Analysis of Node.js Application Performance using MongoDB drivers

OMITTED

OMITTED
OMITTED

Abstract. At the last few years, the usage of NoSQL databases has increased, and consequently, the need for integrating with different programming languages. In that way, database drivers provide an API to perform database operations, which may impact on the performance of applications. In this article, we present a comparative study between two main drivers solutions to MongoDB in Node.js, through the evaluation of CRUD tests based on quantitative metrics (time execution, memory consumption, and CPU usage). Our results show which, under quantitative analysis, the MongoClient driver has presented a better performance than Mongoose driver in the considered scenarios, which may imply as the best alternative in the development of Node.js applications.

Keywords: Performance, Node.js application, MongoDB, drivers, NoSQL databases

1 Introduction

At the last few years, the growth of data volume has changed the perspective of how organizations behavior, from simple data recording to potential advantage in competitive markets.

This event, known as Big Data, not only implies in large storage but also perspectives related to variety, velocity, and value [1]. The traditional architecture of relational databases based on ACID (atomicity, consistency, isolation, and durability) properties, which affect the aspects related to availability and efficiency directly in Big Data environments [2]. The non-relational databases (NoSQL) have been proposed oo solve the side-effects, and allowing more structural flexibility, scalability, and support to replication and eventual consistency [3].

In this context of development environments and programming languages, a variety of database drivers aim to support the execution of the internal database operations. However, in many cases, the development of these drivers are very recent and may present defects or limitations, which results in side-effects to the access and manipulation of data [4]. Thus, the usage decision of which NoSQL database and driver may impact on the performance, due to unknown factors previously.

In this work, we conduct a comparative study of performance between MongoClient [1] and Mongoose [2], both solutions of database drivers to MongoDB [3] in Node.js applications. The main difference between them is the predefinition of schema, a factor which is not mandatory in the majority of NoSQL databases, and MongoDB too; but in one of these drivers is required. In that way, this experimental study analyses the impact of each driver at CRUD (create, read, update, and delete) operations.

The choose of MongoDB based on a crescent number of studies in the research community, and also it is the main option of the document-oriented database. In about Node.js, despite recent development, it presents technical viability to implement robust applications. Also, the database system and application lead to uniformization, because both are implemented in JavaScript.

This study is unique in the investigation of effects of performance in different database drivers in Node.js application since the other works have analyzed performance between database [5–7] or the modeling impact on the performance in databases [8].

The remaining of this article as follows: Section 2 presents relevant topics related to NoSQL databases and MongoDB. Section 3 presents the conception of an experimental project. Section 4 describes the obtained results, which analysis is in Section 5. Finally, Section 6 presents the final remarks of the presented study.

2 Background

2.1 NoSQL Databases

NoSQL databases were developed to fulfill storage requirements in big data environments. In that way, their schemaless data structure provides more flexibility to many applications, such as e-mails, documents, and social media content [9, 10].

The NoSQL term refers to wide variety of storage systems, which are non-strict ruled by ACID properties, to allow better data structure and horizontal performance [4], join operations, high scalability, and data modeling by simplified queries [10]. The relational databases are divided into four categories: document-oriented, column-oriented, key/value-oriented, graph-oriented, and multimodal.

This work focus on oriented-documents databases, which modeling is similar to object-oriented data definition using registers with fields and complex operations [6]. Each database contains collections, each collection defines similar content groups, and each item corresponds to a document structured as JSON (JavaScript Object Notation) or XML (Extensible Markup Language).

[1] https://mongodb.github.io/node-mongodb-native/
[2] https://mongoosejs.com/
[3] https://www.mongodb.com/

2.2 MongoDB

MongoDB is an open-source document-oriented database, which provides besides storage functionalities such as data sorting, secondary indexing, and interval queries [11]. It does not require a defined schema, despite the similarity of elements into a collection [8, 12]. There are two main approaches to document modeling:

- **Embedded data modeling:** the data are defined in a unique data structure or document, which results in a high concentration of data.
- **Normalized data modeling:** the data have references among documents to represents relationships.

