Comparação entre bancos de dados relacionais e bancos de dados orientados a documentos

Renan Scarela, Vitor Santos, Wenderson Alexandre

***Resumo*—Uma das consequências do avanço da tecnologia é o aumento exponencial da quantidade de informação gerada e armazenada pela população de usuários. Comparadas a aplicações passadas, aplicações atuais têm de lidar com cargas muito maiores de dados, sendo necessárias consultas extremamente complexas para cruzamento e análises de informações sob diferentes cenários. Tal crescimento acaba por expor certas limitações nos predominantes bancos de dados relacionais, levando desenvolvedores a adotarem a tecnologia emergente dos bancos de dados não-relacionais. Sendo estes compostos por diversos paradigmas, este artigo tem como foco a comparação de performance entre o paradigma orientado a documentos e o paradigma relacional. Para tal, foram realizados testes de diversas operações, como consultas, inserções, atualizações e exclusões. Os resultados obtidos são apresentados através de gráficos tridimensionais que possibilitam a visualização do comportamento das operações de acordo com o crescimento dos dados.**

***Palavras-Chave*—Bancos de dados, Gráficos tridimensionais, NoSQL, Orientação a documentos, SQL.**

I. Introdução

ATUALMENTE, o armazenamento de informações é essencial no dia-a-dia da população. Com o avanço da tecnologia e da conectividade, as pessoas estão constantemente realizando acessos a diferentes mídias de diferentes dispositivos, como computadores, smartphones, tablets, Smart TVs, entre outros.

Consequentemente, o volume de informações armazenadas vem crescendo de forma irrefreável. De acordo com Eric Schmidt, ex-CEO do Google, a população gera, em apenas dois dias, o equivalente a cinco bilhões de gigabytes, uma quantidade infinitamente maior que as primeiras métricas realizadas, datadas de 2003 [3].

Por muitos anos os bancos de dados SQL se mantêm como principal sistema de armazenamento utilizado por corporações. Contudo, o constante crescimento no volume de informações, somados a estruturas de dados cada vez mais complexas, acabaram por expor algumas limitações nos bancos de dados relacionais, tais como capacidade de hardware, armazenamento de estruturas dinâmicas, entre outros desafios [7].

No final da década de 1990, os bancos de dados não-relacionais, conhecidos por NoSQL, foram criados sugerindo um afastamento dos bancos de dados SQL. Com um boom significativo em 2009, estes se tornam cada vez mais populares [7].

Diversos bancos de dados não-relacionais foram criados atendendo a diferentes problemas, como flexibilidade de estruturas de dados, escalabilidade horizontal, estruturação de relacionamentos em forma de grafos, entre outros.

Os bancos de dados orientados a documentos vêm se popularizando nos últimos anos, preenchendo algumas das limitações dos bancos de dados relacionais, como escalabilidade, e apresentando uma estrutura de dados flexível para o armazenamento das informações.

Tais fatores levam a comparar, em diferentes aplicações, os bancos de dados orientados a documentos e os populares bancos de dados relacionais. E embora essa comparação não tenha sido especificamente abordada por outros autores, alguns estudos foram feitos para se comparar bancos de dados relacionais e não-relacionais, a fim de analisar as vantagens de um sobre o outro.

Ian Thomas Varley [6] diz que estamos longe de dizer que há um vencedor entre os dois tipos de bancos de dados. Ele cita o fato de bancos relacionais possuírem uma grande supremacia por seus fundamentos matemáticos para estabelecer relações, sua estrutura voltada para a modelagem de dados e sua engenharia que faz com que seja capaz de obter um bom desempenho na maioria das situações. Apesar disso, Varley afirma que bancos de dados não-relacionais primam pela eficiência e escalabilidade ao invés de se preocupar fortemente com o modo como os dados são acessados, o que acaba sendo um grande benefício para a obtenção de um desempenho melhor.

Uma Bhat e Shraddha Jadhav [8] são mais entusiastas em relação aos bancos de dados não-relacionais. Com base em análises estatísticas, citam como vantagens a integridade, escalabilidade, robustez, maior velocidade de computação e balanceamento de carga. Para Bhat e Jadhav, bancos de dados não-relacionais podem significar uma revolução na era moderna de supercomputadores por serem versáteis e centrados em recursos.

Ameia Nayak, Anil Poriya e Dikshay Poojary [9], além de apontarem vantagens dos bancos de dados não-relacionais normalmente citadas por outros autores como escalabilidade, rapidez e flexibilidade, também enumeram diversos pontos

Renan Scarela e Vitor Silva de Carvalho Santos são graduandos do curso de Ciência da Computação da Universidade Anhembi Morumbi (e-mails: renan.scarela@gmail.com, vitorsdcs@outlook.com).

Wenderson Alexandre de Sousa Silva é professor mestre na Escola de Engenharia e Tecnologia da Universidade Anhembi Morumbi (e-mail: wenale@gmail.com)

negativos, dentre eles a imaturidade, a não padronização de linguagem (como os relacionais possuem a SQL), o fato de alguns bancos NoSQL não seguirem o modelo ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*) e a dificuldade de manutenção.

