with 600 records. We executed the tree scenarios for the prioritized survey forms (*Residential* and *Economic*). In each scenario, we took the average time as a measure for each storage engine. Finally, we used this time to do the analysis and draw conclusions.

Also, we defined five search criteria for each type of survey and obtained the response time in the two storage engines to the insertion and query processing. The criteria are indicated below:

Economic: 1. Query geographical coordinates, respondent name, type of housing, number of residents of the house for each survey; 2. Query the surveys where the house has electricity service by public network (interconnection), sewerage and municipal or rural aqueduct; 3. Obtain the surveys where there is consumption of firewood or charcoal in the house, and query the cost of charcoal, firewood purchased and self-appropriate for each; 4. Query amount, name of the respondent, type and power of bulbs specified in each survey where the quantity is greater than zero and sort them ascending order by the quantity field and; 5. Query the equipment that has energy consumption in all the surveys, the quantity of each and order them ascending by the amount.

Residential: 1. Query geographic coordinates, distance in km to the municipal seat and the type of vehicle that you use mainly; 2. Query the surveys where the electric power is obtained connected to the public network (interconnection), the form of ownership of the own property and if they have property titles or registration documents in public instruments; 3. Obtain the road type status used to get from the farm to the municipal seat for each survey; 4. From the surveys that get the electric power by interconnection, consult if they have a meter, how many days a week have electric power service, order by number of days per week of service ascending and; 5. Get the current average cost of one hectare and the current average lease cost of one hectare for all surveys, show the cost of leasing upwards and present the average cost for both cases, excluding costs less than 1000. Table IV shows the times obtained within the query process in the storage engines.

The test execution environment had the following specification: a virtual machine over VirtualBox 6 with Centos 7.0 operating system, with 16 GB of RAM, Intel (R) Xeon (R) CPU E5-2603 v4 @ 1.70 GHz of 6 cores, with a 256 GB hard disk and 10000 Mb / s Ethernet network adapter. Additional installed software was Apache Tomcat 9, MySQL 5.7 under the InnoDB storage mechanism and MongoDB 4.0.6. The mobile device to send the survey was: Samsung Galaxy Tab A 10.1 inches with Android 6 operating system (Marshmallow) and 2 GB RAM. In the process of sending information, the surveys that were already entered in the database were taken and placed again on the mobile device for each scenario, and from there we sent the information through a local network of data (LAN).

III. Results

3.1 Data from execution

In Table II, we present the results for each of the proposed scenarios in relation to the execution and insertion times on the data of the *Economic*-type survey form. During the test, we used an object composed of 34 questions and approximately 227 responses per survey (see Table I).

In Table III, we present the results for each of the proposed scenarios in relation to execution and insertion times on the data of the *Residential*-type survey form. In this test, we used an object composed of 90 questions and approximately 1,169 responses per survey (see Table I).

In Table IV, we present the results for each of the proposed scenarios in relation to the queries defined for each survey form.

3.2 Data Analysis

During the execution of the scenarios to obtain the data of the execution and insertion processes of the *Economic*-type surveys, we observed

Table I. Selected instruments for the performance test.

Instruments – Survey Type Form	Number of questions per survey	Number of possible answers per survey	Number of surveys	Min. Survey size (Bytes)	Max. Survey size (Bytes)	Average Survey size (Bytes)	Average Survey size (kB)
Residential	90	1169	600	158783	207462	183123	179
Economic	34	227	600	20707	49340	35024	34

Table II. Execution and insertion time for *economic*-type survey.

Samples	Execution time (s)		Insertion time (s)	
	MySQL	MongoDB	MySQL	MongoDB
30	0.79	0.34	0.013	0.0058
300	3.5	1.52	0.012	0.0051
600	6.44	2.72	0.011	0.0045

Table III. Execution and insertion time for *residential*-type survey.

Samples	Execution time (s)		Insertion time (s)	
	MySQL	MongoDB	MySQL	Mongo DB
30	2.55	1.42	0.043	0.023
300	11.8	6.68	0.039	0.022
600	23.15	12.33	0.038	0.02

Table IV. Query times.

Query	Economic-type Survey time (s)		Residential-type Survey time (s)	
	MySQL	MongoDB	MySQL	MongoDB
Query 1	1.92	1.28	5.63	3.72
Query 2	2.03	1.13	5.48	2.64
Query 3	2.16	1.61	5.47	3.64
Query 4	2.07	1.29	5.46	3.77
Query 5	2.27	1.42	5.75	6.18

significant differences in the two selected engines. MongoDB showed better times. In Fig. 1 we show the comparative results between the two storage engines. This result is consistent with the validation process of primary and foreign keys that are included in the table in which the information is inserted, actions that are not carried out within MongoDB. Hence the speedup of the process compared to MySQL.

