Teste de performance em bancos de dados NoSQL: Apache Cassandra vs. Neo4j

Amanda V. Soares[1], João Vítor F. Sonego[1], Leonardo O. Spilere[1]

[1]Curso de Ciência da Computação– Universidade do Extremo Sul Catarinense (UNESC)

Abstract. *In the current technological scenario, "Big Data" redefines database systems, in the need of handling massive volumes and multiple data sources. To overcome limitations, organizations started to adopt NoSQL databases to enable large-scale data analysis and storage. This study compares response times between Apache Cassandra and Neo4j for data insertion, queries, updates, and deletes with single client requests and multiple clients making simultaneous requests. The results indicate that Cassandra outperforms Neo4j in terms of response time across all evaluated tests.*

Resumo. *No cenário tecnológico atual, o "Big Data" redefine sistemas de banco de dados, lidando com volumes massivos e múltiplas fontes de dados. Para superar limitações, organizações passaram a adotar bancos de dados NoSQL a fim de permitir análise e armazenamento de dados em larga escala. O presente estudo apresenta comparações do tempo de resposta entre Apache Cassandra e Neo4j para realização de inserção, consultas, atualizações e remoções de dados considerando um único usuário e múltiplos clientes realizando requisições simultâneas. Os resultados sugerem que o Cassandra possui melhor desempenho que o Neo4j com relação ao tempo de resposta em todos os testes avaliados.*

1. Introdução

No cenário dinâmico da tecnologia da informação, o termo "Big Data" tem emergido como um divisor de águas no campo de banco de dados. À medida que a sociedade e as empresas geram uma quantidade inimaginável de informações diariamente, o gerenciamento e a análise eficaz desses dados se tornaram uma prioridade crítica. O Big Data, caracterizado por conjuntos de dados extremamente volumosos, complexos e de múltiplas fontes, está redefinindo os fundamentos dos sistemas de banco de dados tradicionais.

O conceito central por trás do Big Data é a capacidade de coletar e processar grandes volumes de informações de forma eficiente e escalável. Essa abordagem não se limita apenas ao armazenamento de dados, mas também à extração de insights valiosos a partir deles. Com a rápida evolução da tecnologia, os bancos de dados tradicionais estão encontrando limitações na manipulação desses gigantescos repositórios de informações, o que gerou a necessidade de soluções mais flexíveis e inovadoras (MELO, 2016).

À medida que as organizações buscam tirar proveito do Big Data, estão adotando novos paradigmas, como bancos de dados *Not Only SQL* (NoSQL), que são projetados para lidar com a variedade e a velocidade das informações. Esses sistemas desafiam as estruturas rígidas dos bancos de dados relacionais e abrem caminho para a exploração de

novas formas de armazenamento e análise de dados em larga escala (SADALAGE e FOWLER 2013).

Entretanto, Toth (2011) destaca que a falta de benchmarks sólidos pode tornar desafiador para os profissionais de tecnologia e desenvolvedores tomarem decisões embasadas sobre a escolha de sistemas NoSQL. Portanto, é crucial enfatizar a importância de estudos de benchmarking e análises mais aprofundadas na avaliação do desempenho desses sistemas

1.1. Objetivo

O presente trabalho tem como objetivo comparar a performance do tempo de execução de inclusões, atualizações, remoções e consultas em dois bancos de dados NoSQL.

1.2. Banco de dados NoSQL

Os bancos de dados NoSQL são uma categoria de sistemas de gerenciamento de banco de dados que se destacam por sua abordagem não relacional. Diferentemente dos bancos de dados relacionais tradicionais, que usam tabelas com linhas e colunas, os bancos de dados NoSQL utilizam diversos modelos de dados, como documentos, grafos, colunas e chave-valor. Essa flexibilidade permite que eles armazenem e processem dados de forma mais ágil e escalável, tornando-os ideais para aplicações que envolvem grandes volumes de informações não estruturadas ou semiestruturadas.

