Analisando o desempenho de bancos de dados orientados a grafos e relacionais sobre discos mecânicos e de estado sólido: uma abordagem comparativa

Priscila Oliveira, Kleber R Stamboni, Jose F Rodrigues-Jr

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP) – São Carlos – SP

[priscila2.oliveira,kleber.stamboni,junio]@usp.br

Abstract. *Relational databases define the dominant model of the market in data processing; however, recently, other models started to gain attention. In the context of today's world, where more attention is given to relationships between people or things, the graph-oriented model can be very useful. In this work, the performances of the relational database PostgreSQL and of the graph-oriented Neo4j are compared based on their execution time. We also evaluate both the databases with respect to solid-state disk performance. The goal is to elucidate the situations in which each model is more appropriate, providing guidance to users and developers.*

Resumo. *Banco de dados relacionais definem o modelo dominante do mercado em processamento de dados; no entanto, outros modelos começaram a ganhar destaque recentemente. No contexto do mundo atual em que se dá mais atenção às relações entre pessoas ou coisas, o modelo orientado a grafos pode ser de grande utilidade. Neste trabalho, os desempenhos do banco de dados relacional PostgreSQL e do orientado a grafos Neo4j são comparados com base no tempo de execução. Também é verificado o comportamento dos SGBDs tanto para discos mecânicos quanto para os de estado sólido. O objetivo é elucidar a respeito das situações nas quais cada modelo é mais apropriado, provendo orientação a usuários e desenvolvedores.*

1. Introdução

Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) já existem comercialmente desde a década de 1970, no entanto, ainda são a tecnologia predominante se comparados a outros modelos de bancos de dados [DB-Engines 2017]. Na segunda década dos anos 2.000, no entanto, os modelos concorrentes, muitas vezes denominados NoSQL (*Not Only SQL*), têm se tornado cada vez mais populares. Dado este cenário, uma comparação entre os modelos torna-se desejável, a fim de se entender as limitações e vantagens de cada um de maneira mais esclarecedora.

Assim, neste artigo, é feita uma comparação entre um banco de dados relacional e um banco de dados orientado a grafos; o objetivo é identificar em quais situações cada abordagem é mais vantajosa, e também avaliar se os SGBDRs são mais adequados do que os bancos orientados a grafos. Para isso, PostgreSQL e Neo4j foram as ferramentas selecionadas como representantes da versão relacional e da orientada a grafos para tal comparação. O PostgreSQL foi escolhido por ser o banco de dados *open-source* mais usado no mercado, e o Neo4j, por ser o banco orientado a grafos também mais usado no

mercado, assim como reportado no sítio Web DB-Engines (https://db-engines.com/en/ranking).

Para a comparação, foram elaborados testes de desempenho. Os testes são divididos em operações de filtragem de dados, junção, agrupamento e ordenação; e são aplicados sobre diferentes quantidades de dados. Além disso, também foi analisada a influência do armazenamento em disco mecânico (HDD) e do armazenamento em unidade de estado sólido (SSD); os testes foram realizados em ambas as tecnologias para se averiguar o impacto do meio de armazenamento.

Este artigo foi estruturado da seguinte maneira: na Seção 2, são apresentados conceitos relacionados aos modelos estudados; na Seção 3, são mostradas as principais características dos *softwares* utilizados; na Seção 4, citam-se as configurações do conjunto de *software* e de *hardware*, além da contextualização dos dados; na Seção 5, discutem-se as definições de cada teste proposto; por fim, na Seção 6, são apresentadas as conclusões.

2. Paradigmas de bancos de dados

A função de um banco de dados é definir uma camada de *software* que suporta a construção de aplicações oferecendo, de maneira abstrata, acesso rápido, concorrente, e persistente a dados. Os bancos relacionais se diferenciam segundo a maneira como os dados são guardados, modelados e acessados.

2.1. Bancos de dados relacionais

Banco de dados relacional, como define o próprio nome, refere-se à tecnologia que armazena dados segundo o modelo relacional, o qual se baseia em relações, compostas por tuplas de elementos ordenados definidos sob domínios que caracterizam atributos. Essas relações são geralmente representadas em linhas, colunas e tabelas. Cada tabela representa uma entidade: alguma 'coisa' ou relacionamento do mundo real — e cada coluna, por sua vez, significa um atributo: alguma característica da 'coisa' em questão [Silberschatz 2006]. Toda tabela possui um atributo especial, chamado de chave, sua função é identificar cada linha e por isso deve ser um valor único para cada uma. Um exemplo pode ser observado na Figura 1.