Regarding the times obtained in the insertion process, we obtained an average improvement of the three scenarios of up to 57.33%, which can be observed in Fig. 2. The improvement in the time of insertion can also be explained due to the integrity validations reference and the process of storing the information in binary format for the BLOB-type field of MySQL, while in MongoDB the document is not stored in a specific field, on the

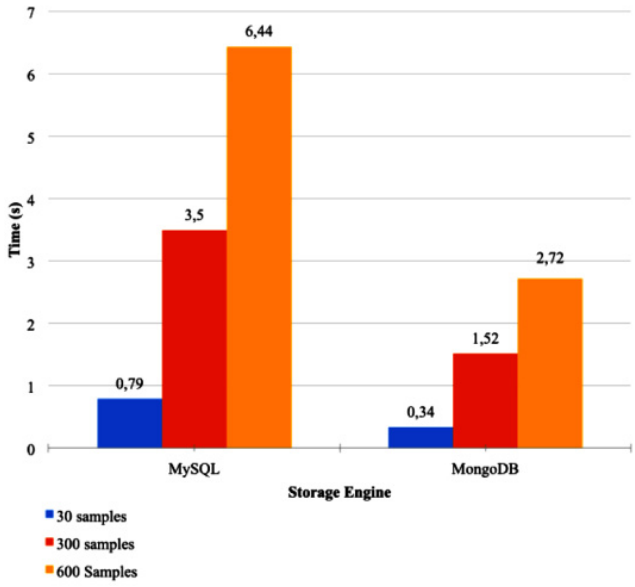

Fig. 1. Execution time comparison between MySQL and MongoDB for 30, 300 and 600 samples – *Economic*-type Survey Form.

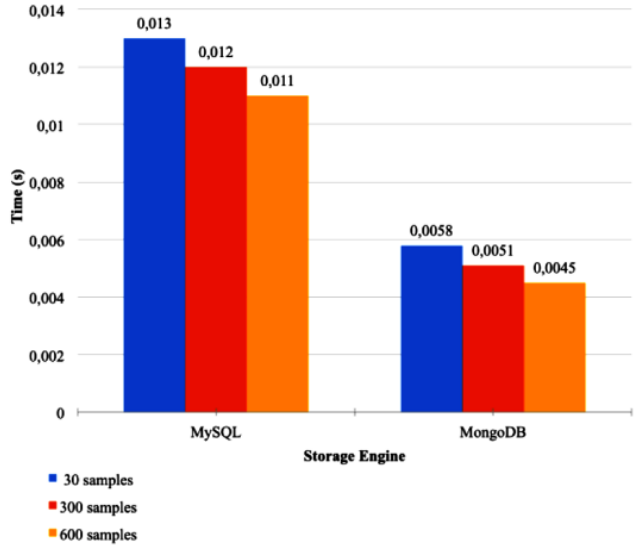

Fig. 2. Insertion time comparison between MySQL and MongoDB for 30, 300 and 600 samples – *Economic*-type Survey Form.

contrary, it is stored as a complete document in hard disk that belongs to a collection in BSON format action that improves the performance with respect to MySQL.

In Fig. 3 and Fig.4, we present the information related to the execution of the scenarios and insertion processes of the *Residential*-type surveys. As one can see, there is an improvement in the performance of operations in MongoDB of up to 45% compared to the performance of MySQL.

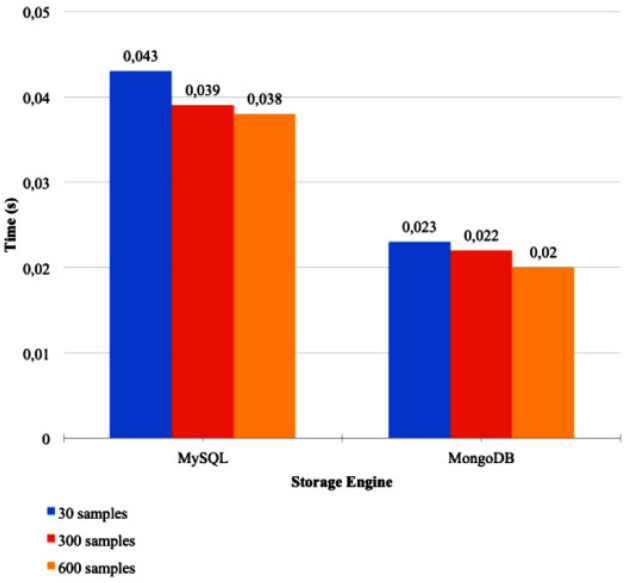

Fig. 3. Execution time comparison between MySQL and MongoDB for 30, 300 and 600 samples – Residential-type Survey Form.