O Teorema CAP afirma que sistemas distribuídos escolhem entre três características que podem entregar aos usuários, formando a sigla (IBM, s.d.):

- Consistência (*Consistency*) - em que os dados retornados serão os mesmos para todos os usuários quando vistos ao mesmo tempo;
- Disponibilidade (*Availability*) - em que os usuários sempre receberão um retorno, mesmo que o sistema esteja parcialmente inacessível;
- Tolerante a Falhas/Partição (*Partition tolerance*) - em que o sistema continua o processamento independentemente de quantas falhas de comunicação ocorram durante o processo.

1.3. Objetos de Estudo

1.3.1. Apache Cassandra

O Apache Cassandra é um sistema de banco de dados distribuído e descentralizado, podendo ser operado em múltiplas máquinas para garantir confiabilidade e eliminar pontos de falha, proporcionando alta disponibilidade. A escalabilidade horizontal eficiente permite a adição de nós conforme a demanda, tornando-o ideal para sistemas com cargas de trabalho variáveis. Além disso, o Cassandra adota um modelo de dados orientado a colunas, otimizado para consultas eficientes e alto desempenho em cenários de leitura intensiva (ANICETO e XAVIER, 2014).

Considerando o teorema CAP, o Cassandra opta pela disponibilidade e pela tolerância a falhas, oferecendo uma consistência ajustável, permitindo aos desenvolvedores escolher o nível de consistência dos dados com base em suas necessidades específicas, equilibrando entre desempenho e consistência.

1.3.2. Neo4j

O Neo4j (2023) é um sistema de gerenciamento de banco de dados orientado a grafos, que se destaca como uma alternativa aos tradicionais bancos de dados relacionais. Enquanto os sistemas de banco de dados relacionais armazenam dados em tabelas com linhas e colunas, o Neo4j armazena informações em estruturas de grafo e relacionamentos. Essa abordagem é especialmente eficaz quando se lida com dados altamente interconectados, sendo destacado nas "Top 10 Reasons for Choosing Neo4j for Your Graph Database" (2023), como um modelo adequado para representar relacionamentos complexos, apresentando escalabilidade, distribuição eficiente de dados em clusters e flexibilidade nos nós que podem conter propriedades variadas.

O Neo4j, portanto, se destaca por sua capacidade de modelar e consultar relacionamentos de forma eficaz, priorizando, de acordo com a ObjectRocket (2018), a disponibilidade e consistência. Isso o torna uma escolha popular por sua versatilidade para projetos que dependem de dados altamente interconectados.

2. Trabalhos Correlatos

De Diana e Gerosa (2010) descreveram em seu artigo as principais características dos bancos de dados NoSQL e suas diferenças com banco de dados relacionais, relacionando as características e diferenças no teorema CAP, ACID e BASE. Além disso, descreve, também, os principais modelos de dados mais comuns utilizados em bancos NoSQL.

Partindo da premissa que bancos de dados NoSQL priorizam o conceito de disponibilidade em comparação com consistência, Diogo, Cabral e Bernardino (2019) objetivaram com seu estudo comparar e analisar o modelo de consistência de dados implementados por cinco bancos de dados NoSQL.

Já o estudo realizado por Abramova e Bernardino (2013) descreve as principais características, tipos e princípios de banco de dados NoSQL com um foco sobre os bancos MongoDB e Apache Cassandra. Ainda, realiza experimentos a fim de comparar o tempo de execução de inserções, atualizações e consultas em ambos os bancos considerando diferente quantidade de dados.

Oliveira, De Morais e De Freitas (2018) realizaram um estudo executando inserções, atualizações, remoções e consultas em três banco de dados do tipo NoSQL e compararam os tempos de execução para cada uma dessas tarefas para cada banco de dados estudado. Ainda, realizaram o estudo simulando em dois formatos: um único usuário realizando consultas em sequência e múltiplos usuários realizando diferentes consultas simultaneamente (threads).

Ademais, tratando de um caso com finalidade específica, Amghar, Cherdal e Mouline (2022) realizaram experimentos em cinco banco dados NoSQL, com diferentes modelos de armazenamento de dados, para avaliar o mais propício para armazenamento de *Tweets*.

