COMPARAÇÃO DE DESEMPENHO ENTRE ARQUITETURAS DE BANCOS DE DADOS NOSQL

PAIVA, João Paulo de Oliveira[1]; TREVIZANO, Waldir Andrade[2]

[1] Graduação Ciência da Computação - UNIFAGOC
[2] Docente Ciência da Computação - UNIFAGOC

paiva.joaopaulo8@gmail.com
waldir@unifagoc.edu.br

RESUMO

Bancos de dados NoSQL surgiram para suprir limitações existentes no modelo relacional em cenários com alto volume de dados, porém a vasta gama de opções em cada uma das categorias não relacionais, cada qual com características específicas, torna árdua a tomada de decisão. O trabalho proposto pretende abordar os conceitos de bancos de dados NoSQL e terá como principal objetivo comparar as quatro arquiteturas não relacionais – orientado a documentos, chave-valor, orientado a colunas e baseado em grafos, considerando um representante típico de cada uma dessas arquiteturas, medindo o desempenho que cada uma fornece em relação as operações transacionais de inserção, alteração, busca e exclusão. Como resultado, identificou-se que o representante da categoria de SGBDs orientados a documentos se destaca em relação aos outros em grande parte dos casos.

Palavras-chave: NoSQL. Comparativo. Teste de desempenho.

INTRODUÇÃO

Os sistemas de gerenciamento de bancos de dados relacionais são a tecnologia predominante para armazenar dados estruturados em aplicativos da Web e de negócios. Em paralelo, conforme Demchenko *et al.* (2013), a quantidade de dados cresce a cada dia, assim como a possibilidade para sua utilização. Considerando tal situação, todos os dias são criados 2,5 quintilhões de bytes em forma de dados (sendo 1 quintilhão igual a 10 elevado a 18ª potência) e, hodiernamente, 90% de todos os dados que estão presentes no mundo foram criados nos últimos dois anos (IBM, 2014). Fatos como os citados se mostram prejudiciais ao modelo relacional, uma vez que este tem problemas de escalabilidade, pois seu desempenho degrada rapidamente conforme o volume dos dados aumenta (CUER, 2014).

Esse cenário levou ao desenvolvimento de um novo modelo de dados, que foi denominado NoSQL. Embora o termo NoSQL tenha sido desenvolvido anteriormente, foi após a introdução do conceito de banco de dados como serviço (DBaaS) que tal modelo ganhou um reconhecimento proeminente. O nome NoSQL é usado como um termo abrangente para todos os bancos de dados que não seguem os princípios dos bancos de dados relacionais (RDBMS) (TIWARI, 2011) e, devido à alta escalabilidade fornecida por eles, são vistos como concorrentes do modelo de banco de dados relacional.

Tendo como principal objetivo o de ampliar o escopo do modelo relacional e solucionar problemas mais específicos, os SGBDs NoSQL foram subdivididos de acordo com suas características próprias. Por exemplo, a Amazon utiliza seu sistema chave-valor Dynamo (DECANDIA *et al*., 2007) para gerenciar os dados de suas aplicações. Existem também sistemas orientados a colunas que foram baseados no Bigtable da Google (CHANG *et al*., 2008). E os orientados a documentos, como o MongoDB, e os conhecidos como bancos triplos ou orientados a grafos - neo4j. Cada um suprindo diferentes necessidades existentes nos bancos relacionais.

Problema e sua importância

Embora o uso de bancos de dados não relacionais tenha aumentado nos últimos anos, suas capacidades não foram expostas por completo. A classificação em quatro categorias que representam a classe de problemas tratados pelos diversos bancos NoSQL se mostram insuficientes para a tomada de decisão sobre a solução ideal a se adotar em determinada situação. Somado a isso, existem cerca de 225 opções de SGBDs NoSQL disponíveis (NOSQL, s.d.), o que dificulta ainda mais a escolha de um SGBD mais apropriado para um negócio específico, uma vez que é importante entender suas principais características. Cada SGBD não relacional também possui diferentes otimizações, resultando em diferentes tempos de carregamento de dados e tempos de execução para leituras ou atualizações.