Bancos relacionais têm como característica marcante a adequação às características ACID (acrônimo para Atomicidade, Consistência, Isolamento e Durabilidade). A linguagem padrão para se comunicar com bancos de dados relacionais é o SQL (*Structured Query Language*). Ela é dividida em dois conjuntos principais, um para definir os dados (comandos CREATE, ALTER e DROP), e outro para manipulá-los (comandos INSERT, UPDATE, SELECT e DELETE). Como já mencionado, este trabalho usa o banco de dados relacional PostgreSQL, detalhado na Seção 3.1.

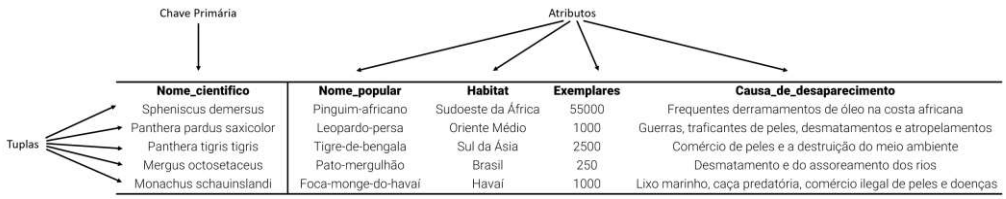

Figura 1. Representação da entidade "Animais_em_extincao" (com informações do sítio Web http://super.abril.com.br/galeria/conheca-20-animais-que-correm-risco-de-extincao).

2.2. Bancos de dados orientados a grafos

Um banco de dados orientado a grafos representa os dados de acordo com a Teoria de Grafos. Um grafo é um modelo matemático capaz de representar relações entre elementos sendo formado por dois conjuntos: um de vértices e um de arestas. Um vértice representa uma entidade — podendo ser do mundo real ou não, já uma aresta representa a conexão entre dois vértices e pode possuir um sentido, uma orientação e, se necessário, dados sobre esse relacionamento [Sato 2014].

A principal motivação em seu uso refere-se a contextos em que a produção de dados define grafos naturalmente, como o exemplo apresentado na Figura 2.

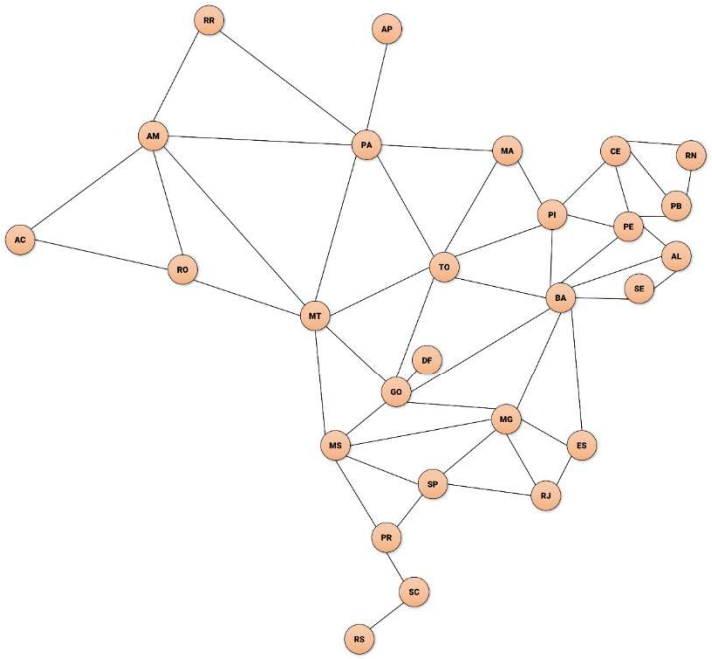

Figura 2. Representação do grafo dos estados brasileiros que fazem fronteira entre si [Feofiloff et al. 2011].

Este trabalho, especificamente, usa o banco de dados orientado a grafos Neo4j — detalhado na Seção 3.2.

3. Bancos de dados considerados

A seguir, as ferramentas usadas nos experimentos são apresentadas.

3.1. PostgreSQL

O PostgreSQL é um Sistema Gerenciador de Banco de Dados Objeto-Relacional (SGBDOR) de código aberto. Ele usa a linguagem de consulta SQL, é compatível com ACID e oferece suporte completo para chaves estrangeiras, junções (JOINs), visões, gatilhos, e procedimentos armazenados, entre outros recursos [Comunidade Brasileira de PostgreSQL 2017].