3. Metodologia

A fim de alcançar o objetivo proposto e baseado nos estudos de Oliveira, De Morais e De Freitas (2018), será elaborada uma REST API em C# contendo cinco métodos (*endpoints*) idênticos para cada banco a ser avaliado. Sendo assim, será possível

realizar inserção (POST), atualização (PUT), remoção (DELETE), consulta (GET) e consulta com identificador (GET by id). Ainda, ressalta-se que a API será implementada sem considerar nenhum tratamento para requisições ou respostas inválidas, servirá apenas como meio para receber a requisição, solicitar a informação ao banco e reencaminhar a resposta ao cliente.

As requisições para esta API serão realizadas utilizando Apache JMeter em dois cenários de clientes distintos. O primeiro, em que um único usuário realizará 1000 requisições em sequência para cada *endpoint*. Já no segundo, 600 usuários virtuais (threads) realizarão consultas simultâneas, sendo que cada usuário realizará 5 requisições (iterações) a cada 1 segundo para cada *endpoint*, totalizando 3000 requisições. A partir disso, espera-se que as requisições com um único cliente sejam respondidas em menor tempo quando comparadas com o cenário de múltiplos clientes.

As informações e requisições serão preenchidas com informações aleatórias pelo JMeter, com exceção da propriedade 'Id' que será preenchida de maneira sequencial. Em seguida, serão comparados os tempos necessários para obtenção da resposta entre os bancos analisados para cada método implementado e para os dois cenários de clientes. Para isso, serão analisadas a média, mediana e o T90% (tempo de conclusão de 90% das requisições).

Os testes foram realizados em ambiente local a fim de mitigar a influência do tráfego a internet e utilizando computador com as seguintes especificações:

- CPU: AMD Ryzen 5 3500X
- RAM: 16GB
- SSD: 240GB
- Sistema operacional: Windows 10

3.1 Modelo de dados

Em ambos os objetos de estudo, as propriedades dos dados a serem armazenadas serão idênticas e seguem o modelo presente no Quadro 1. Com isso, cada grafo inserido no Neo4j possuirá as mesmas propriedades que a tabela criada no Apache Cassandra para inserção dos dados.

Quadro 1 - Modelo das propriedades dos dados

Propriedade	Tipo da propriedade		
	C#	Cassandra	Neo4j
Id	int	int (PK)	integer
Texto	string	text	string
Numero	int	int	integer
Num_Decimal	float	float	float
Data	DateTime	timestamp	Local DateTime

(Fonte: Elaborado pelos autores).

4. Resultados

A partir dos métodos supracitados e da execução dos testes no ambiente descrito na seção anterior, os resultados de cada banco foram dispostos em gráficos comparando-os para um mesmo método de requisição e cenário de clientes.

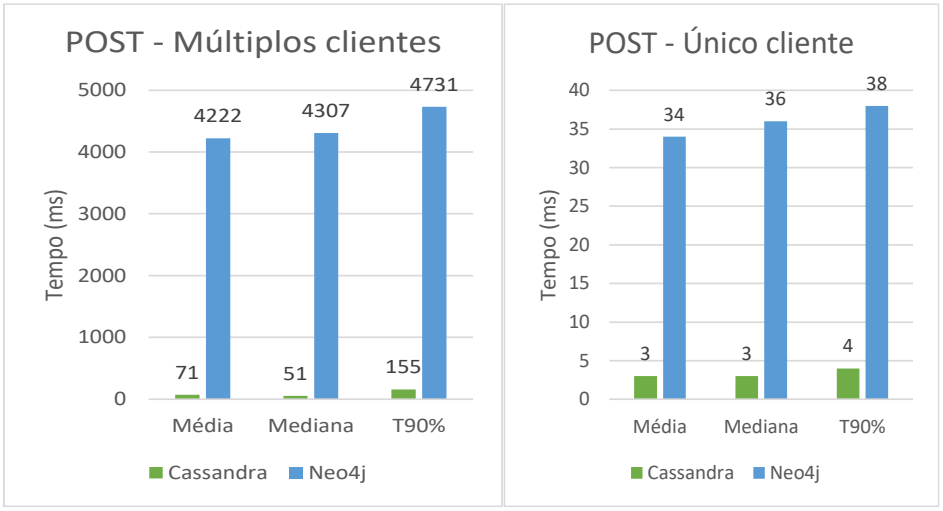

Figura 1 – Tempos de respostas para inserções.
(Fonte: Elaborado pelos autores).