The adopted format is JSON, which passes through a codification to binary format BSON [4] . About the integration to programming languages, several drivers are available to Java, C++, C#, PHP and Python [12], and Node.js too.

Concerning to Node.js drivers to MongoDB, the first is the MongoClient [5], the official distributed solution, which provides an API to manipulation of data. The main feature is the implicit document-object modeling, which discards any need for data description.

The second option is the Mongoose [6], a database driver that provides data modeling based on the object-relational model. It implies that all data elements must describe the definition of attributes, which allows verification of types and validation, nullity checking by a defined schema.

3 Experimental Setup

In this section, we present the definitions of the experimental project which intents to compare the performance of each driver with MongoDB. Figure 1 presents the structure of experiment.

The analysis aimed to identify which couple (driver-database) presents better performance in a Node.js application based on quantitative parameters.

We developed a tool to run the test with each driver. In that way, to perform the comparison, the execution flow of application receives a set of parameters, such as driver and number of elements, in follow the database connection is performed and the respective operations. During the tests, we extracted some metrics related to CPU (Central Processing Unit) and memory usage. The tool is available in the following open-source repository: OMITTED.

We also defined the performance metrics to evaluate the conducted tests:

- **Average Execution Time:** it defines the average time (in milliseconds) in each operation.

[4] http://bsonspec.org/
[5] https://mongodb.github.io/node-mongodb-native/index.html
[6] https://mongoosejs.com/

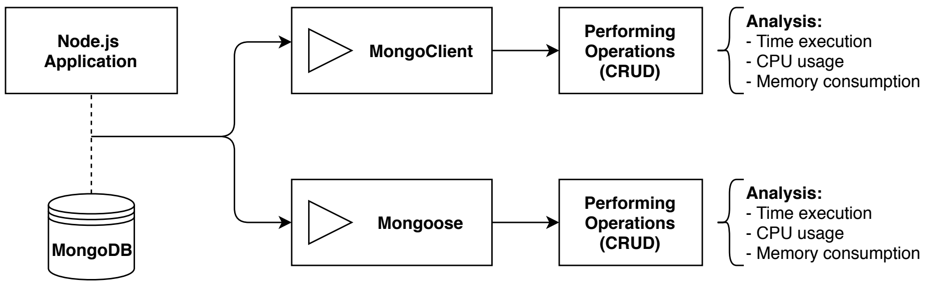

Fig. 1: Comparison between MongoDB drivers.

- **Average CPU Usage:** it defines the average time (in milliseconds) usage of CPU during each operation.
- **Average Memory Usage:** it defines the average variation of RAM memory usage (in kilobytes) in each operation.

Finally, we formulate a set of research questions to lead the analysis of the results:

- **RQ1** – *Does the driver selection impact on application performance under time execution?*
- **RQ2** – *Does the driver selection affect application performance under CPU usage?*
- **RQ3** – *Does the driver selection impact application performance under RAM memory consumption?*

3.1 Dataset

We used a dataset of 18 thousands of instances in the conducting of the experiment. Initially, all registers have 89 attributes (mainly textual) with an average size of 1.37 KB. From the original dataset, we also built a second dataset with reduced instances (only six attributes) using the same instances, which resulted in an average size of 0.13 KB.

We applied both datasets in the experimental process, in which reduce dataset intent to compare the drivers about the relation between the number of attributes and data modeling impact. In Table 1, we summarize the main characteristics of datasets:

Table 1: Description of experimental datasets.

	Dataset I	**Dataset II**
Number of instances	18,000	18,000
Number of attributes	89	6
Average size of instance	1.37 KB	0.13 KB

3.2 Execution Environment

The execution environment consisted of a machine running Ubuntu 18.04.2 Operating System, Intel i3 3217U processor, and 4GB DDR3 RAM. During the execution of the tests, the Node.js application execution environment was set to use the maximum size 3GB heap, thereby restricting the maximum operations of each test.

