Analysis of Node.js Application Performance using MongoDB Drivers

Leandro Ungari Cayres[1], Bruno Santos de Lima[1] Rogério Eduardo Garcia[1], and Ronaldo Celso Messias Correia[1]

Faculdade de Ciências e Tecnologia – Universidade Estadual Paulista (UNESP), Presidente Prudente – SP, Brazil,
{leandro.ungari,bruno.s.lima,rogerio.garcia,ronaldo.correia}@unesp.br

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 Non-Relational 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/

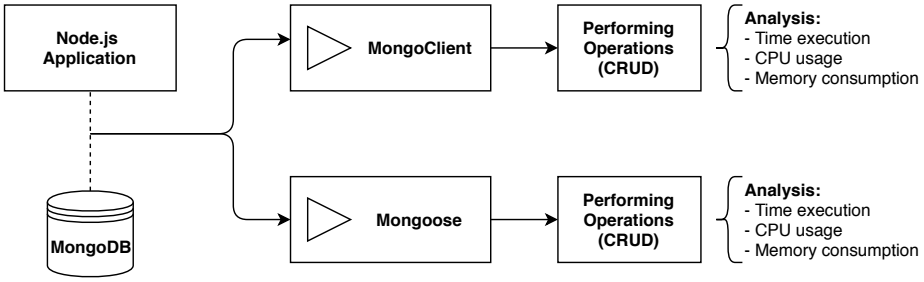

Fig. 1. Comparison between MongoDB drivers.

- **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 Ambiente de Execução

O ambiente de execução foi composto por uma máquina com Sistema Operacional Ubuntu 18.04.2, processador Intel i3 3217U e memória RAM de 4GB DDR3. Durante a execução dos testes, o ambiente de execução da aplicação Node.js foi definido o uso do *heap* de tamanho máximo 3GB, desse modo restringindo o máximo de operações de cada teste.

Em cada cenário de execução, foram extraídos dados relativos ao tempo de execução, tempo de uso de CPU e uso de memória RAM. Foram analisados cenários com diferentes quantidades de operações CRUD, as quais variaram de 1000, 10000, 100000 e 200000; cada qual repetido 10 vezes e registrada a média das execuções. A obtenção das métricas de desempenho foi realizada através da biblioteca JSMeter [7].

4 Resultados do Experimento

A seguir os resultados são apresentados sob a perspectiva de cada uma das operações CRUD, em que 100% dos registros são atingidos em cada operação. Cada resultado refere-se a uma operação específica, da combinação de um *Driver* com o MongoDB em uma aplicação manipulando o conjunto de dados grande (com todos atributos) ou conjunto de dados pequeno (com número reduzido de atributos).

Na Figura 2 é ilustrado graficamente o tempo de execução ao realizar operações CRUD contrastando a utilização de ambos os *Drivers*. Considerando o tempo de execução para operações de inserção – Figura 3a, foi identificado que o tempo de execução com o uso do *Driver Mongoose* foi maior, para os dois conjuntos, em comparação com o uso do *MongoClient*, no qual não demonstrou diferenças significativas entre os conjuntos. Pode-se observar também que ao manipular o conjunto de dados pequeno com o uso do *Mongoose*, ocorreu uma queda no tempo de execução a partir de 100.000 operações de inserção. Um possível fator que justifique esse comportamento consiste na ocorrência de divisão de conjuntos na operação de inserção, quando a quantidade excede 100 000 itens, conforme a documentação do MongoDB, contudo, isso não ocorre para o conjunto de dados grande.

Considerando o tempo de execução para operações de Busca – Figura 3b, o uso do *MongoClient* resultou em um tempo de execução inferior, para ambos os conjuntos, se comparado a utilização do *Mongoose*. Ao utilizar o *Mongoose* neste caso, as execuções para ambos os conjuntos apresentam comportamento crescente e proporcional em detrimento a diferença de tamanho dos registros. Por fim, ainda na perspectiva de tempo de processamento, ao analisar os resultados dos testes tanto para operações de atualização – Figura 3c, quanto para operações de deleção – Figura 3d, observou-se comportamentos similares com o uso de ambos os *Drivers*.