3.2. Neo4j

O Neo4j é um banco de dados NoSQL orientado a grafos de código aberto. Ele oferece transações totalmente ACID [Neo4j 2017] — algo incomum para NoSQLs, os quais geralmente proveem consistência BASE (*Basically Available, Soft state, Eventual*

consistency), a assim denominada consistência eventual. A linguagem do Neo4j é a Cypher. Ela é uma linguagem declarativa, inspirada em SQL e capaz de definir consultas e dados organizados segundo o modelo de grafos [Sato 2014].

Neste banco de dados, as informações são armazenadas em grafos por meio de nós, e relacionamentos entre eles (ou arestas), ambos podem possuir propriedades que representam suas características.

3.3. Comparação sintática entre SQL e Cypher

Uma comparação sintática das cláusulas mais relevantes da linguagem SQL e da linguagem Cypher pode ser visualizada na Tabela 1. A partir dela, notam-se determinadas semelhanças entre as linguagens, também pode-se notar que em alguns casos Cypher é mais conciso.

Tabela 1. Comparação sintática entre SQL e Cypher.

Cláusula	SQL	Cypher
Seleção	`SELECT * FROM usuario;`	`MATCH (u:usuario) RETURN u;`
Filtragem	`SELECT * FROM usuario WHERE username LIKE 'alvstricklan8196';`	`MATCH (u:usuario) WHERE u.username = 'alvstricklan8196' RETURN u;`
Junção	`SELECT * FROM usuario u JOIN post p ON u.username = p.usuario;`	`MATCH c=(u:usuario)-[:PUBLICA]->(p:post) RETURN c;`
Agrupamento	`SELECT u.username, count(*) FROM usuario u JOIN post p ON u.username = p.usuario GROUP BY u.username;`	`MATCH (u:usuario)-[:PUBLICA]->(p:post) RETURN u.username, count (u);`
Ordenação	`SELECT * FROM usuario u ORDER BY u.nome DESC;`	`MATCH (u:usuario) RETURN u ORDER BY u.nome DESC;`

4. Ambiente de testes

A seguir, são mostrados alguns detalhes sobre as características da base de dados usada nos experimentos, além do *hardware* e *software*.

4.1. O hardware

Os testes foram executados em um computador portátil HP Pavillion 14 V066BR com processador Intel Core i7-4510U de 2.0 GHz, memória RAM de 8 GB modelo DDR3L SDRAM a 1600 MHz, unidade de disco rígido de 1 TB a 5400 RPM e unidade de estado sólido Kingston SSDNow V Series com capacidade de 128 GB.

4.2. O software

O sistema operacional usado foi o Windows 10 Home Single Language. As versões dos gerenciadores foram PostgreSQL 9.6 e Neo4j Community Edition 3.0.6. Todas as bases de dados foram instaladas em um único computador e as configurações dos *softwares* foram mantidas como padrão.

4.3. Os dados

O conjunto de dados empregado nos experimentos descreve uma rede social; seu esquema está ilustrado na Figura 3. Ele foi parcialmente obtido usando-se um gerador de dados *online* [Free Data Generator 2014] e a outra parte foi gerada usando-se um *script* implementado em Python para criar relacionamentos e completar a base.