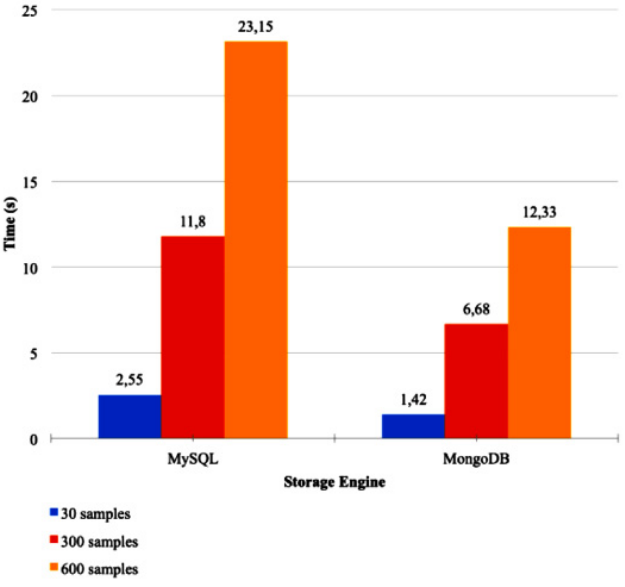

Fig. 4. Insertion time comparison between MySQL and MongoDB for 30, 300 and 600 samples – *Residential*-type Survey Form.

In Fig. 5 we present a comparison of the speedup for both the *Economic*- and *Residential*-type Survey Forms. The speedup definition is as follows: speedup= (exe_time_MySQL - exe_time_MongoDB)/exe_time_MySQL. We did the same calculation for the results of each sample size. As result, we observed a speed up above 50% on average for the *Economic*-type Survey Form and a speed up above 40% on average for the *Residential*-type Survey Form.

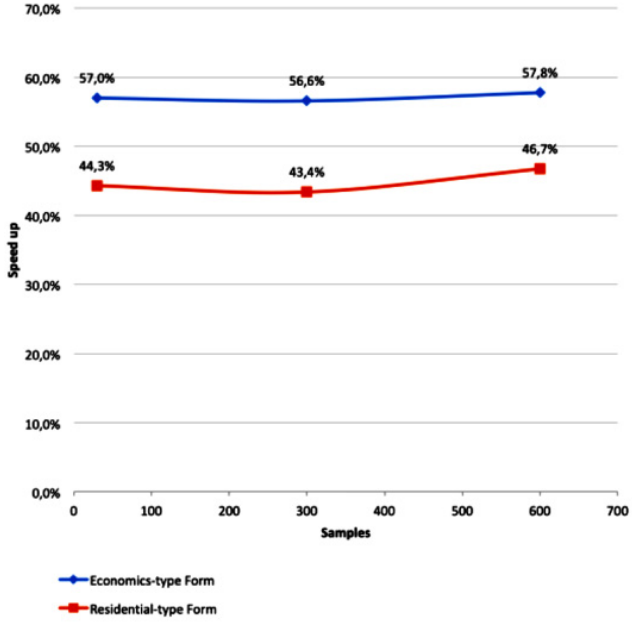

Fig. 5. Speedup comparison for the execution times in both types of Survey Forms.

In the process of executing the scenarios to obtain the data of the query times related to the *Economic*-type Survey form, MongoDB showed an improvement of the times of approximately 35.2% with respect to MySQL (see Fig. 6).

Regarding the time obtained for the *Residential*-type Survey Form, we evidenced an improvement of 37.4%, particularly in queries 1, 2, 3 and 4 (see Fig. 7). These improvements can be explained by the efficiency of MongoDB's Query language. On the other hand, in query 5 of the *Residential*-type Survey Form, we observed an improvement of the performance of MySQL with respect to MongoDB, a situation that can be explained by the large amount of data that was generated in the process of grouping and ordering operations, which are made in main memory and exceeds 100 MB. This situation generates an error in MongoDB. To solve this error we used the function *allowDiskUse*[6], this function is activated when the data to group exceeds 100 MB of main memory, and is supported by a temporary directory on the hard disk to perform such operations. Accessing a secondary storage medium generates longer response times in queries, actions that are also presented in MySQL, which in this case offers better response times with respect to MongoDB. The previous difficulty was also evidenced in the study carried