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

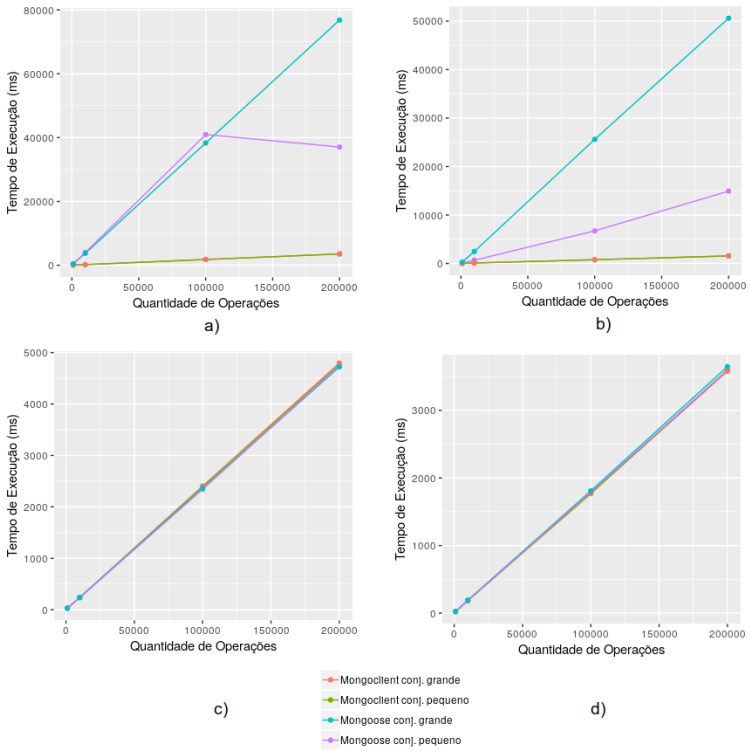

Fig. 2. Contraste do uso dos *Drivers* com a aplicação de operações CRUD em relação ao tempo de execução.

Na Figura 3 é ilustrado graficamente o consumo de CPU ao realizar operações CRUD contrastando a utilização de ambos os *Drivers*. Analogamente a análise anterior, para as operações de inserção – Figura 4a e busca – Figura 4b, o uso *MongoClient* prove um tempo médio de consumo de CPU significativamente inferior, em ambos os conjuntos, em detrimento do alto tempo de consumo apresentado pelo *Mongoose*. Para a inserção, também apresenta o caso de exceção para 200 000 operações, cuja possível justificativa é semelhante a análise anterior.

Na Figura 4c é ilustrado o consumo de CPU ao realizar operações de atualização, em que pode-se observar que com o uso do *Driver Mongoose* com conjunto de registros maiores apresentou maior tempo de consumo da CPU, diferentemente dos demais, que foram semelhantes e com tempo menor, mesmo com oscilações. É importante salientar que o tempo de processamento de todas as execuções foram inferiores a 250 ms. Na Figura 4d, é ilustrado o consumo de CPU ao realizar operação de deleção, cada execução apresentou comportamento relativamente instável, nas quais o uso do *Driver MongoClient* apresentou um tempo mais elevado de consumo da CPU, entretanto não há diferença significa-

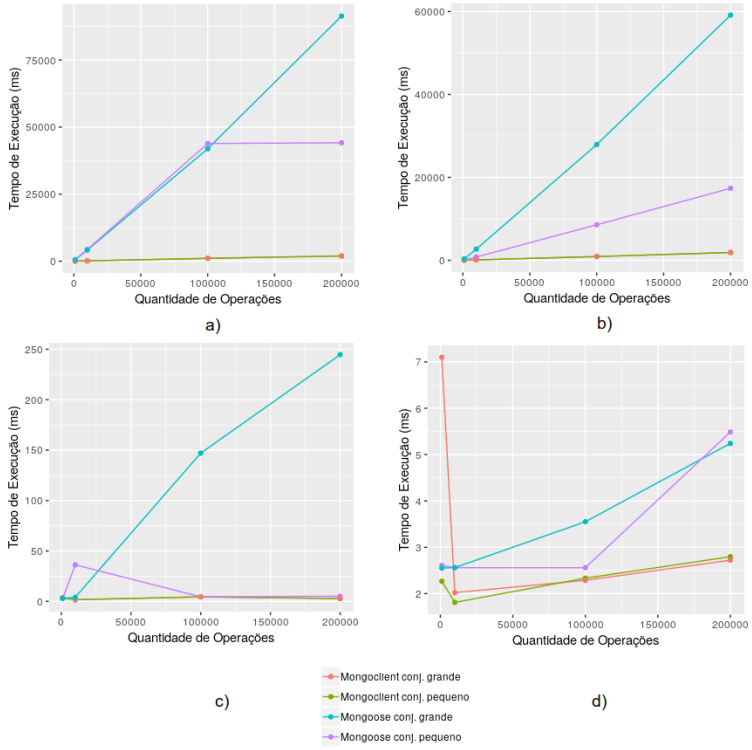

Fig. 3. Contraste do uso dos *Drivers* com a aplicação de operações CRUD em relação ao consumo de CPU.