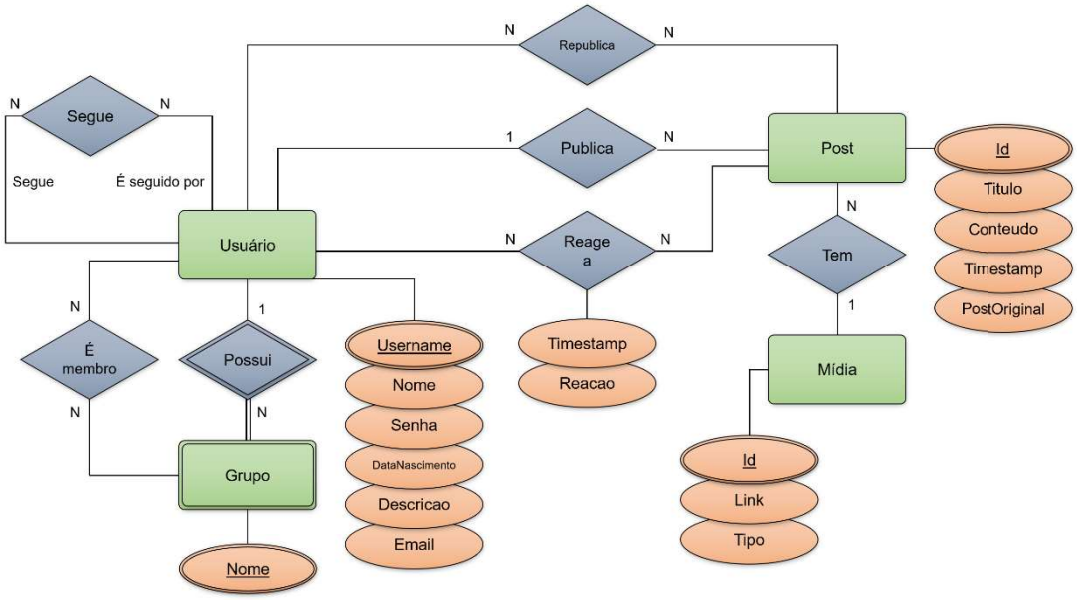

Figura 3. Esquema da base de dados usada nos experimentos.

Os experimentos visam comparar os desempenhos de PostgreSQL e Neo4j usando uma base de dados equivalente em cada uma das ferramentas; ou seja, os dados foram gerados sistematicamente e possuem os mesmos valores em cada banco de dados.

5. Experimentos

Os experimentos foram projetados para comparar os dois tipos de gerenciadores de bases de dados considerando a abrangência de operações esperada por *softwares* desta natureza. Foram consideradas: operações de entrada e saída, filtragem, junção, agrupamento e ordenação. Nesta seção são apresentados detalhes sobre os comandos usados na experimentação; na seção seguinte são exibidos os resultados em termos de espaço de armazenamento e tempo de execução. Foram atribuídos códigos para cada operação a fim de facilitar a identificação de cada uma. A Tabela 2 sumariza os testes realizados e seus respectivos códigos.

Como será detalhado na próxima seção, os experimentos consideraram conjuntos de dados com tamanhos crescentes e dispositivos de armazenamento mecânicos e de estado sólido.

5.1. Inserção, remoção e tamanho

Denominou-se a primeira operação como IRT1 — veja a Tabela 2; nesta operação foram obtidos os tempos de inserção de registros na entidade Usuário, veja o esquema na Figura 3. Todas as inclusões necessárias foram feitas sequencialmente por meio de um *script*.

Do mesmo modo, na operação IRT2, foram obtidos os tempos de remoção dos registros de usuários usando um procedimento equivalente. Por fim, na operação IRT3, analisou-se o espaço ocupado em *megabytes* pelos registros de usuários.

5.2. Filtragem

Quando é necessário encontrar um subconjunto de tuplas que satisfaçam determinados critérios, conectamos cláusulas com operadores conjuntivos (AND) e disjuntivos (OR). O primeiro deles seleciona tuplas que atendam estritamente a todos os critérios de busca. Já o segundo seleciona tuplas que atendam apenas a um (ou porventura mais de um) desses critérios. Os casos analisados para a base estudada são descritos posteriormente.

Na primeira operação, denominada CD1, é realizada uma consulta conjuntiva, na qual buscam-se usuários que tenham determinado nome, descrição com uma palavra especificada, e que estejam em um intervalo de idade.

Na segunda operação, CD2, uma consulta disjuntiva é definida na qual obtém-se usuários com determinado nome ou domínio de e-mail ou tenham mais do que uma idade especificada.

Na terceira, CD3, é realizada uma consulta mais complexa, combinando operadores conjuntivos e disjuntivos, na qual obtém-se usuários com determinado nome ou que possuam descrição com uma palavra específica ou um domínio de e-mail e tenham mais do que uma idade especificada.

Na quarta, CD4, avalia-se o uso de índices. Primeiro obtém-se usuários com uma idade estabelecida; em seguida, realiza-se o teste novamente após a geração dos índices. E por fim, em CD5, é verificado o desempenho das ferramentas na criação e exclusão dos índices.

5.3. Junção

Nos bancos de dados relacionais, chaves estrangeiras referenciam chaves primárias de outras tabelas. Assim, junções são realizadas com base na chave primária e na chave estrangeira de instâncias de tabelas. Para a computação de junções, é necessária a operação de produto cartesiano entre conjuntos, o que faz com que esta operação seja das mais custosas.

Banco de dados orientados a grafos não realizam a operação de junção porque são usados os relacionamentos entre os nós para percorrer o grafo. E, portanto, esse é um dos pontos que pode gerar maiores diferenças de desempenho.