REFERENCIAL TEÓRICO

Teste de desempenho

Segundo Costa (2012), testes de desempenho têm como propósito fornecer um indicador capaz de avaliar um sistema quanto aos seus requisitos de desempenho, expondo, consequentemente, as limitações do mesmo em diferentes cenários. Para se avaliar o desempenho será utilizado o teste de *stress*, que apura o comportamento de um sistema quando o mesmo é submetido a requisições acima de seus limites habituais de carga.

O requisito de desempenho considerado no desenvolvimento deste é a latência, que diz respeito ao tempo entre uma requisição e a obtenção de sua resposta.

NoSQL

O NoSQL, também conhecido como NOREL (Não Relacionais) ou NoJOIN (Não JOIN) (SHARMA *et al*., 2017), é um termo genérico que representa bancos de dados que não suportam o dialeto SQL, havendo vários modelos, cada um com linguagem e comportamento específicos. Projetados para lidar com dados estruturados e não estruturados, como: texto, multimídia, imagens, áudio e vídeo (PRADO, 2019), os bancos de dados não relacionais não organizam seus dados em tabelas.

O NoSQL é recomendado para aplicações que lidam com um grande volume de dados devido ao seu alto desempenho na recuperação destes, além de sua alta escalabilidade (CUER, 2014) e flexibilidade de esquema. Foi projetado para processamento distribuído, o que possibilita seu uso em arquiteturas distribuídas em rede (PRADO, 2019). O acesso aos dados é feito via API (GESSERT e RITTER,2016) e

garante a disponibilidade de dados e tolerância a partição, mas, em decorrência disso, as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) não podem ser obedecidas simultaneamente. Por fim, SGBDs NoSQL são open source, sendo assim, todos podem ver seu código livremente, atualizá-lo de acordo com suas necessidades e compilá-lo (NOSQL, s.d.).

Categorias NoSQL

Cada banco de dados NoSQL é classificado de acordo com a forma em que armazena os dados, de modo a pertencer a uma das seguintes categorias (POPESCU, 2010):

- Armazenamento orientado a documentos: Neste modelo, pode existir um conjunto de documentos e em cada documento um conjunto de atributos com seus respectivos valores. Cada documento possui um identificador único. Esse modelo não possui esquema e armazena os dados em uma estrutura JSON (JavaScript Object Notation) ou XML (Extensible Markup Language) (PRADO, 2019).
- Armazenamento chave-valor (*Key*/*Value*): Os dados são armazenados em pares chave-valor. Ele é projetado para lidar com muitos dados e cargas pesadas. Os bancos de dados do tipo chave-valor armazenam dados como uma tabela de hash em que cada chave é exclusiva, e o valor pode ser um JSON, BLOB (Grande Objeto Binário) ou *strings*, por exemplo. É um dos tipos mais básicos de bancos de dados NoSQL, sendo usado como uma coleção, dicionários, matrizes associativas, entre outros (LEAVITT, 2010).
- Armazenamento orientado a colunas: Um banco de dados orientado a colunas é otimizado para recuperação rápida de colunas de dados, uma vez que armazena dados por colunas em vez de por linhas. Ao armazenar uma coluna inteira, minimiza-se o acesso ao disco ao selecionar algumas colunas de uma linha contendo muitas colunas. Em bancos de dados orientados a linhas, não há diferença entre selecionar apenas um ou todos os campos de uma linha (OUSOUSS *et al*., 2013).
- Armazenamento baseado em grafos: O armazenamento baseado em grafos fundamenta-se na teoria dos grafos. Basicamente, os grafos consistem em nós, propriedades e arestas. Cada nó representa uma entidade, suas propriedades representam os atributos e as arestas representam os relacionamentos (LEAVITT, 2010).

MÉTODO DE DESENVOLVIMENTO

Os SGBDs

Para a definição dos SGBDs a serem manipulados, fez-se uso do *ranking* DB-Engines[1], que classifica os sistemas de gerenciamento de banco de dados de acordo com sua popularidade. O *ranking* é atualizado mensalmente, suprindo outro fator importante. Esse *ranking* apontou MongoDB[2], Redis[3], Cassandra[4] e Neo4j[5] como os

[1] DB-Engines Ranking. Disponível em: https://db-engines.com/en/ranking. Acesso em 15 jul. 2019.
[2] MongoDB. Disponível em: https://www.mongodb.com/. Acesso em 13 jul. 2019.
[3] Redis. Disponível em: https://redis.io/. Acesso em 13 jul. 2019.

