Análise de Desempenho dos Drivers do MongoDB em um Ambiente de Aplicação Node.js

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. Nos últimos anos, o número de Banco de Dados NoSQL tem crescido significativamente. Drivers tem sido desenvolvidos visando flexibilizar e apoiar execução das operações pertences aos bancos de dados. Há casos em que, para o mesmo Banco de Dados, existem diversas soluções, o que dificulta o entendimento dos impactos que a escolha de um Driver pode resultar no desempenho da aplicação. Neste artigo, é apresentado um estudo comparativo entre dois Drivers para MongoDB em aplicações Node.js, em que foram executadas operações CRUD para analisar o possível impacto no desempenho da aplicação com base em métricas de tempo de execução, consumo de CPU e memória. Os resultados demonstram que, sob análise quantitativa, o Driver MongoClient demonstrou possuir melhor desempenho em comparação ao Driver Mongoose, considerando as métricas empregadas, podendo ser uma melhor escolha a ser utilizada ao desenvolver uma aplicação Node.js.

Keywords: Análise de Desempenho, Aplicação Node.js, MongoDB, Drivers, Data Base

1 Introdução

Nos últimos anos, o crescimento no volume de dados mudou a utilização desses por empresas e organizações. Inicialmente como agentes passivos, relacionados às regras de negócio empresarial; os dados se tornaram potenciais oportunidades de lucro e obtenção de conhecimento através de processos de análise de dados.

O advento do *Big Data* não implicou somente em maior espaço de armazenamento, mas em uma mudança de organização, considerando características como volume, variedade, velocidade e valores [1]. A arquitetura dos tradicionais Bancos de Dados Relacionais são baseadas em propriedades ACID (*Atomicity*, *Consistency*, *Isolation* e *Durability*), contudo, em ambientes de *Big Data* a alta consistência provida afeta diretamente os aspectos de disponibilidade e eficiência, que são primordiais, devido ao alto volume, variedade e velocidade [2]. Nesse cenário, os Banco de Dados *NoSQL* (*Not only SQL*) proporcionam maior flexibilidade estrutural, escalabilidade, suporte a replicação e consistência eventual, desse modo, surgem como alternativas de destaque [3].

Neste contexto, para os diferentes ambientes de desenvolvimento e linguagens de programação, *Drivers* tem sido desenvolvidos de modo a viabilizar e apoiar execução de operações pertences aos bancos de dados. Entretanto, em muitos casos, além de recentes, os *Drivers* apresentam limitações em sua implementações, falhas de entendimento e efeitos colaterais no acesso aos dados [4]. Assim, a decisão de qual combinação entre Banco de Dados *NoSQL* e *Driver* a ser empregada pode ser um problema, devido ao desconhecimento dos pontos positivos e negativos.

Neste artigo, por meio de um experimento comparativo de desempenho, é realizado a avaliação das duas principais soluções de *Drivers* para o MongoDB [1], respectivamente *MongoClient* [2] e *Mongoose* [3], em ambientes de aplicação Node.js. O principal fator considerado para a realização dessa análise consiste na definição prévia de esquema para a manipulação dos dados em operações de CRUD (*create-read-update-delete*). Conceitualmente, os Bancos de Dados *NoSQL* não requerem essa predefinição, proporcionando flexibilidade, entretanto, não existe nada que impeça sua utilização, principalmente quanto a respeito do desempenho; possibilitando algum impacto relevante.

A escolha do MongoDB ocorreu devido a sua crescente investigação por parte da comunidade cientifica em diversas pesquisas [5–8]; além de ser uma das principais opções de armazenamento orientada a documentos. Quanto ao Node.js, mesmo recente, alguns trabalhos apontam a sua viabilidade tecnológica no desenvolvimento de aplicações [9]. Com essa escolha, tanto aplicação quanto Banco de Dados utilizam *JavaScript*, uniformizando o sistema em termos de linguagem de programação.

Este trabalho está situado como único presente na literatura com investigação na perspectiva de avaliar a influência de diferentes Drivers do MongoDB no desempenho de uma aplicação Node.js, uma vez que, os demais trabalhos investigam outros aspectos relacionados ao MongoDB como por exemplo: comparação de desempenho com outros Bancos de Dados [6, 5, 7] e influência na performance do Banco de Dados ao utilizar diferentes modelagens de dados [8].

Este artigo está organizado do seguinte modo: na Seção 2 são apresentados conceitos de Banco de Dados *NoSQL*, MongoDB e seus *Drivers*. A Seção 3 é apresentada a estruturação do experimento. Na Seção 4 são descritos os resultados quantitativos obtidos, cuja a análise está situada na Seção 5. Por fim, a Seção 6 expressa as considerações finais.