Para a operação J1, é analisada uma consulta complexa em SQL para se obter os relacionamentos de terceiro grau entre usuários. Por ser uma informação obtida naturalmente com as arestas dos grafos, a sintaxe em Cypher é mais simplificada. Na operação J2, a finalidade é avaliar o impacto de uma junção com várias entidades em uma mesma consulta. Veja na Tabela 2.

5.4. Agrupamento

Para a operação de agrupamento (A), foram calculadas a idade mínima, a máxima, a média, e a soma das idades de todos os usuários registrados na base.

5.5. Ordenação

Já para a operação de ordenação (O), foram selecionadas e organizadas as tuplas com os registros de usuários, de acordo com o nome.

5.6. Sumário das operações usadas nos experimentos

A seguir é apresentada a Tabela 2, a qual sumariza as operações usadas nos experimentos da Seção 6. A tabela tem o propósito de oferecer um panorama dos experimentos e também de servir com referência na interpretação dos resultados.

Tabela 2. Tabela resumo das operações usadas nos experimentos.

Fator avaliado	Código	Comando ilustrativo SQL e Cypher
Inserção de dados	IRT1	`INSERT INTO usuario (username, senha, email, nome, idade, descricao) VALUES ('haydking1322', 'l3Xp48HUA', 'Hayd.KI9373@gmail.com', 'Hayden King', 14, 'Lorem ipsum ultricies parturient ante, dolor ipsum sagittis curae; non mattis nibh.');`
		`CREATE (heatwalter8057: usuario{username: "heatwalter8057", senha: "shQ2H2", email: "Heat.WALT4022@live.com", nome: "Heather Walter", idade: 37, descricao: "Lorem ipsum rutrum habitasse habitant nam nullam felis adipiscing vel quam tempor."});`
Remoção de dados	IRT2	`DELETE FROM usuario;`
		`MATCH (m:usuario) DELETE m;`
Consulta conjuntiva	CD1	`SELECT * FROM usuario u WHERE u.nome LIKE '%Sawyer%' AND u.descricao LIKE '%platea%' AND u.idade BETWEEN 13 AND 21;`
		`MATCH (u: usuario) WHERE u.nome =~ '.*Sawyer.*' AND u.descricao =~ '.*platea.*' AND u.idade >= 13 AND u.idade <= 21 RETURN u;`
Consulta disjuntiva	CD2	`SELECT * FROM usuario u WHERE u.nome LIKE '%Sawyer%' OR u.email LIKE '%yahoo%' OR u.idade > 21;`
		`MATCH (u: usuario) WHERE u.nome =~ '.*Sawyer.*' OR u.email =~ '.*yahoo.*' OR u.idade > 21 RETURN u;`
Consulta conjuntiva e disjuntiva	CD3	`SELECT * FROM usuario u WHERE u.nome LIKE '%Sawyer%' OR u.descricao LIKE '%platea%' OR u.email LIKE '%gmail%' AND u.idade > 21;`
		`MATCH (u: usuario) WHERE u.nome =~ '.*Sawyer.*' OR u.descricao =~ '.*platea.*' OR u.email =~ '.*gmail.*' AND u.idade > 21 RETURN u;`
Consulta indexada	CD4	`SELECT * FROM usuario u WHERE u.idade = 21;`
		`MATCH (u:usuario{idade: 21}) RETURN u;`
Operações com índice	CD5	`CREATE INDEX idade_idx ON usuario(idade); DROP INDEX ON :usuario(idade);`
		`CREATE INDEX ON :usuario(idade); DROP INDEX ON :usuario(idade);`
Junção: Tipo 1	J1	`SELECT b.seguidor, s3.usuarioSeguido as "seguido3" FROM (`

		SELECT a.seguidor, s2.usuarioSeguido as "seguido2" FROM (SELECT u.username as "seguidor", s1.usuarioSeguido as "seguido1" FROM usuario u, segue s1 WHERE u.username = s1.usuarioSeguidor) a, segue s2 WHERE a.seguido1 = s2.usuarioSeguidor) b, segue s3 WHERE b.seguido2 = s3.usuarioSeguidor;
		MATCH (a:usuario)-[:SEGUE*3]->(b:usuario) RETURN a.username, b.username;
Junção: Tipo 2	J2	SELECT u.username, p.id, g.nome FROM usuario u JOIN post p ON u.username = p.usuario JOIN grupo g ON u.username = g.dono;
		MATCH (g:grupo)<-[:POSSUI]-(u:usuario)-[:PUBLICA|:REPUBLICA]->(p:post) RETURN u.username, p.id, g.nome;
Agrupamento de dados	A	SELECT MIN(idade), MAX(idade), AVG(idade), SUM(idade) FROM usuario;
		MATCH (m:usuario) RETURN MIN(m.idade), MAX(m.idade), AVG(m.idade), SUM(m.idade);
Ordenação de dados	O	SELECT * FROM usuario ORDER BY nome;
		MATCH (m:usuario) RETURN m ORDER BY m.nome;