Com muitos estudos buscando fazer comparações que envolvam os bancos de dados não-relacionais como um todo, este artigo irá manter seu foco apenas em bancos de dados orientados a documentos, de forma a obter uma análise de performance que não seja afetada por outros paradigmas de bancos de dados não-relacionais.

Este artigo está estruturado da seguinte maneira: o capítulo II apresenta os bancos de dados relacionais, mostrando sua estrutura, conceitos e linguagem. O capítulo III explica o funcionamento dos bancos de dados não-relacionais. O capítulo IV é uma abordagem dos bancos de dados orientados a documentos, trazendo seus conceitos e detalhes do seu funcionamento. O capítulo V é um descritivo sobre a metodologia e técnicas aplicadas para a resolução do problema proposto pelo trabalho. No capítulo VI são apresentadas as análises de desempenho envolvendo os tipos de bancos de dados apresentados nos capítulos 2 e 4 e seus respectivos resultados. Por fim, o capítulo VII traz as conclusões obtidas no desenvolvimento do trabalho e o que se pode afirmar com base nas análises realizadas no capítulo VI. Também são citadas recomendações para trabalhos futuros.

II. BANCOS DE DADOS RELACIONAIS

No início da década de 1970 foi proposta uma nova estrutura de representação de dados chamada "modelo de dados relacional", que veio a ser um marco histórico no desenvolvimento de sistemas de bancos de dados [4].

Conceitualmente, um banco de dados relacional armazena informações em forma de tabelas. A descrição dos dados contidos nessas tabelas é chamada de esquema (ou schema). O esquema de uma tabela especifica seu nome, o nome de cada coluna e seus respectivos tipos. Por exemplo, para armazenarmos informações de alunos de uma universidade, podemos definir o seguinte esquema:

Alunos (id-aluno: string, nome: string, login: string, idade: integer, média: real)

O esquema acima descreve quais informações serão armazenadas para cada aluno da universidade. A tabela criada a partir deste esquema consistirá de linhas (ou registros), e cada linha nesta tabela descreve um aluno.

Características importantes incluem, também, a utilização de chaves (primárias e estrangeiras) e o suporte a transações, fundamentadas nos princípios ACID [4].

As propriedades ACID são assim descritas:

- Atomicidade: Cada transação deve ser atômica: ou todas ações são executadas ou nenhuma delas é executada;
- Consistência: As transações devem preservar a consistência do banco de dados;
- Isolamento: Diferentes transações não podem sofrer com o efeito de outras que estejam no mesmo plano de execução;
- Durabilidade: A conclusão de uma transação deve persistir mesmo que o sistema falhe antes que as alterações sejam refletidas em disco.

Um sistema de gerenciamento de banco de dados, ou SGBD, é o *software* que auxilia na manipulação e manutenção de dados.

De forma a definir esquemas de tabelas, manipular seus dados e controlar o privilégio de objetos de um SGBD, é necessário se comunicar com o banco de dados através de uma linguagem. No caso dos bancos de dados relacionais, a mais popular é chamada SQL. Ela é específica para bancos de dados relacionais e não é utilizada fora deste contexto [3].

Segundo Alex Kriegel [3], a SQL é dividida em três áreas que englobam operações para a interação com o banco de dados:

- *Data Definition Language* (DDL) – Linguagem de Definição de Dados: Quando se faz necessário criar, estruturar, modificar e excluir objetos dentro de um SGBD, são utilizados recursos provenientes da DDL, que possui comandos como CREATE, ALTER e DROP;
- *Data Manipulation Language* (DML) – Linguagem de Manipulação de Dados: Como o próprio nome diz, a DML possui comandos que servem para manipular dados contidos numa estrutura, como os comandos INSERT, UPDATE e DELETE;
- *Data Query Language* (DQL) – Linguagem de Consulta de Dados: Algumas vezes incluída na DML por alguns autores, a DQL possui um único e poderoso comando: o SELECT.

III. BANCOS DE DADOS NÃO-RELACIONAIS

Com o avanço da tecnologia houve mudanças significativas no desenvolvimento de *software*. Aplicações, outrora centralizadas, tornaram-se soluções distribuídas e escaláveis, isto é, capazes de estarem preparada para o crescimento de dados.

Apesar de os conservadores bancos de dados relacionais serem extremamente robustos e maduros, apresentam algumas limitações, principalmente no que diz a respeito à performance em ambientes distribuídos.

Bancos de dados não-relacionais, por sua vez, trazem características e funcionalidades que buscam suprir as limitações do modelo relacional [7].

O termo NoSQL, cujo significado é "Not Only SQL" (não apenas SQL, em tradução livre), evidencia o afastamento do paradigma relacional.