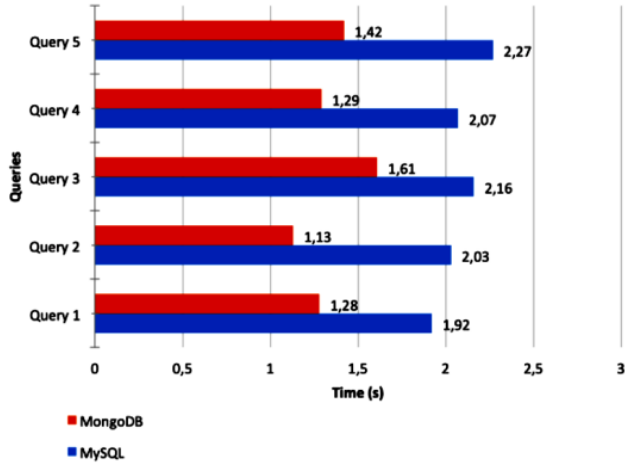

Fig. 6. Query Time Comparison – *Economics*-type Survey Form.

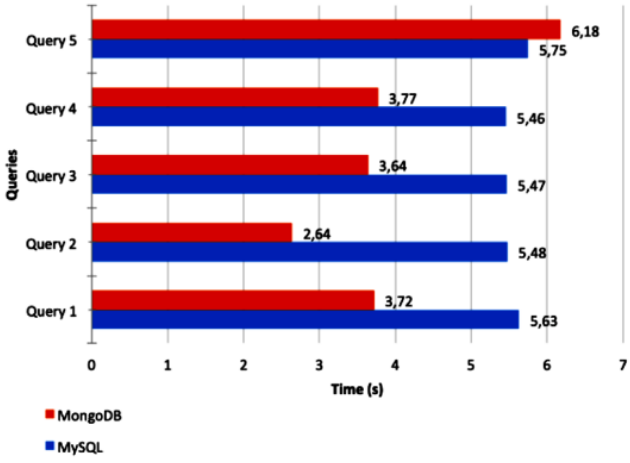

Fig. 7. Query Time Comparison – *Residential*-type Survey Form.

out by[12]. It is important to remember that in all three scenarios, we are searching for files with an average size of 179 KB, where each file contains 90 questions and approximately 1169 survey responses (see Table I).

In Fig. 8 we present a comparison of the speedup for each query in both the *Economic*- and *Residential*-type Survey Forms. The speedup definition is as follows: speedup = (query_time_ MySQL - query_time_MongoDB) / query_time_ MySQL. In general, there is a gain in the processing times for most of the queries executed in MongoDB. The exception comes in Query 5 for the *Residential*-type Survey Form. In this case we observed a negative performance (-7.5%) of MongoDB in comparison to the same query ran in MySQL.

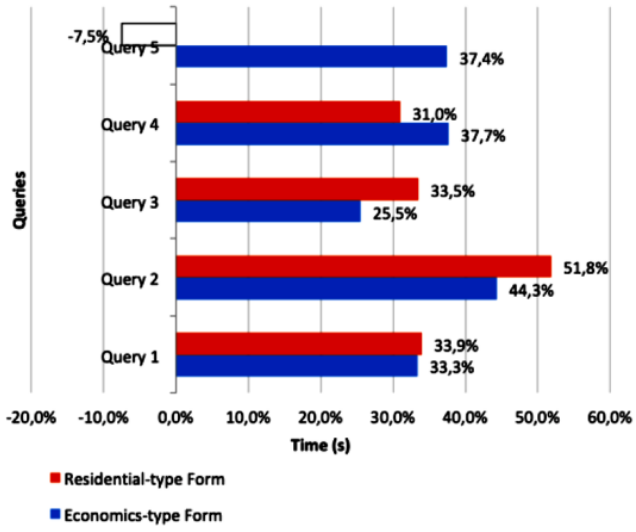

Fig. 8. Speedup Query Time Comparison for *Residential*- and *Economic*-type Survey Form.

IV. Conclusions

Dynamic data model implementation on documentary databases[20] such as MongoDB allows software developers to store and query directly without performing additional tasks to disaggregate information. This fact benefits software developers by avoiding the use of ORM, that is, developers can remove an additional software layer within the persistence layer of the applications' data.

In the processes of execution, insertion and query of dynamic information on MongoDB, we evidenced an improvement in the response times for these three criteria in the three test scenarios. In addition, the processing times between the two engines are very similar for queries that require large amounts of data (>100 MB) on grouping and ordering operations. This result is consistent with the observation by[12] and[21].

MongoDB database is an alternative for information storage, not only for its flexibility to store various information structures, but also, it offers support for ACID-type transactions since version 4.0. The lack of this feature was considered by many to be a disadvantage compared to relational-type engines.