6. Resultados

Cada teste foi realizado três vezes, anotando-se o tempo de execução e então calculando-se a média aritmética. Para cada teste, os SGBDs foram reiniciados (*cold run*) para a obtenção de resultados mais precisos. Além disso, o tempo foi comparado considerando-se o meio de armazenamento, disco mecânico (HDD) ou unidade de estado sólido (SSD), e usando-se diferentes tamanhos de base de dados — 5 mil, 10 mil, 20 mil, 50 mil e 100 mil tuplas — para avaliar escalabilidade. Os resultados estão organizados de acordo com as categorias e os identificadores expostos na Seção 5.

6.1. Inserção, remoção e tamanho

Nas atividades de inserção (IRT1) e remoção (IRT2), e com relação ao tempo e ao espaço ocupado pelos registros (IRT3), o Neo4j mostrou-se com menor eficiência que o PostgreSQL na maioria dos testes realizados.

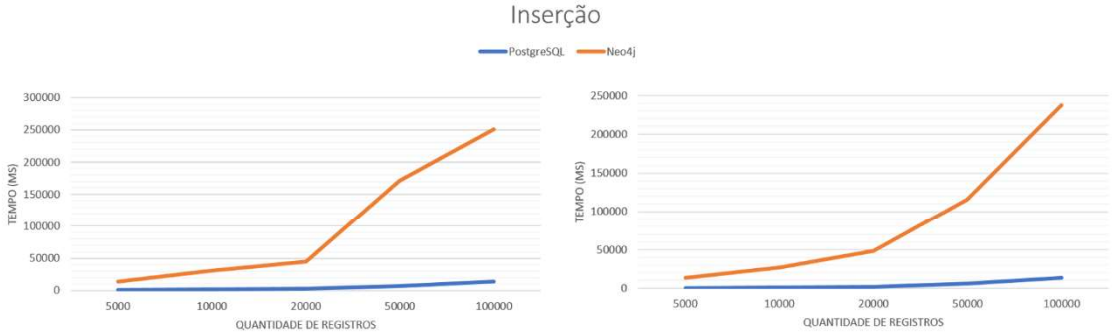

Figura 4. (a) Gráfico comparativo para o tempo da operação IRT1 em HDD; (b) Gráfico comparativo para o tempo da operação IRT1 em SSD.

Na operação IRT1, pode-se notar uma perda expressiva de desempenho do Neo4j conforme o número de registros da base de dados cresce, como mostra a Figura 4. O tempo médio de inserção de cada registro para o banco orientado a grafos foi de 2,8 ms (HDD) e de 2,5 ms (SSD). Para o banco relacional, o tempo médio de inserção foi de 0,2 ms (HDD) e de 0,1 ms (SSD). O tempo de inserção no SSD foi, em média, 23% menor para o caso orientado a grafos, e 9% menor para o relacional.

Na operação IRT2, o Neo4j tem desempenho praticamente equivalente ao PostgreSQL nas bases menores; no entanto, a performance diminui com o crescimento da base. Pode-se ver esse comportamento na Figura 5. O PostgreSQL terminou em vantagem na maior parte dos casos analisados. Ademais, ambos registraram, em média, eficiência 1% para o orientado a grafos e 6% menor em SSD para o caso do relacional.