2 Fundamentação Teórica

Nesta seção, são discutidos alguns conceitos fundamentais para o trabalho: Bancos de Dados *NoSQL*, as características do MongoDB e dos *Drivers MongoClient* e *Mongoose*.

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

2.1 Banco de Dados *NoSQL*

Os Bancos de Dados *NoSQL* foram desenvolvidos visando armazenar e processar grandes volumes de dados. Em linhas gerais os bancos de dados *NoSQL* são livres de esquematizações e mais propícios a lidar com dados não estruturados como e-mails, documentos e mídias sociais de maneira eficiente [10, 11].

O termo *NoSQL* é utilizado para se referir a uma ampla variedade de armazenamentos de dados, em que as restrições de transação ACID foram suavizadas permitindo melhor dimensionamento e desempenho horizontal [4], proporcionando esquemas menos estruturados, suporte a operações de junção, alta escalabilidade, modelagem de dados simples com linguagem de consulta simples [11]. Os bancos de dados *NoSQL* são categorizados em: armazenamento de documentos, famílias de colunas, chave/valor, grafos e multimodais [2].

Este trabalho tem como foco a categoria orientada a documentos, a qual possui modelagem de dados estreitamente relacionados a programação orientada a objetos, possibilitando flexibilidade no armazenamento de registros com atributos distintos, sendo útil na modelagem de dados não-estruturados e polimórficos. Permite consultas robustas, em que qualquer combinação de campos no documento pode ser realizada [5]. Os dados são organizados em coleções de documentos, as quais utilizam uma estrutura semelhante a JSON (*JavaScript Object Notation*) ou XML (*Extensible Markup Language*).

2.2 MongoDB

O MongoDB é um Banco de Dados orientado a documentos que possui código-aberto, o qual provê funcionalidades como ordenação, indexação secundária e consultas de intervalo [12].

O banco de dados não impõe um esquema prévio, entretanto, normalmente todos os documentos em uma coleção são de propósito semelhante ou relacionado [8, 13]. Há duas abordagens para modelagem de documentos:

- **Modelo de dados incorporado:** Os dados relacionados são incorporados em uma única estrutura ou documento. Esses esquemas são geralmente conhecidos como modelos sem normalização.
- **Modelo de dados normalizado:** Os dados possuem referências de documentos para registrar relacionamentos entre esses, mas a combinação de documentos deve ser feita diretamente no código-fonte da aplicação.

O armazenamento dos dados ocorre através da serialização de objetos *Javascript*, também conhecidos como JSON, cuja implementação interna utiliza uma codificação binária chamada BSON[4]. O banco de dados MongoDB disponibiliza diversos *Drivers* para linguagens de programação como Java, C++, C#, PHP e Python [13], e também em Node.js.

Dentre os *Drivers* existentes para o MongoDB, tem-se como destaque o ***MongoClient***[5], consiste na solução oficial e nativa provida organização, provendo

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

um conjunto de funcionalidades que permite a manipulação dos dados e uso de recursos avançados do sistema. Essa solução é caracterizada pela modelagem documentos-objeto (*ODM – Object-Document Modeler*) de modo implícito ao banco de dados.

Assim como o anterior, o ***Mongoose*** consiste também em um *Driver* para MongoDB, em que provê a modelagem de dados utilizando um mecanismo objeto-relacional (*ORM – Object Relational Mapping*) utilizado em banco de dados relacionais [14], executando diversas tarefas de verificação e validação dos dados, como nulidade ou tipagem, previamente definidos por meio da elaboração de um esquema.

3 Composição do Experimento

Nesta seção é apresentado a composição do experimento comparando os *Drivers* *MongoClient* e *Mongoose* em integração com o MongoDB. Na Figura 1 é ilustrado a esquematização do experimento.

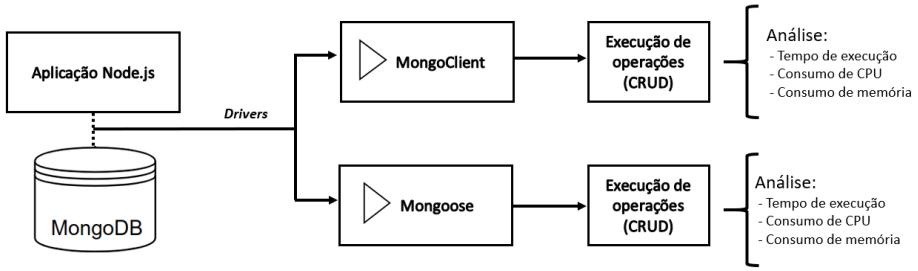

Fig. 1. Experimento: Comparação entre os *Drivers* do MongoDB.