SGBDs orientado a documentos, orientado a colunas, chave-valor e baseado em grafos mais utilizados, sendo os três primeiros classificados também entre os 10 SGBDs mais populares no *rank* geral.

Ambiente de testes

Para a realização dos testes foram criados ambientes virtuais utilizando a ferramenta de virtualização Oracle VM VirtualBox[6] na versão 6.0.12, com o intuito de deixar o ambiente para cada tipo de NoSQL o mais homogêneo possível e garantir que nenhum dos SGBDs tenha vantagens em relação aos outros. Cada SGBD ficou em uma máquina virtual, contendo as mesmas configurações de memória RAM e o mesmo sistema operacional. A máquina nativa utilizada para os testes foi um microcomputador com processador Intel(R) Core(TM) i5-8400U CPU @ 2.8GHz, 8 GB de memória RAM DDR4, placa de vídeo Nvidia Geforce Gtx 1050 2gb Gddr5 128 Bits e 1 TB de armazenamento em disco rígido, rodando o Microsoft Windows 10 Home Single Language 64 bits.

As máquinas virtuais foram configuradas com o sistema operacional Ubuntu Linux 18.04.2 de 64 bits, 2GB de memória RAM, e HD dinamicamente alocado, podendo se expandir até 1 TB, e com 2 cores reservados para o processador.

Metodologia dos testes

Para cada operação executada nos bancos, foi aferido o tempo gasto em sua realização, em milessegundos. O tempo retornado foi avaliado levando em conta quantidade de registros e a operação transacional. Assim, foi possível identificar qual foi o mais eficiente em uma determinada operação e para que tipo de aplicação o banco tem uma resposta satisfatória. Mesmo que a operação seja a mesma, o tempo de execução pode ser diferente. Dessa forma, foram realizados 12 testes para cada transação e eliminados o menor e o maior intervalos computados. O resultado final é a média das 10 execuções restantes.

A cada teste realizado os SGBDs eram reiniciados, isso porque eles guardam as instruções no *buffer* de memória diminuindo o tempo a cada execução, o que mudaria drasticamente os resultados. Não foi instalado nenhum programa a não ser o próprio banco de dados a ser executado, evitando que outros processos consumissem a memória e outros recursos que pudessem influenciar nos testes. No ambiente físico também não havia nenhum programa executando a não ser a máquina virtual e processos nativos necessários.

Formação das estruturas de dados

As estruturas para armazenamento dos dados possuem dois campos: código, do tipo inteiro e definido como atributo identificador e descricao, do tipo *string*.

Foram criadas, em cada SGBD, 4 dessas estruturas: com um milhão, dez milhões, cinquenta milhões e cem milhões de "registros".

---

[4] Apache Cassandra. Disponível em: http://cassandra.apache.org/. Acesso em 13 jul. 2019.

[5] Neo4j. Disponível em: https://neo4j.com/. Acesso em 13 jul. 2019.

[6] Oracle VM VirtualBox. Disponível em: https://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html. Acesso em 23 jul. 2019.

RESULTADOS E DISCUSSÃO

Inserção

Os primeiros testes realizados foram os de inserção. Para uma grande massa de dados as documentações dos SGBDs recomendam a importação de um arquivo. Dessa forma, é possível inserir os dados a partir de um arquivo .csv, onde cada linha do documento .csv é uma linha a ser inserida e os atributos são separados por vírgula. Os códigos abaixo mostram como é feita a inserção por importação de arquivo csv.

O comando para inserção por importação do SGBD Neo4j não equivale aos dos demais SGBDs. Ele armazena os dados na memória para somente ao final da transação realizar o *commit*. Para deixar a comparação justa, foi utilizada a cláusula PERIODIC COMMIT, que instrui o Neo4j a confirmar os dados após um certo número de linhas. Isso reduz a sobrecarga de memória durante a transação, e, consequentemente, diminui a latência da transação.