tiva, porque todas as execuções obtiveram tempo de processamento inferior a 10 ms.

Na Figura 4 é ilustrado graficamente o consumo de Memória RAM ao realizar operações CRUD contrastando a utilização de ambos os *Drivers*. Nas Figuras 5a e 5b são ilustrados respectivamente os consumos de memória para as operações de inserção e busca, no qual não pode se identificar um padrão de uso de memória, entretanto pode-se identificar que predominantemente o *Driver Mongoose* tem um consumo maior de memória nas operações realizadas, em ambos os conjuntos, com relação aos casos do *Driver MongoClient*. Para ambas operações, há um uso adicional de memória em torno de 30 a 40MB.

Por fim, nas Figuras 5c e 5d, respectivamente são ilustrados os consumos de memória ao realizar operações de atualização e deleção, também não apresentam padronização para ambos os *Drivers* e conjuntos. Apesar de apresentar alguns pontos de instabilidade, é possível notar que tais operações consomem pouca memória adicional, aproximadamente 1MB ou menos, mesmo considerando os picos de oscilação.

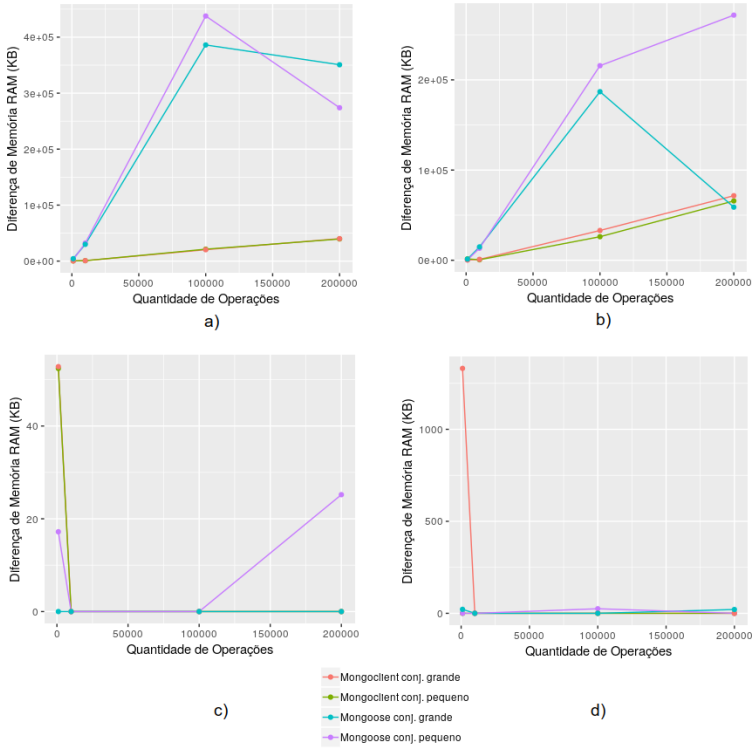

Fig. 4. Contraste do uso dos *Drivers* com a aplicação de operações CRUD em relação ao consumo de memória.

5 Análise

Nessa seção são apresentadas as análises a respeito dos resultados obtidos, as quais são descritas sob a perspectiva das questões de pesquisa descristas na Seção 3.

5.1 *QP1 – A escolha do Driver impacta no desempenho da aplicação Node.js que utiliza o MongoDB, considerando o tempo de execução de cada uma das operações CRUD?*

Com relação a QP1, em linhas gerais, os testes executados utilizando o *Driver Mongoose* apresentaram tempo de execução superior se comparado com a utilização do *MongoClient* em duas operações, enquanto nas demais operações o desempenho foi semelhante. Desse modo, sob a perspectiva de tempo médio de execução de cada operação realizada no Banco de Dados *MongoDB*, temos que, a escolha do *Driver* pode impactar no desempenho, tendo o *MongoClient* como melhor opção para uma aplicação Node.js que faz uso do *MongoDB* quando o tempo de execução for um fator crítico para essa aplicação.