Segundo Silvan Weber [7] bancos de dados NoSQL, por definição, devem ser não-relacionais, distribuídos e horizontalmente escaláveis, sendo caracterizados, respectivamente, de modo que o banco de dados não-relacional:

- Não deve seguir o modelo relacional, ou seja, não devem existir relacionamentos entre os dados;

- Deve estar apto a replicação de dados, de modo que possa armazenar dados em múltiplos servidores (cluster);
- Deve ser, preferencialmente, de código aberto. Weber acrescenta que a regra não deve ser seguida à risca, visto que existem bancos de dados NoSQL proprietários e pagos, como o DynamoDB (Amazon Web Services);
- Deve estar disponível em múltiplos servidores e se torna mais performático a cada servidor acrescentado no *cluster*. Uma característica essencial, uma vez que bancos de dados SQL não são escaláveis de forma nativa.

Rick Cattell [1] afirma que, além das características citadas anteriormente, também é fundamental que:

- As interfaces de comunicação (linguagens) sejam simplistas, contrastando a linguagem SQL;
- Os modelos de concorrência e consistência sejam menos rígidos que os modelos apresentados nos bancos de dados relacionais (ACID);
- Haja a capacidade de acrescentar, dinamicamente, novos atributos aos dados armazenados.

Segundo Cattell, bancos de dados NoSQL não seguem as propriedades fundamentadas no ACID. Para tal, foi definido o acrônimo BASE (*Basically Available*, *Soft state*, *Eventually consistent*).

Assim, bancos de dados não-relacionais possuem altíssima disponibilidade (*basically available*) e, ao ocorrerem atualizações, estas são eventualmente propagadas aos demais servidores (*eventually consistent*). Contudo, o estado após uma transação não é mais um estado sólido, de forma que a garantia de consistência seja eventual e limitada (*soft state*).

A ideia fundamentada no conceito BASE, totalmente oposta ao ACID, é que abrir mão da consistência proposta no ACID viabilize atingir alta performance e garantir a escalabilidade [1].

É importante ressaltar que, a partir do fundamento proposto acima, a escolha de um sistema de banco de dados, para uma determinada aplicação, não deve ocorrer de forma leviana, sendo necessária uma avaliação de propósitos e necessidades.

Apesar de a escalabilidade ser uma característica fundamental em bancos de dados não-relacionais e, como proposto no conceito BASE, solucionar a dificuldade de escalabilidade presente em bancos de dados relacionais, trabalhar com bancos de dados distribuídos podem levar a novos problemas. Para avaliar o possível impacto do uso de um banco de dados não-relacional, é necessária uma visão geral do sistema que será ser implementado [6].

Segundo Varley, Eric Brewer [10] propôs o teorema CAP (*Consistency*, *Availability*, *Partition tolerance*), definindo que um sistema de banco de dados distribuído não pode conter, simultaneamente, as três seguintes dimensões:

1. Consistência: o mesmo dado estará homogêneo em todos os nós do cluster;
2. Disponibilidade: os dados sempre estarão disponíveis;
3. Tolerância a particionamento: caso ocorram falhas em parte dos nós do cluster, durante uma operação, ainda assim a mesma é concluída corretamente.

Varley acrescenta ainda que grande parte dos sistemas de bancos de dados não-relacionais assumem, por natureza, a tolerância a particionamento. Contudo, ainda é necessária a escolha entre consistência e disponibilidade.

Segundo exemplificado por Weber, para um sistema bancário a consistência é fundamental: os valores financeiros devem estar idênticos em todos os nós do cluster sem qualquer tipo de atraso, evitando qualquer transação monetária inconsistente. Já no caso de um sistema online de uma rede de venda de livros, este não requer uma atualização de consistência instantânea, visto que não há grandes impactos caso o preço de um livro permaneça desatualizado, em um dos nós do cluster, por curtos períodos de tempo, o que levaria a uma preferência a disponibilidade.

Diferente de bancos de dados relacionais, os quais possuem implementações semelhantes, bancos de dados não-relacionais possuem diferentes implementações, categorizadas em diferentes paradigmas (um ou mais), cada qual buscando atender a propósitos diferentes [7].

Embora o foco deste trabalho seja os bancos de dados orientados a documentos, existem outros paradigmas de bancos de dados não-relacionais como é o caso dos orientados a grafos, orientado a colunas e chave/valor.

IV. Bancos de dados orientados a documentos

Bancos de dados orientados a documentos são bancos de dados não-relacionais facilmente escaláveis [2]. O conceito central do paradigma é a sua estrutura de dados: documentos organizados em coleções, os quais substituem linhas e colunas, e podem representados em estruturas como XML, JSON, BSON, YAML, entre outros. Os documentos apresentam estruturas de dados flexíveis, capazes de armazenar desde simples valores textuais e numéricos a estruturas complexas, como coleções e dicionários [5].

Segundo os autores Pramod J. Sadalage e Martin Fowler [5], bancos de dados orientados a documentos são muito próximos ao paradigma chave/valor, afirmando assim:

"[...] pense neles como depósitos de chave-valor, em que o valor pode ser examinado." [5].