Código 1: comando para inserção de arquivo .csv no MongoDB

```
mongoimport --db banco --collection colecao_a --type csv --columnsHaveTypes –fields "atributo_valor.int32(),colecao_a_id.int32()" --file dadosMongo.csv
```

Código 2: comando para inserção de arquivo .csv no Cassandra

```
COPY tabela_a (_id, descricao) FROM 'dadosCassandra.csv' WITH HEADER = true;
```

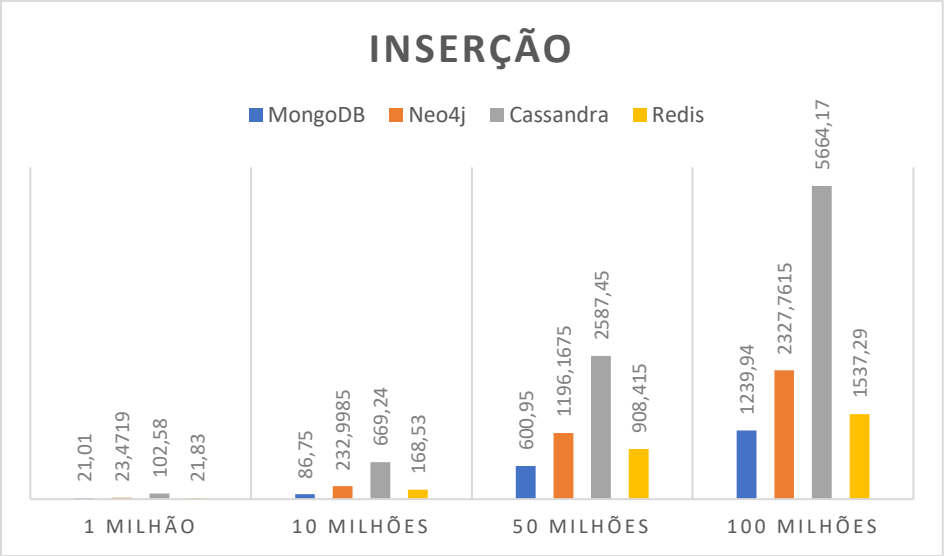
Código 3: comando para inserção de arquivo .csv no Redis

```
cat dadosRedis.csv | redis-cli –pipe
```

Código 4: comando para inserção de arquivo .csv no Neo4j

```
USING PERIODIC COMMIT 1
LOAD CSV WITH HEADERS FROM 'dadosNeo4j.csv' AS line
CREATE (:tabela_a { descricao: line.Descricao})
```

Figura 1: Gráfico gerado a partir do resultado dos testes de inserção



Fonte: dados da pesquisa.

Analisando o gráfico, pode-se notar que o MongoDB possui uma performance melhor que a dos outros SGBDs em todos os cenários quando se trata de operações de inserção. Em contraparte, o Cassandra apresenta resultados fortemente inferiores em todos os cenários.

Testes de busca

Os testes de busca foram divididos em duas fases: busca por todos os registros e busca por todos os registros usando a cláusula INNER JOIN.

Busca por todos os registros

Foi realizada a busca de todos os registros nas estruturas de dados. Os códigos abaixo mostram os comandos utilizados para tal consulta.

Código 5: comando de consulta de todos os registros no MongoDB

```
db.colecao_a.explain(“executionStats”).find();
```

Código 6: comando de consulta de todos os registros no Cassandra

```
select * from tabela_a;
```

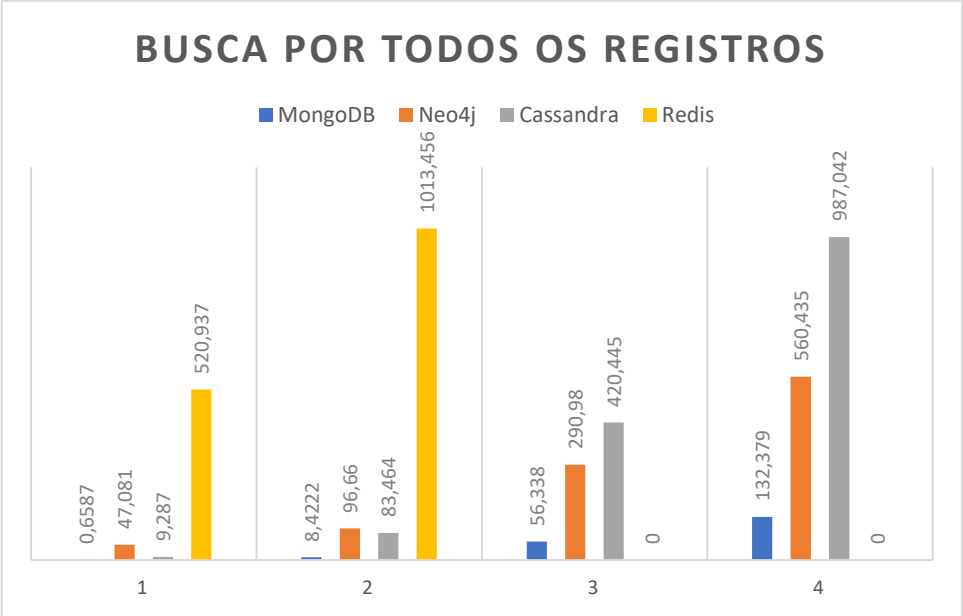
Código 7: comando de consulta de todos os registros no Redis

```
KEYS *
```