As análises conduzidas visaram identificar qual dupla (MongoDB e *Drive*) possui melhor desempenho em uma aplicação Node.js, ou seja, causa menor impacto no desempenho da aplicação. Além disso, foi investigado se o tamanho médio de cada registro presente no conjunto de dados pode influenciar nesse desempenho e qual a proporção desse impacto.

Para isso, uma aplicação Node.js foi implementada para a condução dos testes. Essa aplicação é responsável por realizar a conexão com o Banco de Dados MongoDB e executar operações CRUD. A ferramenta de testes implementada em Node.js está disponível para a comunidade científica em: [OMITIDO], podendo ser utilizada em outras pesquisas.

As métricas escolhidas para as análises de cada um dos testes conduzidos foram:

– **Tempo médio de Execução:** Consiste tempo médio total de execução de cada operação específica.

– **Consumo médio de CPU:** Consiste no tempo médio de uso do processador durante a execução de cada operação específica.
– **Consumo médio de Memória:** Consiste na variação média de uso de memória RAM durante a execução de cada operação específica, sendo expressa em kilobytes (KB).

Por fim, foram formuladas as seguintes de questões de pesquisa que guiaram formalmente as análises executadas:

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?*

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?*

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?*

3.1 Características dos Conjuntos de Dados

A condução do experimento utilizou um conjunto de dados com cerca 18 mil instâncias.Originalmente, todos os registros presentes são compostos por 89 atributos, predominantemente textuais, obtendo um tamanho médio de 1,37KB. A partir do conjunto original, foi construído um conjunto reduzido em número de atributos (6 atributos), com o mesmo total de instâncias, porém com tamanho médio de 0,13KB.

Deste modo, dois conjuntos de dados foram utilizados para a experimentação, observe suas características descritas na Tabela 1. O conjunto reduzido tem o intuito de comparar a influência dos dois *Drivers* na manipulação de registros com diferentes quantidade de atributos.

Table 1. Características dos Conjuntos de Dados utilizado no Experimento

	Conjunto de Dados 1	**Conjunto de Dados 2**
Quantidade de instâncias	18 mil	18 mil
Quantidade de atributos	89	6
Tamanho de um registro	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 [6].

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 retrabalho, 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*.

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 ap-

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

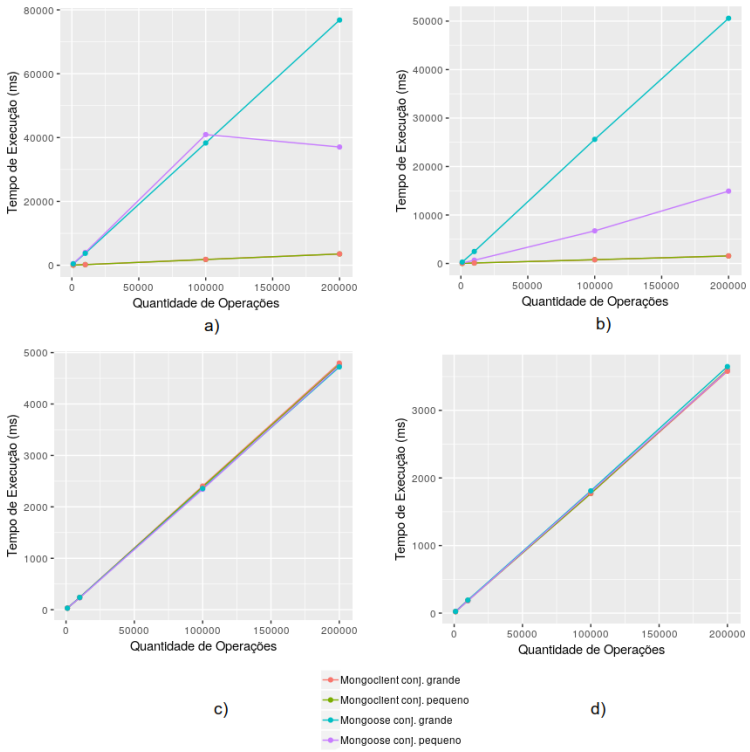

Fig. 2. Contraste do uso dos *Drivers* com a aplicação de operações CRUD em relação ao tempo de execução.

resentado 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 significativa, 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,