Bancos de dados orientados a documentos, segundo Weber, são schema free ou schemaless, ou seja, não possuem esquemas definidos, o que permite que cada documento possua sua própria estrutura, um recurso extremamente útil, uma vez que documentos não são dependentes entre si [7].

Contrastando com sistemas de bancos de dados relacionais, tabelas dão lugares a coleções [5]. Segundo a autora Kristina Chodorow [2] o conceito de coleções e tabelas são extremamente próximos, onde uma coleção contém um conjunto de documentos, com estruturas similares, sob um determinado contexto. Sadalage e Fowler afirmam que, diferente de bancos de dados relacionais, documentos podem apresentar estruturas diferentes entre si e, ainda assim, pertencer a mesma coleção.

Os autores contrastam ainda as diferenças de tratamento para valores não definidos e para adição de novos atributos. No modelo orientado a documentos, a flexibilidade estrutural dos mesmos permite que valores nulos, ou em branco, simplesmente não sejam incluídos no respectivo documento, bem como permite que um novo atributo seja adicionado a um documento sem a necessidade de definição ou de alteração nos documentos já existentes naquela coleção.

Chodorow evidencia a importância da estrutura flexível dos documentos e o suporte a documentos embarcados para o dia-a-dia dos desenvolvedores. Tal estrutura se aproxima de forma extremamente natural ao modo como os desenvolvedores modelam os dados de suas aplicações em linguagens modernas, orientadas a objetos. Segundo a autora, a ausência de schemas e facilidade de remoção e adição de novos atributos trazem um grande ganho aos desenvolvedores, deixando-os livres para diversas experiências durante a modelagem da aplicação, ou seja, se torna possível experimentar, rapidamente, dezenas de modelos de dados diferentes até que se chegue ao que melhor atenda ao propósito da aplicação.

Em contrapartida, Sadalage e Fowler acrescentam que não é recomendável apoiar-se no paradigma orientado a documentos em casos que sejam necessárias transações atômicas em múltiplos documentos ou em cenários em que ocorram buscas em estruturas agregadas variáveis.

V. METODOLOGIA

Para as análises comparativas, foram utilizados o banco de dados relacional MySQL e o banco de dados orientado a documentos MongoDB.

Os testes foram executados em um servidor que estava disponível com as seguintes configurações de hardware e sistema operacional:

- Memória RAM: 512MB;
- Processador: Intel Hex-Core 2.5 GHz;
- Disco rígido: 20GB SSD;
- Sistema operacional: Ubuntu Server 14.04 LTS x64.

O servidor contava com as seguintes instalações de banco de dados:

- MongoDB 2.6.4;
- MySQL Community Server 5.5.37.

A base de dados utilizada para a implementação dos testes foi a base de CEP's do Brasil[1], que é disponibilizada por meio do site oficial dos Correios. Esta base é constituída de elementos de endereçamento até nível de seção de logradouros e Códigos de Endereçamento Postal (CEP).

A escolha da base de CEP's dos Correios para o desenvolvimento dos testes propostos neste artigo deve-se pelo seu grande volume de dados, alto relacionamento entre tabelas e diversidade de tipos de dados, o que permite uma grande flexibilidade na elaboração dos gráficos e, consequentemente, a capacidade de analisar cenários dos mais diferentes níveis de complexidade.

Os dados, inicialmente em SQL, precisaram passar por um processo de conversão para que pudessem ser armazenados também na estrutura do MongoDB. Este processo, além de transformar as tabelas SQL em documentos JSON (estrutura em que documentos são armazenados no MongoDB), levou em consideração as boas práticas ditadas pelos bancos de dados orientados a documentos, transformando relações em documentos embutidos onde isso se fez necessário.

As análises comparativas de performance entre os bancos de dados foram feitas através da realização de testes de consultas, inserções, atualizações e exclusões, tanto com o MySQL quanto com o MongoDB. Em cada um dos casos, três variáveis foram levadas em conta:

1. Quantidade de operações realizadas: valores de diferentes ordens de grandeza que determinam quantas vezes a operação (inserção, consulta, atualização e exclusão) foi realizada, viabilizando a visualização crescente de operações.
2. Tamanho do banco de dados: para que o comportamento do banco seja analisado, a quantidade de informações já existentes é fundamental. Assim, é possível observar a variação de performance de acordo com o crescimento dos dados. O tamanho do banco de dados pode dizer respeito tanto à quantidade de registros existentes numa única tabela ou documento, denominados aqui como "cenários simples", quanto ao número de tabelas ou níveis de documentos embutidos, o que chamaremos de "cenários complexos";
3. Tempo de execução: o tempo necessário para a execução de um determinado número de operações em um banco de dados com uma determinada quantidade de informações pré-existentes.

Com base nessas variáveis, os testes práticos visaram observar o comportamento do banco de dados, durante as operações, de acordo com o crescimento do banco de dados.