In each execution scenario, data regarding the runtime, CPU usage time, and RAM usage were extracted. Scenarios with different quantities of CRUD operations were analyzed, ranging from 1,000, 10,000, 100,000, and 200,000; each repeated 10 times and recorded the average of the executions. Performance metrics were obtained through the JSMeter library. [7].

4 Experiment Results

The results are presented from the perspective of each of the CRUD operations, where 100% of the records are reached in each operation. Each result refers to a specific operation of combining a driver with MongoDB in an application manipulating the large data set (with all attributes) or small data set (with a reduced number of attributes).

Figure 2 graphically illustrates the execution time when performing CRUD operations contrasting the use of both drivers. Considering the execution time for insert operations - Figure 2a, it was identified that the execution time with the use of driver Mongoose was longer for both sets, compared to the use of MongoClient, which showed no significant differences between the sets. It can also be noted that when manipulating the small dataset using Mongoose, there was a drop in execution time from 100,000 insert operations. A possible factor that justifies this behavior is the occurrence of set splitting in the insert operation when the quantity exceeds 100,000 items, according to MongoDB documentation, however, this does not occur for the large data set.

Considering the execution time for Search operations - Figure 2b, using MongoClient resulted in lower execution time for both sets compared to using Mongoose. Using Mongoose in this case, the executions for both sets exhibit increasing and proportional behavior to the detriment of the record size difference. Finally, from a processing time perspective, when analyzing test results for both update operations - Figure 2c and deletion operations - Figure 2d, similar behaviors were observed with the use of both drivers.

Figure 3 graphically illustrates CPU consumption when performing CRUD operations by contrasting the use of both drivers. Similarly to the previous analysis, for insert operations - Figure 3a and fetch - Figure 3b, MongoClient usage provides significantly lower average CPU consumption time in both sets, to the detriment of the high consumption time presented by Mongoose. For insertion, it also presents the exception case for 200,000 operations, whose possible justification is similar to the previous analysis.

[7] https://github.com/wahengchang/js-meter

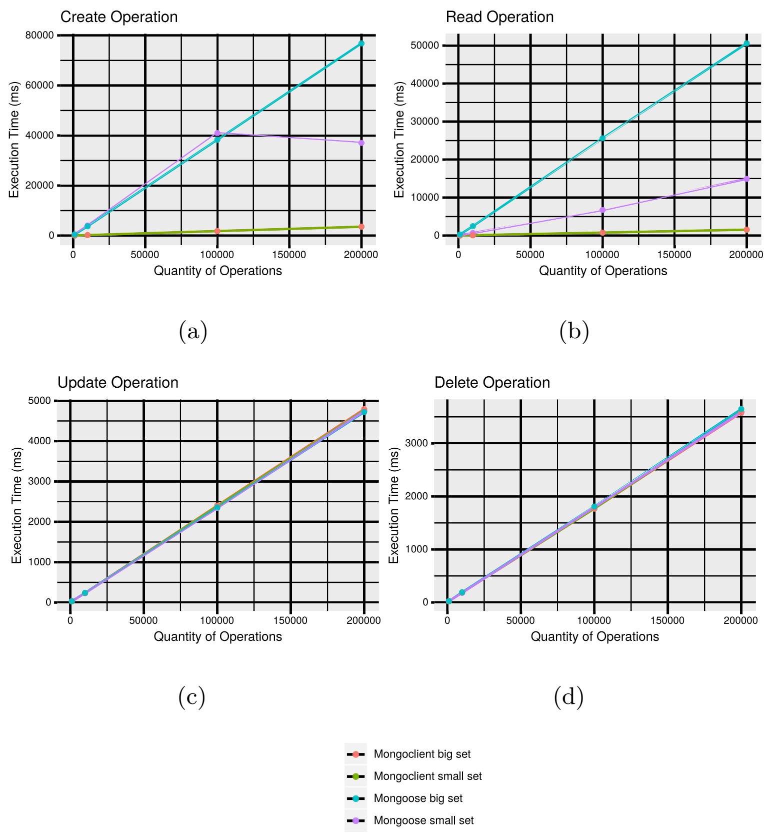

Fig. 2: Comparison of the use of drivers with the application of CRUD operations in relation to the runtime.