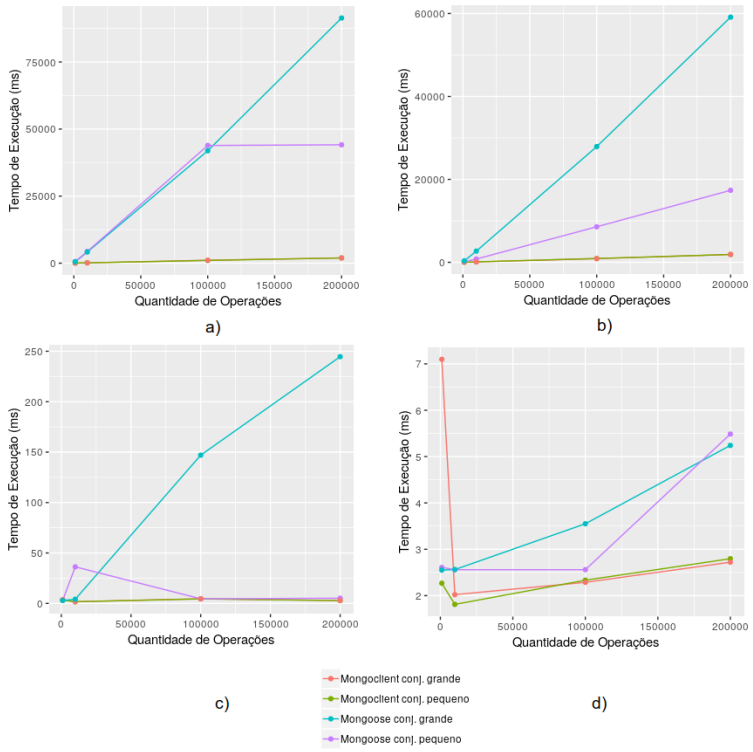

Fig. 3. Contraste do uso dos *Drivers* com a aplicação de operações CRUD em relação ao consumo de CPU.

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.

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.

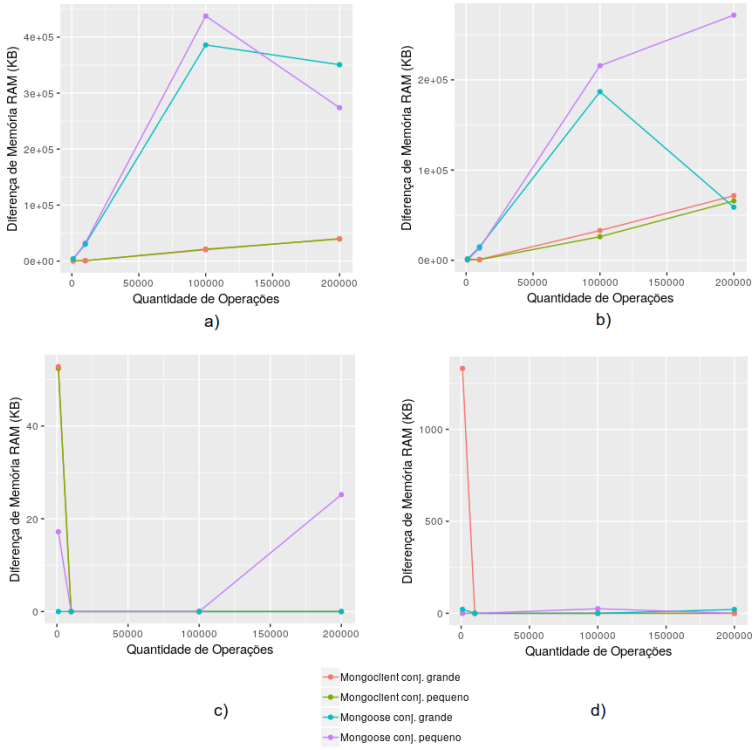

Fig. 4. Contraste do uso dos *Drivers* com a aplicação de operações CRUD em relação ao consumo de memória.

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 conjun-

tos, 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.Na avaliação foram conduzidas análises quantitativas 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. 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)
6. 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)
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. Chaniotis, I.K., Kyriakou, K.I.D., Tselikas, N.D.: Is node. js a viable option for building modern web applications? a performance evaluation study. Computing 97(10), 1023–1044 (2015)
10. 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)
11. 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)
12. Membrey, P., Plugge, E., Hawkins, D.: The definitive guide to MongoDB: the noSQL database for cloud and desktop computing. Apress (2011)
13. 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)
14. Mardan, A.: Boosting your node. js data with the mongoose orm library. In: Practical Node. js, pp. 149–172. Springer (2014)