Código 8: comando de consulta de todos os registros no Neo4j

```
MATCH (n)
RETURN n;
```

Figura 2: Gráfico gerado a partir do resultado dos testes de busca por todos os registros



Fonte: dados da pesquisa.

Nota-se que o Redis leva desvantagem em todos os cenários de busca, mas isso fica mais evidente a partir de 50 milhões de registros, quando ele não conseguiu retornar a requisição por falta de memória. Isso se deve ao fato de que o comando utilizado para tal consulta bloqueia o servidor, fazendo com que ele não esteja disponível para realizar outras operações durante sua execução e consome recursos significativos para preparar o *buffer* de resposta, esgotando a memória do servidor.

Seguindo a documentação do Redis, o uso do comando KEYS deve ser realizado com cautela, uma vez que pode prejudicar o desempenho quando executado em bancos com uma grande massa de dados.

Busca por todos os registros usando INNER JOIN entre tabelas A e B

Para esta fase dos testes foi necessário omitir os bancos de dados Redis e Cassandra por não terem suporte oficial a junções.

O MongoDB também não suporta operações que envolvam relacionamentos, porém, ao contrário dos citados anteriormente, nele é possível simular um relacionamento colocando em uma nova coleção um atributo que seja preenchido em seguida com um código indentificador existente em outra coleção. Para fazer a junção entre as duas coleções utiliza-se a função *aggregate* do MongoDB. No Neo4j, relacionamentos são tratados normalmente com a cláusula MATCH, sendo necessário apenas criar outro grafo.

Código 9: comando de consulta de todos os registros no MongoDB

```
db.colecao_b.aggregate([{$lookup:{from:"colecao_a",localField:"coluna_a_id”,foreignField: "_id", as: "join"}}]);
```

Código 10: comando de consulta de todos os registros no Neo4j

```
MATCH (a:tabela_a)-[:relacionamento]->(b:tabela_b)
WHERE ID(a) = ID(b)
RETURN a;
```

Figura 3: Resultado dos testes de busca por todos os registros usando INNER JOIN

Fonte: dados da pesquisa.

Analisando o gráfico, nota-se que, na média dos resultados, o SGBD Neo4j se sai melhor, algo que fica ainda mais evidente nos testes com 50 e 100 milhões de registros. Isso pode ser explicado pelo fato de que o Neo4j tem suporte nativo e otimizações para funções de junção, fatores esses inexistentes no MongoDB.

Atualização por atributo identificador

Os códigos abaixo demonstram como realizar uma alteração por atributo identificador.

Código 11: comando de alteração de registro por atributo identificador no MongoDB

```
db.colecao_a.explain("executionStats").update({_id:20},{$set:{atributo_descricao:"novovalor"}},
{upsert: true});
```

Código 12: comando de alteração de registro por atributo identificador no Cassandra

```
UPDATE tabela_a SET descricao="novovalor" WHERE _id=20;
```