Figure 3c illustrates the CPU consumption when performing update operations, where it can be observed that using the Mongoose driver with a larger record set had a longer CPU consumption time, unlike the others, which were similar and with a shorter time, even with oscillations. Importantly, the processing time of all runs was less than 250 ms. In Figure 3d, CPU consumption is shown when performing deletion operation, each execution presented relatively unstable behavior, in which the use of driver MongoClient had a higher CPU consumption time, however, there is no significant difference because all executions obtained processing time of less than 10 ms.

Figure 4 graphically illustrates RAM consumption when performing CRUD operations contrasting the use of both drivers. Figures 4a and 4b respectively show the memory consumptions for insert and search operations, in which a pattern of memory usage cannot be identified, however it can be identified that predominantly driver Mongoose has a higher memory consumption in operations

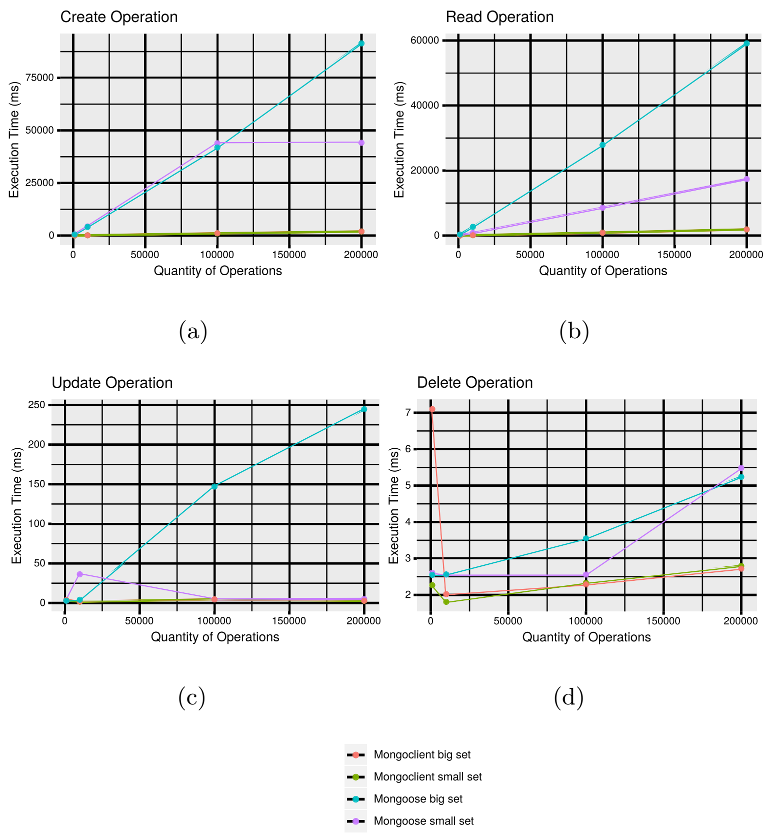

Fig. 3: Comparison of the use of drivers with the application of CRUD operations in relation to the CPU consumption.

performed on both sets to the driver MongoClient cases. For both operations, there is additional memory usage of around 30 to 40MB.

Finally, Figures 4c and 4d respectively show the memory consumption when performing an update and delete operations, do not show standardization for both drivers and sets. Despite having some points of instability, it is possible to notice that such operations consume little additional memory, approximately 1MB or less, even considering the oscillation peaks.