A necessidade de análise dos já citados cenários simples e complexos tem como objetivo verificar o crescimento vertical (número de registros) e horizontal (número de relacionamentos/embutimentos) nos bancos de dados. Cada um destes cenários teve como resultante quatro gráficos tridimensionais (um para cada operação), totalizando oito gráficos, através dos quais pode-se analisar as três variáveis citadas anteriormente. Assim, os gráficos resultantes foram:

- Gráfico A1: Tempo X tamanho do banco de dados (número de registros) X quantidade de consultas;
- Gráfico A2: Tempo X tamanho do banco de dados (número de relacionamentos/embutimentos) X quantidade de consultas;
- Gráfico B1: Tempo X tamanho do banco de dados (número de registros) X quantidade de inserções;
- Gráfico B2: Tempo X tamanho do banco de dados (número de relacionamentos/embutimentos) X quantidade de inserções;

[1] Base de CEP's dos correios. Acessado em 2 de dezembro de 2014. http://www.correios.com.br/para-voce/precisa-de-ajuda/o-que-e-cep-e-por-que-usa-lo/bases-de-cep

- Gráfico C1: Tempo X tamanho do banco de dados (número de registros) X quantidade de atualizações;
- Gráfico C2: Tempo X tamanho do banco de dados (número de relacionamentos/embutimentos) X quantidade de atualizações;
- Gráfico D1: Tempo X tamanho do banco de dados (número de registros) X quantidade de exclusões;
- Gráfico D2: Tempo X tamanho do banco de dados (número de relacionamentos/embutimentos) X quantidade de exclusões;

Cada ponto do gráfico foi preenchido após a realização do mesmo teste dez vezes, de forma a se obter a média de tempo de execução de cada operação. Nenhuma linguagem de programação ou ferramenta auxiliar foi utilizada para a execução dos testes, uma vez que tais medidas poderiam trazer alterações de desempenho que acabariam por deixar a análise menos realista. As operações foram, portanto, executadas diretamente em seus respectivos bancos de dados.

O *software* multiplataforma Gnuplot foi utilizado para plotar os pontos gerados em gráficos tridimensionais.

VI. Resultados e Análises

A execução dos testes, para ambos MySQL e MongoDB, resultou nos seguintes gráficos:

A1. Consultas simples

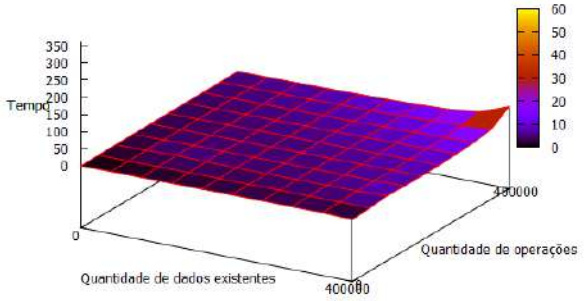

Fig. 01: Gráfico A1, MySQL

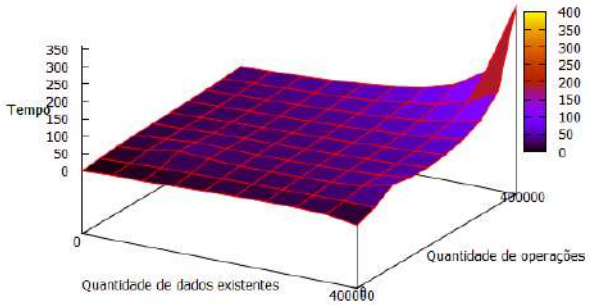

Fig. 02: Gráfico A1, MongoDB

B1. Inserções simples

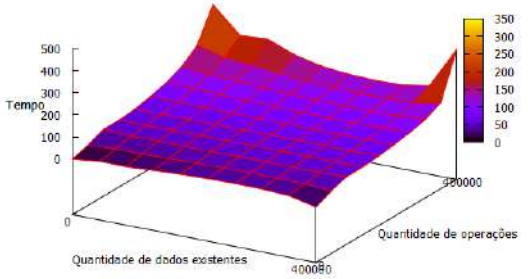

Fig. 03: Gráfico B1, MySQL

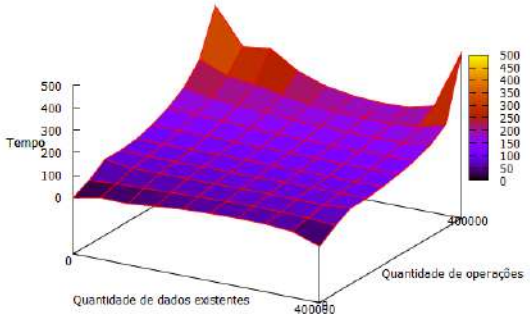

Fig. 04: Gráfico B1, MongoDB

C1. Atualizações simples

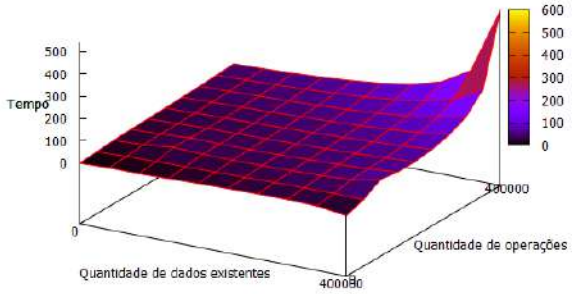

Fig. 05: Gráfico C1, MySQL

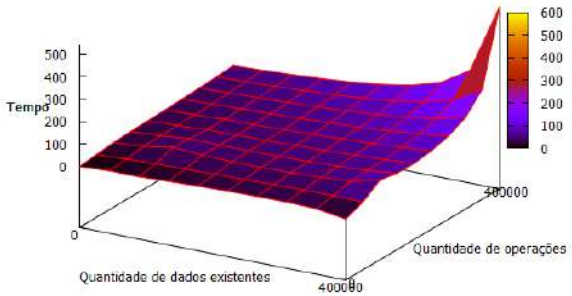

Fig. 06: Gráfico C1, MongoDB

D1. Exclusões simples

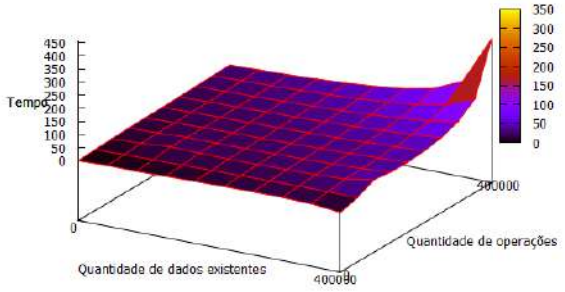

Fig. 07: Gráfico D1, MySQL

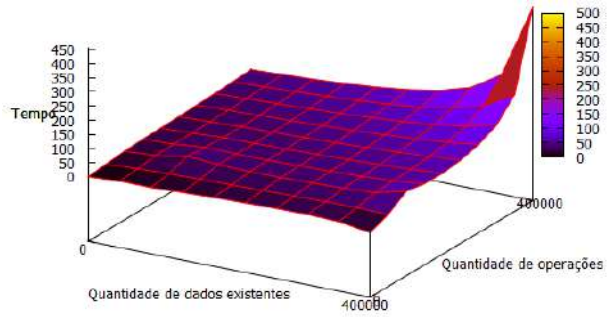

Fig. 08: Gráfico D1, MongoDB

A2. Consultas complexas

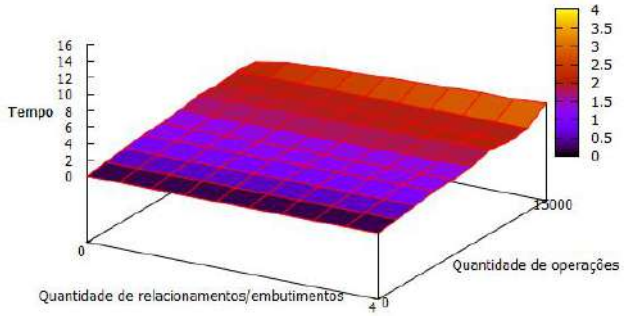

Fig. 09: Gráfico A2, MySQL

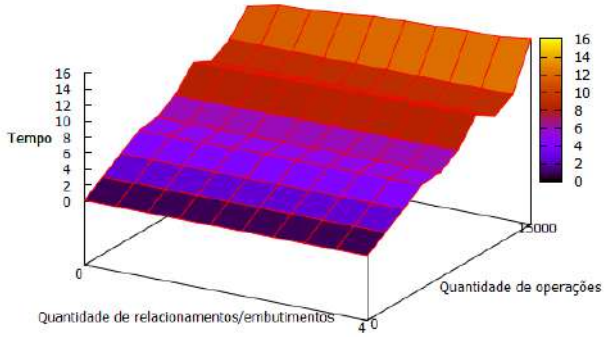

Fig. 10: Gráfico A2, MongoDB

B2. Inserções complexas

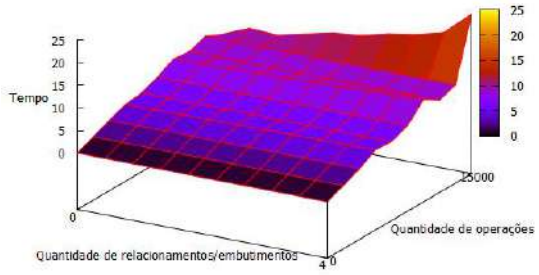

Fig. 11: Gráfico B1, MySQL

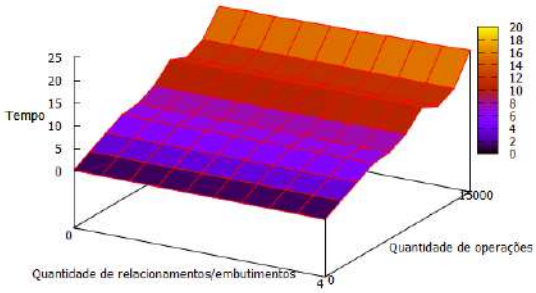

Fig. 12: Gráfico B2, MongoDB

C2. Atualizações complexas

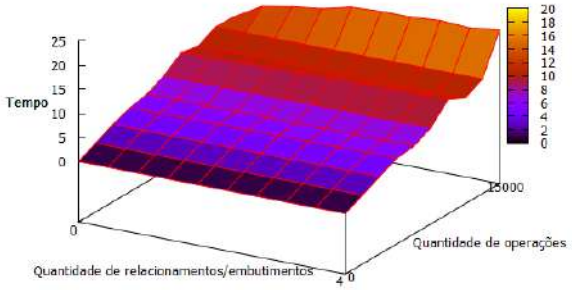

Fig. 13: Gráfico C2, MySQL

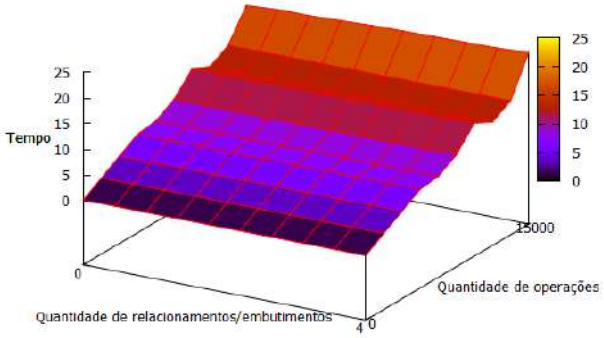

Fig. 14: Gráfico C2, MongoDB

D2. Exclusões complexas

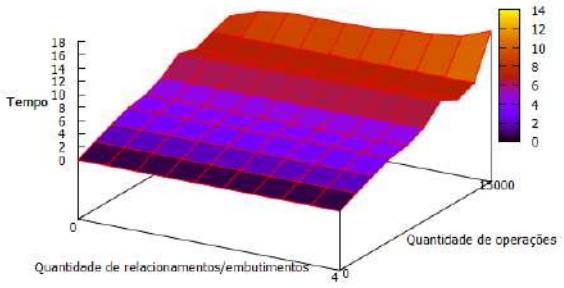

Fig. 15: Gráfico D2, MySQL

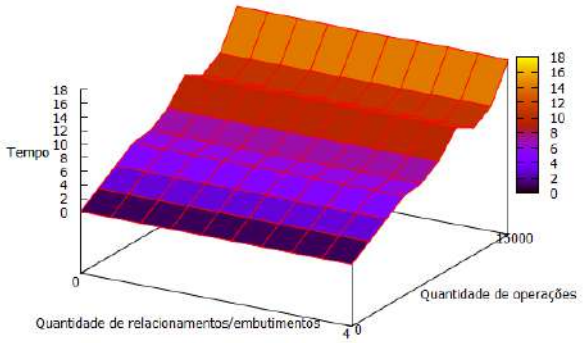

Fig. 16: Gráfico D2, MongoDB

Os resultados obtidos através dos gráficos apresentados são analisados em detalhes na Tabela 1.

Tabela 1: Análise

Gráfico	Resultado
A1. Consultas simples	As Fig. 01 e Fig. 02 mostram que MySQL e MongoDB, respectivamente, reagem de forma semelhante a diferentes quantidades de dados. O MySQL, no entanto, realiza as operações em menor tempo.