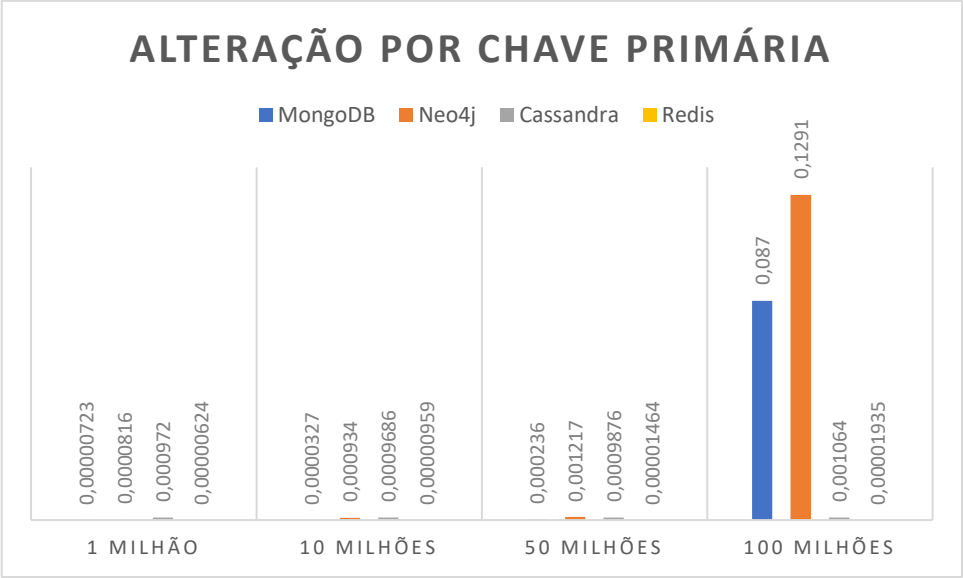
Código 13: comando de alteração de registro por atributo identificador no Redis

```
SET 20 "novovalor"
```

Código 14: comando de alteração de registro por atributo identificador no Neo4j

```
MATCH (n)
WHERE ID(n) = 20
SET n.descricao = 'novovalor'
```

Figura 4: gráfico gerado a partir do resultado dos testes de alteração por atributo identificador



Fonte: dados da pesquisa.

A partir da análise do gráfico de resultados, nota-se que a operação de atualização por chave primária acontece quase que instantaneamente em todos os SGBDs na maioria dos casos, com exceção para 100 milhões de registros, em que os SGBDs MongoDB e Neo4j têm uma ligeira perda de performance.

Remoção por atributo identificador

Os códigos abaixo mostram como realizar uma remoção por atributo identificador.

Código 15: comando de remoção de registro por atributo identificador no MongoDB

```
db.colecao_a.remove( { qty: { _id: 20 } } )
```

Código 16: comando de remoção de registro por atributo identificador no Cassandraa

```
DELETE FROM tabela_a WHERE _id=20;
```

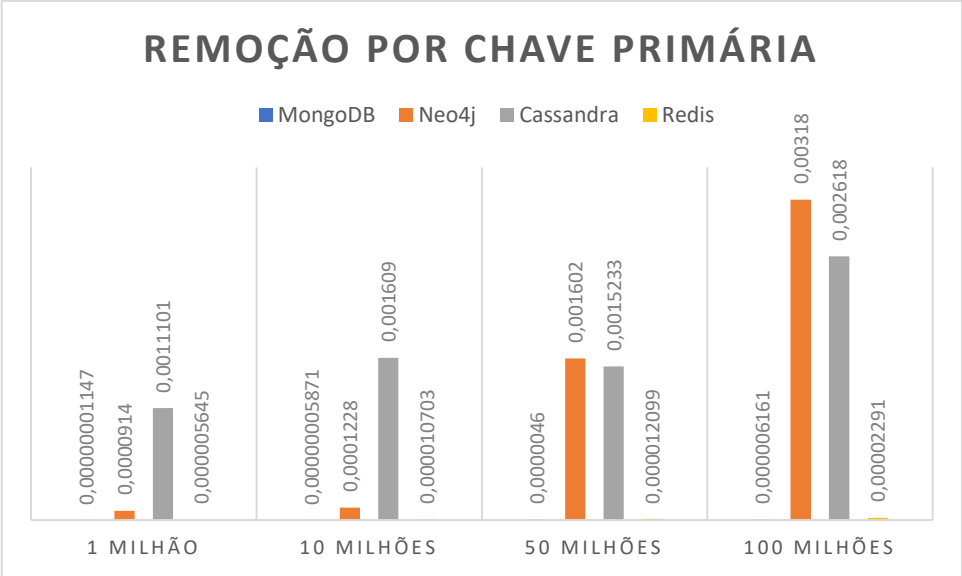
Código 17: comando de remoção de registro por atributo identificador no Redis

```
DEL 20
```

Código 18: comando de alteração de registro por atributo identificador no Neo4j

```
MATCH (t:tabela_a) where ID(t)=1
OPTIONAL MATCH (t)-[r]-()
DELETE r,t
```

Figura 5: Gráfico gerado a partir do resultado dos testes de remoção por atributo identificador



Fonte: dados da pesquisa.