Com relação às inserções presentes na Figura 1, pode-se observar que o Cassandra apresentou tempos mais baixos tanto para os cenários com múltiplos clientes quanto com apenas um único cliente, necessitando 16,8% (múltiplos) e 8,8% (único) do tempo médio necessário pelo Neo4j para a mesma requisição.

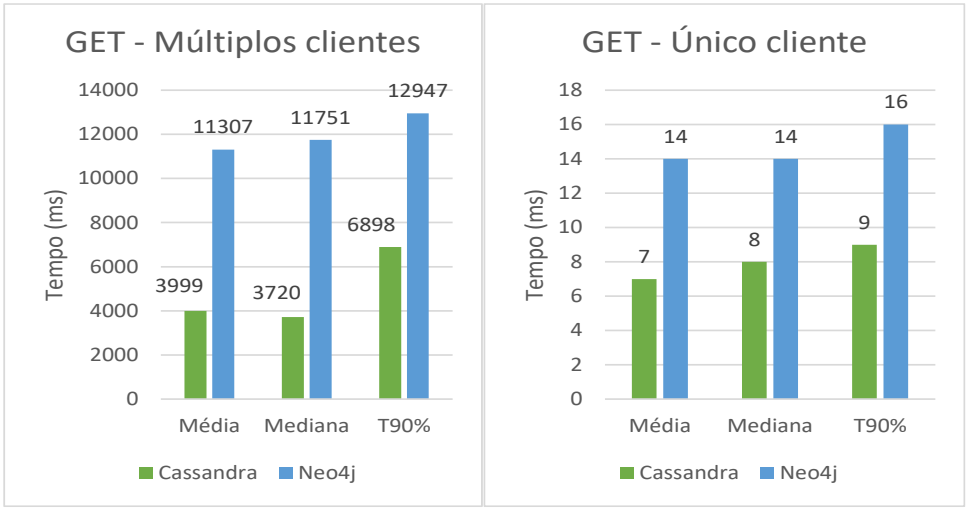

Figura 2 – Tempos de respostas para consultas.
(Fonte: Elaborado pelos autores).

Já para os casos de consulta (Figura 2) e consulta por identificador (Figura 3), o Cassandra continuou apresentando tempos inferiores para ambos os cenários, assim como nos casos de inserção. Contudo a diferença entre ambos os bancos foi menor para este tipo de requisição, com exceção do cenário de consultas por identificados com múltiplos clientes. Observa-se que o Cassandra necessitou de 35,4% (múltiplos) e 50,0% (único)

do tempo médio necessário pelo Neo4j para consultas e 13,0% (múltiplos) e 25,0% (único) para consultas com identificador.

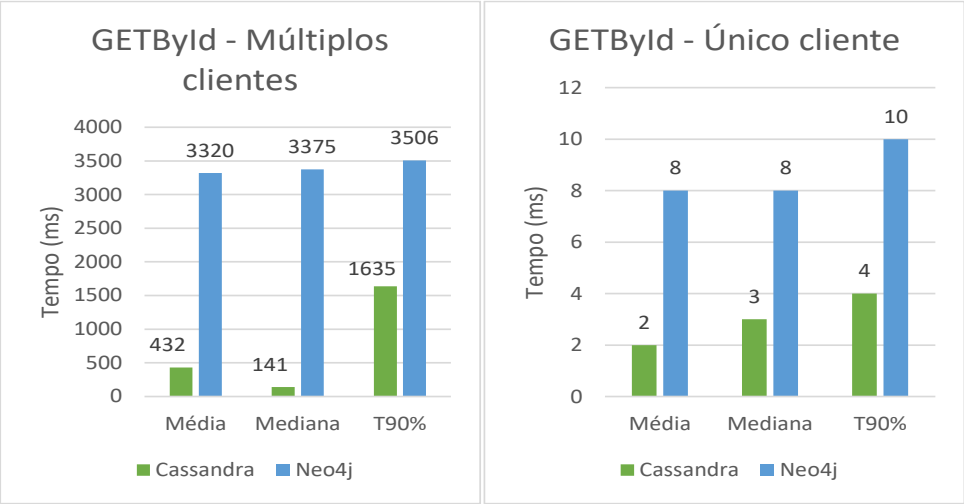

Figura 3 – Tempos de respostas para consultas por identificador.
(Fonte: Elaborado pelos autores).