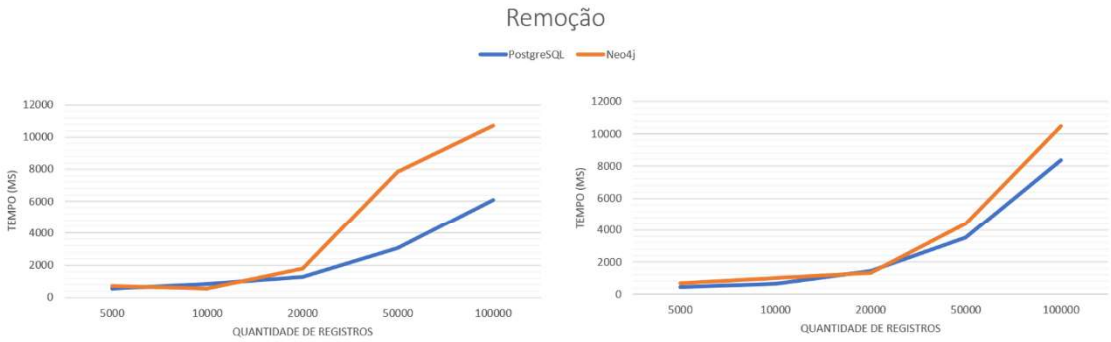

Figura 5. (a) Gráfico comparativo para o tempo da operação IRT2 em HDD (b) Gráfico comparativo para o tempo da operação IRT2 em SSD.

Com relação à análise IRT3, o banco orientado a grafos ocupa menos espaço comparado ao relacional quando são armazenados menos de 20 mil registros; a partir dessa quantidade, o espaço ocupado torna-se maior para o Neo4j. Essa situação é demonstrada na Figura 6. Em média, para todos os casos estudados, o tamanho dos registros em memória secundária do Neo4j é 1,6x maior do que o do PostgreSQL.

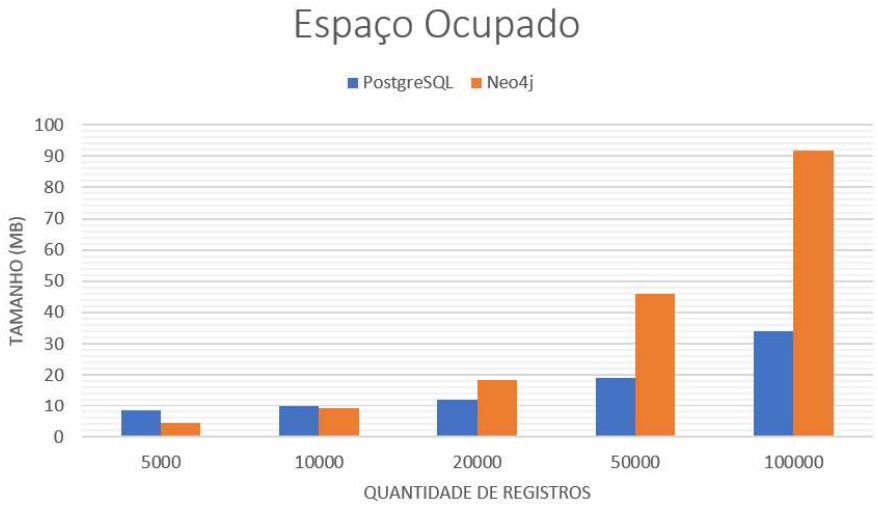

Figura 6. Gráfico comparativo para o espaço ocupado pelos registros armazenados de acordo com o tamanho da base (IRT3).

6.2. Filtragem

Na operação CD1, como mostrado na Figura 7, o Neo4j mostrou-se com o melhor desempenho a partir de bases com 20 mil registros para unidade de disco HDD. No entanto, em SSD, o PostgreSQL apresentou melhor desempenho em todos os casos analisados.

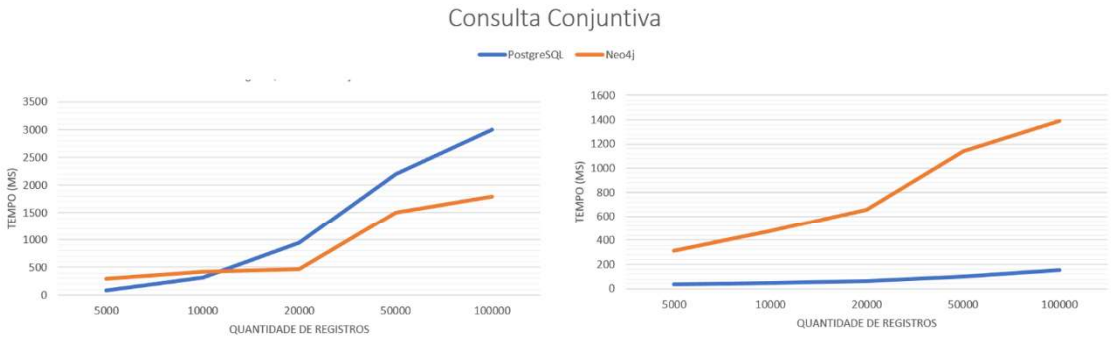

Figura 7. (a) Gráfico comparativo para o tempo da consulta CD1 em HDD; (b) Gráfico comparativo para o tempo da consulta CD1 em SSD.