5 Analysis

This section presents the analysis of the results obtained, which are described from the perspective of the descriptive research questions in the Section 3.

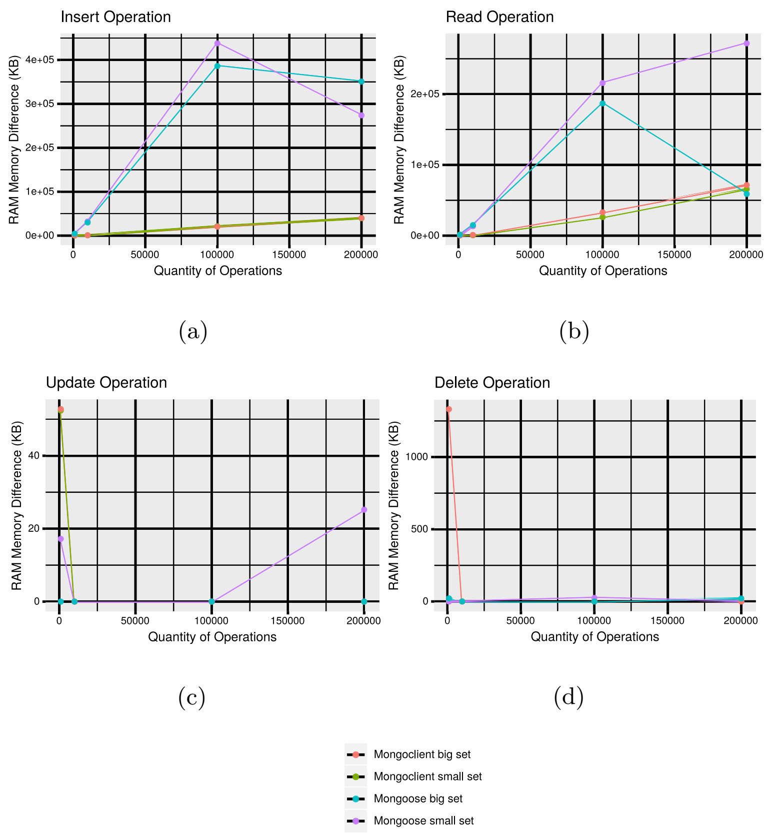

Fig. 4: Comparison of the use of drivers with the application of CRUD operations in relation to the Memory consumption.

5.1 *RQ1 – Does the driver selection impact on application performance under time execution?*

Regarding RQ1, in general, the tests performed using the Mongoose driver had a higher execution time compared to using MongoClient in two operations, while in the other operations the performance was similar. Thus, from the average execution time perspective of each operation performed on the MongoDB database, driver choice can impact performance, with MongoClient being the best choice for a Node.js application that makes use of MongoDB if time execution is a critical factor for this application.

5.2 *RQ2 – Does the driver selection affect application performance under CPU usage?*

As in RQ1, tests performed using MongoClient had lower CPU consumption time compared to Mongoose in both sets, mainly for insert and fetch operations, while in the other operations (update and delete), it was also recorded better performance, but to a lesser extent. In short, in terms of processing time, driver choice can influence performance, also presenting MongoClient as the best option for a Node.js application that makes use of MongoDB.

5.3 *RQ3 – Does the driver selection impact application performance under RAM memory consumption?*

Considering RQ3, we have that for insert and search operations, in most execution cases, driver Mongoose has higher memory consumption, while for update and delete operations there are no significant differences. However, it is noteworthy that in none of the comparisons there was stable behavior Thus, in terms of memory consumption, the choice of driver does not significantly impact all operations, despite the lower consumption by driver MongoClient.

5.4 Overview

In general, the data obtained indicate that insertion and search operations are the most costly in execution time, in the worst cases approximately 70,000 to 80,000 ms, while the others are less than 5,000 ms. From the perspective of CPU consumption time the same analysis is also used, the operations indicate approximate proportionality compared to the total execution time. As for memory usage, both operations are also costly, even if nonlinear, at levels close to 30 to 40MB, to the detriment of other operations with usage close to or less than 1MB.