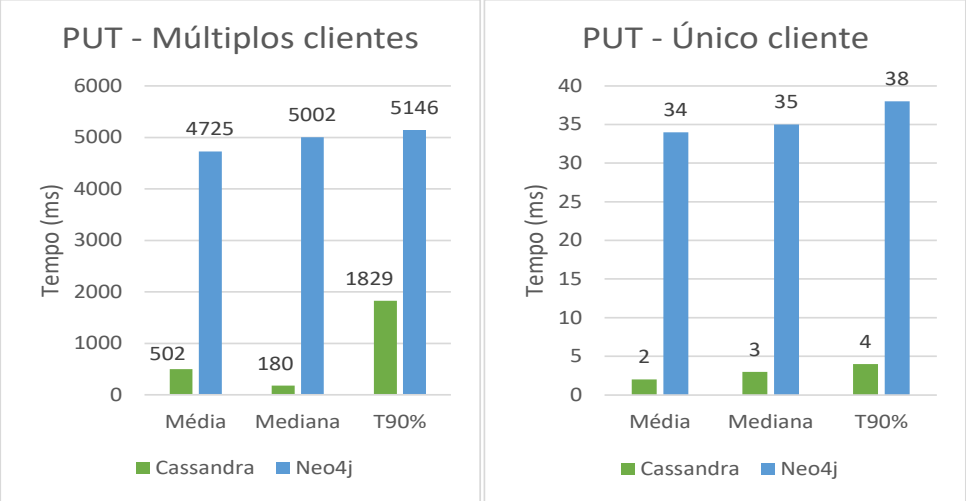

Figura 4 – Tempos de respostas para atualizações.
(Fonte: Elaborado pelos autores).

Já com relação às atualizações (Figura 4) e remoções (Figura 5), os resultados foram novamente similares aos da inserção, onde o Cassandra apresentou 10,6% (múltiplos) e 5,9% (único) do tempo médio necessário pelo Neo4j para atualizações e 11,6% (múltiplos) e 8,3% (único) para remoções.

Por fim, observa-se que os tempos necessários para realização das inserções no cenário com um único cliente são inferiores quando comparados com o cenário de múltiplos clientes para ambos os bancos avaliados para todas as requisições testadas.

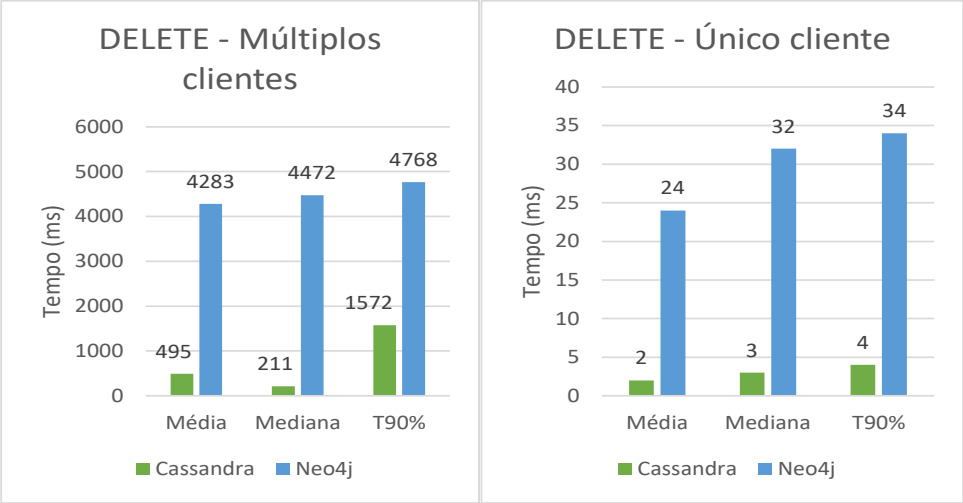

Figura 5 – Tempos de respostas para remoções.
(Fonte: Elaborado pelos autores).