5.2 *QP2 – A escolha do Driver pode influenciar no desempenho da aplicação Node.js que utiliza o MongoDB, considerando o tempo de consumo da CPU ao executar operações CRUD?*

Assim como na QP1, os testes executados utilizando o *MongoClient* obtiveram menor tempo de consumo da CPU em relação ao *Mongoose*, em ambos os conjuntos, principalmente para as operações de inserção e busca, enquanto nas demais operações (atualização e deleção), também foi registrado melhor desempenho, contudo em proporção menor. Em suma, em termos de tempo de processamento, a escolha do *Driver* pode influenciar no desempenho, também apresentando *MongoClient* como melhor opção para uma aplicação Node.js que faz uso do MongoDB.

5.3 *QP3 – A escolha do Driver impacta de modo relevante no desempenho da aplicação Node.js que utiliza o MongoDB, considerando o consumo médio de memória RAM ao executar operações CRUD?*

A respeito da QP3, tem-se que para as operações de inserção e busca, na maioria dos casos de execução, o *Driver Mongoose* apresenta maior consumo de memória, enquanto para as operações de atualização e deleção não há diferenças significantes. Contudo cabe ressaltar que em nenhuma das comparações houve comportamento estável. Desse modo, em termos de consumo de memória, temos que, a escolha do *Driver* não impacta de modo relevante para todas as operações, apesar do consumo inferior por parte do *Driver MongoClient*.

5.4 Análise Geral

De modo geral, os dados obtidos indicam que as operações de inserção e busca são as mais custosas quanto ao tempo de execução, para os piores casos, aproximadamente 70 000 a 80 000 ms, enquanto as demais são inferiores a 5 000 ms. Sob a perspectiva de tempo de consumo da CPU também se vale a mesma análise, inclusive as operações indicam aproximada proporcionalidade em comparação ao tempo de execução total. Quanto ao uso de memória, ambas operações também apresentam maior custo, mesmo que não-linear, em níveis próximos de 30 a 40MB, em detrimento das demais operações com uso próximo ou inferior a 1MB.

A respeito da diferença média de tamanho dos registros do conjunto de dados adotado, o *Driver MongoClient* apresentou-se de modo indiferente, não apresentando oscilações significativas, enquanto o *Mongoose* apresenta o desempenho diretamente proporcional ao tamanho do registro.

Em termos de comparação entre os *Drivers*, o *MongoClient* apresentou desempenho mais estável e de menor custo sob a perspectiva de todas as métricas adotadas, em detrimento ao *Mongoose*. Portanto, conclui-se que, sob o critério exclusivo de desempenho, o *MongoClient* apresenta-se como melhor opção com

relação ao concorrente. Cabe ressaltar que se quaisquer recursos adicionais providos por uma das opções, como verificação de dados ou facilidade de implementação, consistirem em fatores relevantes, deve-se reavaliar a escolha, contudo, esse estudo tem caráter quantitativo e não visa mensurar a utilização de recursos adicionais que podem variar de contexto e aplicação utilizados.

6 Considerações Finais

Este artigo apresenta um estudo que analisa a influência do uso de Drives, contrastando a utilização de dois Drivers distintos: *MongoClient* e *Mongoose*, no desempenho de uma aplicação Node.js que faz uso do Banco de Dados *NoSQL* MongoDB.Um estudo de análise de desempenho foi conduzido verificando aspectos quanto a tempo de execução, tempo de consumo da CPU e uso de memória RAM na execução de operações CRUD; além de comparar os impactos com variação do tamanho médio dos registros presentes no conjunto de dados.

De modo geral, através dos resultados quantitativos, foi constatado que o desempenho de uma aplicação Node.js integrada com o MongoDB ao utilizar o *Driver MongoClient* é em geral melhor se comparado com a utilização do *Driver Mongoose*, principalmente sob as métricas de tempo de execução e consumo da CPU, e, de modo menos significativo, quanto ao consumo de memória RAM, considerando uma aplicação desenvolvida em ambiente Node.js.

Como contribuição adicional, tem-se a implementação e disponibilização da ferramenta de testes, de modo a viabilizar a execução de futuras análises em ambientes Node.js.

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)