Considering the average difference in record size of the adopted dataset, driver MongoClient was indifferent, showing no significant oscillations, while Mongoose presented performance directly proportional to the record size.

In terms of driver comparison, MongoClient performed more stable from the perspective of all adopted metrics than Mongoose. Therefore, it is concluded that, under the exclusive performance criterion, MongoClient presents itself as the best option for the Mongoose. It should be noted that if any additional resources provided by one of the options, such as data verification or ease of implementation, are relevant factors, the choice should be reevaluated, however, this study is quantitative and is not intended to measure the use of additional resources. which may vary from context and application used.

6 Final Remarks

This paper presents a study that analyzes the influence of using drivers, contrasting the use of two distinct drivers, MongoClient and Mongoose, on the performance of a Node.js application that makes use of the NoSQL MongoDB database.A performance analysis study was conducted considering aspects such as runtime, CPU time, and RAM usage in performing CRUD operations. Also, the impacts resulting from the variation in the average size of the records in the dataset were compared.

In general, from the quantitative results, it was found that the performance of a Node.js application integrated with MongoDB when using driver MongoClient is generally better compared to using driver Mongoose, especially under the runtime and CPU consumption and, less significantly, RAM consumption, considering an application developed in a Node.js environment.

As an additional contribution, there is the implementation and availability of the testing tool, in order to enable the execution of future analyzes in Node.js environments.

References

1. Ward, J.S., Barker, A.: Undefined by data: a survey of big data definitions. arXiv preprint arXiv:1309.5821 (2013)
2. González-Aparicio, M.T., Younas, M., Tuya, J., Casado, R.: A new model for testing crud operations in a nosql database. In: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). pp. 79–86 (2016)
3. Han, J., Haihong, E., Le, G., Du, J.: Survey on nosql database. In: 2011 6th international conference on pervasive computing and applications. pp. 363–366. IEEE (2011)
4. Rafique, A., Van Landuyt, D., Lagaisse, B., Joosen, W.: On the performance impact of data access middleware for nosql data stores a study of the trade-off between performance and migration cost. IEEE Transactions on Cloud Computing 6(3), 843–856 (2018)
5. Jung, M., Youn, S., Bae, J., Choi, Y.: A study on data input and output performance comparison of mongodb and postgresql in the big data environment. In: 2015 8th International Conference on Database Theory and Application (DTA). pp. 14–17 (2015)
6. Patil, M.M., Hanni, A., Tejeshwar, C.H., Patil, P.: 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. In: 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC). pp. 325–330 (2017)
7. Ongo, G., Kusuma, G.P.: Hybrid database system of mysql and mongodb in web application development. In: 2018 International Conference on Information Management and Technology (ICIMTech). pp. 256–260 (2018)
8. Kanade, A., Gopal, A., Kanade, S.: A study of normalization and embedding in mongodb. In: 2014 IEEE International Advance Computing Conference (IACC). pp. 416–421. IEEE (2014)
9. Mohamed, M., G. Altrafi, O., O. Ismail, M.: Relational vs. nosql databases: A survey. International Journal of Computer and Information Technology (IJCIT) 03, 598 (2014)

10. Ramesh, D., Khosla, E., Bhukya, S.N.: Inclusion of e-commerce workflow with nosql dbms: Mongodb document store. In: 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). pp. 1–5 (2016)
11. Membrey, P., Plugge, E., Hawkins, D.: The definitive guide to MongoDB: the noSQL database for cloud and desktop computing. Apress (2011)
12. Lutu, P.: Big data and nosql databases: new opportunities for database systems curricula. Proceedings of the 44th Annual Southern African Computer Lecturers' Association, SACLA pp. 204–209 (2015)