5. Conclusões

Dos resultados apresentados, fica evidente que o cenário de um único usuário realizando requisições resulta em tempos de resposta inferiores quando comparado ao cenário com múltiplos clientes, tanto no Apache Cassandra quanto no Neo4j. Além disso, o banco de dados Apache Cassandra mostrou consistentemente tempos de resposta mais rápidos em todas as requisições e cenários analisados neste estudo, indicando claramente sua vantagem em termos de desempenho.

Entretanto, é importante ressaltar que este estudo se baseou em um caso genérico, exigindo uma análise mais aprofundada para casos específicos, já que apenas uma parte das funcionalidades de cada banco foi testada. Ainda, é recomendável avaliar a influência dos drivers de comunicação entre os bancos de dados e a API desenvolvida em C#, de visto que a escolha de diferentes drivers poderá afetar os resultados obtidos.

Por fim, em trabalhos futuros, sugere-se a implementação deste estudo em uma máquina projetada especificamente para servidores, proporcionando um ambiente ideal para avaliar o desempenho dos bancos de dados. Além disso, é fundamental realizar testes com conjuntos de dados maiores para investigar a influência do volume de dados no tempo de resposta. Essa abordagem permitiria uma análise mais aprofundada da escalabilidade de cada banco de dados, propriedade inerente de bancos NoSQL.

6. Referências

ABRAMOVA, Veronika; BERNARDINO, Jorge. NoSQL databases: MongoDB vs cassandra. In: Proceedings of the international C* conference on computer science and software engineering, p. 14-22, 2013.

AMGHAR, Souad; CHERDAL, Safae; MOULINE, Salma. Storing, preprocessing and analyzing tweets: finding the suitable noSQL system. International Journal of Computers and Applications, v. 44, n. 6, p. 586-595, 2022.

ANICETO, Rodrigo Cardoso; XAVIER, Renê Freire. Um estudo sobre a utilização do banco de dados NoSQL Cassandra em Dados Biológicos. 2014.

DE DIANA, Mauricio; GEROSA, Marco Aurélio. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. In: Workshop de Teses e Dissertações de Bancos de Dados do Simpósio Brasileiro de Bancos de Dados WTDBD2010, 2010.

DIOGO, Miguel; CABRAL, Bruno; BERNARDINO, Jorge. Consistency models of NoSQL databases. Future Internet, v. 11, n. 2, p. 43, 2019.

IBM. What is the CAP Theorem? IBM. Disponível em: <https://www.ibm.com/topics/cap-theorem>. Acesso em: 4 nov. 2023.

MELO, Tiézer Costa de. Estudo de caso de bancos NOSQL no contexto de big data, 2016.

NEO4J. What Is a Graph Database and Property Graph - Neo4j. Disponível em: <https://neo4j.com/developer/graph-database/>. Acesso em: 4 nov. 2023.

NEO4J. Top 10 Reasons for Choosing Neo4j for Your Graph Database. Disponível em: <https://neo4j.com/top-ten-reasons/>. Acesso em: 5 nov. 2023.

OBJECTROCKET. Using a Polyglot Database Strategy, 2018. Disponível em: <https://www.objectrocket.com/blog/development/using-a-polyglot-database-strategy/>. Acesso em: 6 nov. 2023.

PEREIRA, Diogo Augusto; DE MORAIS, Wagner Ourique; DE FREITAS, Edison Pignaton. NoSQL real-time database performance comparison. International Journal of Parallel, Emergent and Distributed Systems, v. 33, n. 2, p. 144-156, 2018.

SADALAGE, P.J.; FOWLER, M. NoSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota Essencial. 3. ed. São Paulo: Novatec Editora, 2013.

TOTH, Renato Molina. Abordagem NoSQL: uma real alternativa. Universidade Federal de São Carlos. Sorocaba, 2011.