Os testes de remoção por atributo identificador apresentam resultados com uma constância semelhante à dos testes de atualização por chave primária. Nota-se, neste, que o SGBD Cassandra tem os piores resultados em maior parte dos casos, sendo ultrapassado somente pelo Neo4j quando a massa de dados aumenta.

Os resultados detalhados dos testes para cada cenário apresentado anteriormente estão disponíveis em: https://drive.google.com/file/d/13o9UrQhk609l0E0K-sxevCAKXDCHFRq3/view?usp=sharing.

CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi comparar o desempenho dos sistemas gerenciadores de banco de dados MongoDB, Redis, Cassandra e Neo4j, e medir, consequentemente, o desempenho de cada categoria de SGBDs NoSQL em relação às operações transacionais de inserção, alteração, busca e exclusão. Com os resultados obtidos, foi visto que, apesar de o MongoDB ter em parte considerável dos testes um desempenho melhor, ainda há cenários em que os SGBDs das outras categorias se tornam uma boa opção, chegando a ter, em certos momentos, um desempenho superior em relação ao MongoDB.

Como esperado, levando em consideração outros trabalhos relacionados, o MongoDB teve um desempenho significativamente melhor em relação aos outros SGBDs utilizados, uma vez que obteve, em 65% dos testes, a menor latência. Apesar disso, os SGBDs Cassandra e Neo4j também realizaram perfeitamente as buscas com os 2GB de memória RAM alocados para a máquina virtual, comprovando a premissa do NoSQL – abrir mão da consistência em favor da disponibilidade, velocidade e tolerância a partição – e sendo inteiramente capazes de lidar com um grande volume de dados sem precisar possuir uma configuração de *hardware* fora do comum para uma máquina.

A principal contribuição deste estudo é permitir um maior entendimento sobre as vantagens e limitações dos mais infleuntes bancos que existem dentro das

categorias de SGBDs NoSQL quando se trata de desempenho. Além disso, comprovar que, apesar de buscarem atender o mesmo objetivo geral, eles são diferentes em funcionalidades e configurações. Portanto, os usuários que pretendem migrar suas aplicações para o NoSQL devem primeiro considerar vários parâmetros, como linguagem de consulta, interface, disponibilidade, redundância e consistência e analisar os prós e contras de vários modelos de dados antes de escolher um modelo de dados específico.

Nesse sentido, não há soluções perfeitas e não existem SGBDs NoSQL ruins. Determinar qual SGBD melhor se encaixa em um cenário individual depende dos requisitos específicos do próprio sistema. Os testes realizados pelo autor mostram que, em cenários diferentes, soluções têm resultados diferentes. Existem muitas soluções NoSQL disponíveis, cada uma com suas próprias forças e fraquezas, diferindo-se em quase todos os aspectos, seja a estrutura dos dados salvos, consultas, desempenho e/ou ferramentas disponíveis. Uma vez que o NoSQL é uma abordagem bastante atual e, de certa forma, imprevísivel, novos SGBDs podem surgir repentinamente e assumir o posto de algum dos utilizados no desenvolvimento deste no *ranking* de popularidade. Na medida em que se definem os líderes de cada uma das categorias, permanece a necessidade de manter um comparativo atualizado sobre o desempenho dos sistemas emergentes, considerando também a evolução dos bancos já estabelecidos.

Dessa forma, para trabalhos futuros, sugere-se a análise de outros conjuntos de SGBDs, fazendo com que a área seja cada vez mais explorada. Sugere-se, também, que seja realizada uma comparação entre os SGBDs utilizando outras operações transacionais ou alterações das utilizadas neste, como feito por Amanda Prado, que em seu trabalho utilizou buscas, remoções e alterações por campos de descrição e valor. Outra proposta para trabalhos futuros é realizar novamente esta análise de desepenho em ambientes computacionais diferentes do utilizado, tais como em nuvem e/ou em uma plataforma distribuída.