B1. Inserções simples	As Fig. 03 e Fig. 04 demonstram um comportamento semelhante de ambos, com vantagem em performance para o MySQL.
C1. Atualizações simples	As Fig. 05 e Fig. 06 novamente apontam o MySQL como vencedor, ainda que por pequena diferença.
D1. Exclusões simples	As Fig. 07 e Fig. 08 concretizam a vitória do MySQL em operações simples, apresentando melhor performance em cenários onde não existam relacionamentos.
A2. Consultas complexas	As Fig. 09 e Fig. 10 mostram que tanto o MySQL quanto o MongoDB reagem de forma similar ao aumento de relacionamentos/embutimentos; apesar disso, o MySQL tem vantagem no tempo de execução das operações.
B2. Inserções complexas	As Fig. 11 e Fig. 12 demonstram que, enquanto o MySQL tem uma notável perda de performance conforme o número de relacionamentos aumenta, o MongoDB permanece estável.
C2. Atualizações complexas	As Fig. 13 e Fig. 14 apresentam uma maior estabilidade do MongoDB quando o número de relacionamentos aumenta, enquanto o MySQL demonstra uma significante perda de performance.
D2. Exclusões complexas	As Fig. 15 e Fig. 16 também ressaltam a vantagem do MongoDB no que diz respeito ao aumento de relacionamentos, pela sua capacidade de se manter estável frente a grandes números de cruzamento de informações.

VII. Conclusão e Perspectivas

Conforme a análise apresentada, é possível notar que, em operações sem relacionamentos de dados, o MySQL apresentou uma vantagem considerável sobre o MongoDB, contando com menor tempo de execução em todas as operações realizadas. Aplicações menos complexas e com poucos relacionamentos entre entidades podem se beneficiar mais do MySQL em termos de performance.