It is evident that MongoDB is a stable and robust storage engine, and it can be proposed for future software development projects. It allows flexibility

in the structure of storage models at both logical and physical levels. This feature will diminish the negative effects due to the software evolution that have the applications in general. It is a well-known fact that software developers must mitigate these risks as suggested by[3].

Acknowledgment

This work was partially funded by the General Directorate of Research of the Universidad de los Llanos with the project C02F01006-2016. The authors also thank the Observatory of the Territory of the Universidad de los Llanos for their collaboration, advice and logistical support that allowed the development of this research.

References

[1] Observatorio del Territorio, Planes de energización rural sostenible. 2019. http://observatorio.unillanos.edu.co/pers/.

[2] R. Calderón-Moreno & D. Arenas-Seleey, Mapeo de objetos a través de un motor de datos NOSQL, caso de estudio: framework para desarrollo de aplicaciones web, Ingeniería Investigación Y Desarrollo, págs. 61-71. 2017.

[3] S. Roger, Ingeniería de Software un enfoque práctico. Pressman, Mexico DF: McGraw Hill, 2006. p. 6-16.

[4] M. Callejas Cuervo, D. I. Peñalosa Parra y A. C. Alarcón Aldana, Evaluación y análisis de rendimiento de los frameworks de persistencia Hibernate y Eclipselink. Manizales: 2011, Ventana Informática, Vol. 24, págs. 9-23.

[5] M. Fowler, Martin y Sadalage, J. Pramodkumar, NoSQL Distilled, A Brief Guide to the Emerging World of Polyglote. s.l: Addison Wesley, 2013.

[6] MongoDB-Inc. Definition. db.collection.aggregate (pipeline, options). 2018. https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/index.html.

[7] Solid IT. DB-Engines Ranking of Document Stores. Document Stores. 2019. https://db-engines.com/en/ranking/document+store.

[8] A Comparative Study: MongoDB vs MySQL. Soni, Sushil y Ambavane, Mayuresh. 2017, International Journal of Scientific & Engineering Research, Vol. 8, págs. 120-123.

[9] A Comparative Study: MongoDB vs. MySQL, Proceedings of the International Conference on Information Technologies. OLAH, Andrada. Bulgaria: 2018, Proceedings of the International Conference on Information Technologies (InfoTech-2018), p. 20-21.

[10] Comparative analysis of NoSQL (MongoDB) with MySQL Database. Kumar, Lokesh. 2015, International Journal of modern Trends in Engineering and Research ISSN(Online) 2349-9745. 2015.

[11] F. Arboleda, F.J. Moreno, Una comparación de rendimiento entre ORACLE y MONGODB. 1, 2016, Ciencia e Ingeniería Neogranadina, Vol. 26, págs. 109-129.

[12] R. Sánchez-de-Madariaga, A. Muñoz, A. L. Castro, O. Moreno & M. Pascual, Executing Complexity-Increasing Queries in Relational (MySQL) and NoSQL (MongoDB and EXist) Size-Growing ISO/EN 13606 Standardized EHR Databases. 2018. 133, 19 de 03 de 2018, Journal of visualized experiments: JoVE doi:10.3791/57439.

[13] Oracle. Reference Manual. What is MySQL?. 2017. https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html.

[14] Supported Storage Engines. Alternative Storage Engines. 2017. https://dev.mysql.com/doc/refman/5.5/en/storage-engines.html.

[15] MongoDB-Inc. Documentation. Introduction to MongoDB. 2017. https://docs.mongodb.com/manual/introduction/.

[16] JSON and BSON. 2017. https://www.mongodb.com/json-and-bson?lang=es-es.

[17] Visión de Conjunto. Comparación entre MongoDB y MySQL. 2017. https://www.mongodb.com/compare/mongodb-mysql?lang=es-es.

[18] A Comparative Study Between the Capabilities of MySQl Vs. MongoDB as a Back-End for an Online Platform. Cornelia Gy?rödi y Ioana Andrada Olah. 2016, International Journal of Advanced Computer Science and Applications, págs. 73-78.

[19] M. M. Patil, A. Hanni, C. H. Tejeshwar and P. Patil, A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retriewal operations using a web/android application to explore load balancing — Sharding in MongoDB and its advantages. 2017. International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) doi: 10.1109/I-SMAC.2017.8058365, p. 325-330.

[20] M. Fowler y P. Sadalage, NoSQL Distilled, A Brief Guide to the Emerging World of Polyglote, Addison Wesley, 2013.

[21] Y. Abrahami, Scaling to 100M: MySQL is a Better NoSQL. 2018. https://www.wix.engineering/blog/scaling-to-100m-mysql-is-a-better-nosql.