REFERÊNCIAS

CHANG, F. *et al.* **Bigtable:** a distributed storage system for structured data. Disponível em: https://static.googleusercontent.com/media/research.google.com/pt-BR//archive/bigtable-osdi06.pdf. Acesso em:17 mar. 2019.

COSTA, Leandro Teodoro. **Conjunto de características para teste de desempenho:** uma visão a partir de ferramentas. 2012. 113f. Dissertação de Mestrado – Pós-Graduação em Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2012. Disponível em: http://tede2.pucrs.br/tede2/bitstream/tede/5186/1/440236.pdf. Acesso em: 19 maio 2019.

CUER, Eder dos Santos. **Comparação de desempenho de bancos de dados SQL e NOSQL**. 2014. 97 f. Trabalho de Conclusão de Curso (Bacharel) – Graduação Ciência da Computação, Centro Universitário Eurípedes de Marília, Marília, 2014. Disponível em: https://aberto.univem.edu.br/bitstream/handle/ 11077/ 999/Eder%20dos%20Santos%20Cuer.pdf?sequence=1&isAllowed=y. Acesso em: 20 jun. 2019.

DECANDIA, G. *et al.* **Dynamo:** Amazon's highly available key-value store. Disponível em: https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf. Acesso em: 17 mar. 2019.

DEMCHENKO, Y. *et al*. **Addressing big data issues in scientific data infrastructure.** Disponível em: https://www.researchgate.net/publication/256082290_Addressing_Big_Data_Issues_in_Scientific_Data_Infrastructure. Acesso em: 13 mar. 2019.

GESSERT, Felix; RITTER, Norbert. Scalable data management: NoSQL data stores in research and practice. In: Data Engineering (ICDE), 2016. **IEEE 32nd International Conference on IEEE**, 2016. p. 1420-1423.

HAN, Jing *et al*. **Survey on NoSQL databases.** Disponível em: http://bit.ly/2KVSlCZ. Acesso em: 24 jun. 2019.

LEAVITT, N. **Will NoSQL databases live up to their promise?** Disponível em: http://www.sciweavers.org/ read/ will-nosql-databases-live-up-to-their-promise-246913. Acesso em: 21 jun. 2019.

NAYAK, Ameya *et al.* **Type of NOSQL** Databases and its comparison with relational databases**.** Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.3372&rep= rep1&type=pdf. Acesso em: 09 abr. 2019.

NOSQL. **NOSQL databases.** Disponível em: http://nosql-database.org/. Acesso em: 29 abr. 2019.

OUSOUSS A.; BENJELLOUN F. Z.; LAHCEN A. A.; BELFKIH S. **Comparison and classification of NoSQL databases for big data.** Disponível em: https://www.researchgate.net/publication/278963532_ Comparison_and_Classification_of_NoSQL_Databases_for_Big_Data. Acesso em: 20 maio 2019.

POPESCU, A. **Presentation:** NoSQL at CodeMash - an interesting NoSQL categorization**.** Disponível em: http://nosql.mypopescu.com/post/396337069/presentation-nosql-codemashaninteresting-nosql. Acesso em: 11 maio 2019.

PRADO, Amanda L. **Comparação de desempenho entre banco de dados relacional E NOSQL.** 2019. 10 f. Trabalho de Conclusão de Curso (Bacharel) – Graduação em Ciência da Computação, Faculdade Governador Ozanam Coelho, Ubá, 2019.

SHARMA, Sugam *et al***. Leading NoSQL models for handling Big Data:** a brief review. Disponível em: https://www.researchgate.net/publication/270273828_Leading_NoSQL_models_for_handling_Big_Data_A_brief_review. Acesso em: 11 jun. 2019.

SANTOS, Wesley J.; BONETTI, Tiago P. **NoSql:** uma alternativa ao tradicional modelo relacional. Disponível em: http://web.unipar.br/~seinpar/site/publicacao/Wesley_Jos%C3%A9_dos_%20 Santos.pdf. Acesso em: 11 jun. 2019.

TIWARI, S. **Professional NoSQL.** Indianapolis: John Wiley & Sons, Inc. 2011.