Contudo, a vantagem do MongoDB sobre o MySQL se torna notável durante a execução de operações complexas, ou seja, as quais analisam a performance de acordo com o crescimento da quantidade de relacionamentos ou documentos embutidos existentes no banco. Nesse cenário, torna-se visível, observando o eixo "Quantidade de relacionamentos/embutimentos", que a quantidade de relacionamentos afeta a velocidade de execução do MySQL, enquanto o MongoDB, por sua vez, mantém seu tempo de execução estável, sem ser prejudicado pelo aumento do número de documentos embutidos. Isso pode significar que, em casos onde é necessária a escrita constante de dados com estruturas complexas, o MongoDB pode vir a se tornar a melhor opção, já que possui capacidade de se manter estável conforme o número de entidades associadas aumenta.

Assim, conclui-se que o MongoDB pode ser extremamente vantajoso em casos com maior número de operações de escrita do que leitura, especialmente quando os dados persistidos apresentam estruturas dinâmicas e complexas, como por exemplo:

- Histórico e versionamento de dados;
- *Logs* de aplicações;
- Armazenamento de dados temporários.

Nos casos mencionados acima, inserções, atualizações e

deleções são realizadas constantemente, e nas mais variadas estruturas de dados, enquanto consultas são realizadas eventualmente.

Já o MySQL se torna mais vantajoso em casos onde ocorre a persistência de dados com estrutura previamente definida, a necessidade de consistência entre os dados e as quantidades de inserções e consultas sejam balanceadas ou haja um maior número de consultas.

Mesmo com as evidências apresentadas acima, é importante evidenciar que a escolha por qual banco de dados optar não deve ser fundamentada exclusivamente pela performance atingida por ambos os bancos de dados neste trabalho. Este dado serve como um complemento a análises mais profundas que devem ser realizadas antes que a escolha seja feita.

É importante ressaltar que uma série de variáveis devem ser analisadas, tais como as necessidades de constante consistência entre os dados armazenados, escalabilidade horizontal da aplicação ou estruturas de dados flexíveis e dinâmicas. Assim, uma análise mais aprofundada, somada aos resultados aqui apresentados, permitem uma escolha mais clara em relação a necessidade da utilização de um banco de dados relacional, orientado a documentos ou, inclusive, ambos.

Há diversos pontos relacionados a testes de performance que podem ser abordados em futuros trabalhos relacionados. É interessante observar como ambos tipos de bancos de dados reagem a diferentes níveis de concorrência de usuários, uma vez que este é um cenário bastante comum, por exemplo, em aplicações web. Aplicar os testes demonstrados neste artigo utilizando uma arquitetura distribuída pode vir a demonstrar com clareza qual banco de dados se mostra mais escalável horizontalmente. A forma com que os bancos de dados trabalham com seus respectivos mecanismos de cache também é um caso a se estudar, já que isso pode ter um grande impacto de performance. Além disso, outros tipos de operações podem ser abordados de forma a estender os cenários aqui demonstrados. São casos de operações com IN, *subqueries*, agregação, entre outros.

VIII. Referências

[1] CATTELL, Rick. *Scalable SQL and NoSQL Data Stores*. ACM SIGMOD Record, v. 39, n. 4, p. 12-27, dez. 2010.

[2] CHODOROW, Kristina. *MongoDB: The Definitive Guide, Second Edition.* Sebastopol: O'Reilly Media, Inc., 2013. 392 p.

[3] KRIEGEL, Alex. *Discovering SQL: A Hands-on Guide For Beginners*. Indiana: Wiley Publishing Inc., 2011. 400 p.

[4] RAMAKRISHNAN, Raghu; GEHRKE, Johannes. *Sistemas de Gerenciamento de Banco de Dados*. McGraw Hill Brasil, 2008. 905 p.

[5] SADALAGE, Pramod J.; FOWLER, Martin. *NoSQL Essencial: Um Guia Conciso para o Mundo Emergente de Persistência Poliglota*. Trad. sob a direção de Rubens Prates. São Paulo: Novatec, 2013. 216 p.

[6] VARLEY, Ian Thomas. *No Relational: The Mixed Blessings of Non-Relational Databases.* Austin: The University of Texas at Austin, 2009.

[7] WEBER, Silvan. *NoSQL Databases*. Switzerland: University of Applied Sciences HTW Chur, 2010.

[8] BHAT, Uma; JADHAV, Shraddha. *Moving Towards Non-Relational Databases*. International Journal of Computer Applications (0975 - 8887), v. 1, n. 13, 2010.

[9] NAYAK, Ameya; PORIYA, Anil; POOJARY, Dikshay. *Type of NoSQL Databases and its Comparison with Relational Databases*. International Journal of Information Systems, New York, v. 5, n. 4, mar. 2013.

[10] BREWER, Eric. Towards robust distributed systems. *Principles of Distributed Computing*, Portland, Oregon, jul. 2000.