Para a operação CD2, o banco de dados orientado a grafos obteve o melhor desempenho para todos os casos estudados. No entanto, a eficiência foi 29% menor, quando comparado com o HDD. Do mesmo modo, o PostgreSQL obteve apenas 3% a mais de eficiência no disco sólido. Como pode-se ver na Figura 8.

Figura 8. (a) Gráfico comparativo para o tempo da consulta CD2 em HDD; (b) Gráfico comparativo para o tempo da consulta CD2 em SSD.

Em CD3, o Neo4j teve melhor desempenho a partir de 10 mil registros em HDD; em SSD, apenas a partir de 50 mil. Pode-se ver esse comportamento na Figura 9. Além disso, o PostgreSQL executou os testes em um tempo 10% menor, em média, no SSD; no Neo4j, o desempenho foi oposto, sendo 4% maior em tempo.

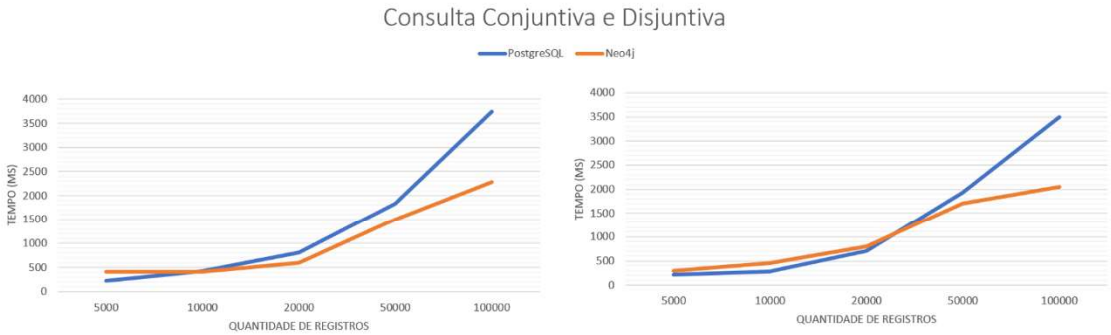

Figura 9. (a) Gráfico comparativo para o tempo da consulta CD3 em HDD;

(b) Gráfico comparativo para o tempo da consulta CD3 em SSD.

Em CD4, o Neo4j teve melhores resultados para HDD na maioria dos casos; no entanto, em SSD e com índice, o PostgreSQL executou a operação em menos tempo, conforme mostrado na Figura 10. É importante notar que o Neo4j não cria seus índices automaticamente.

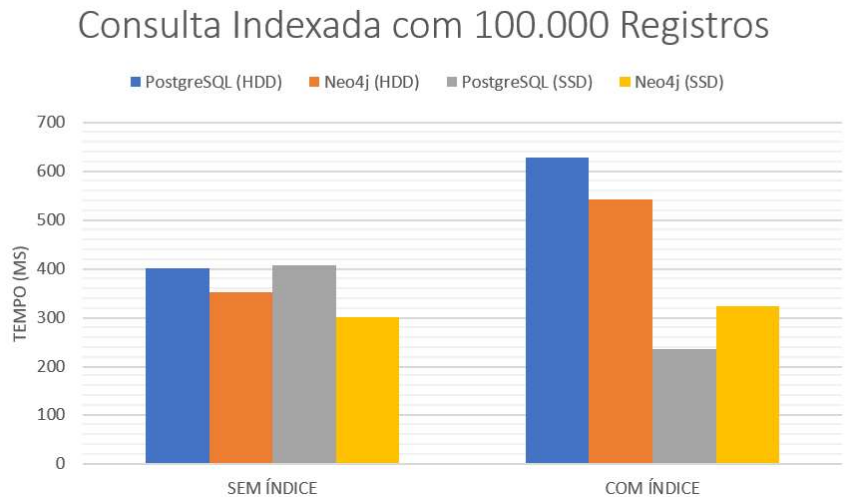

Figura 10. Gráfico comparativo para o tempo da consulta CD4, com e sem índice.

Na Figura 11, são apresentados os tempos de criação e remoção de índices, e como pode-se notar, o banco de dados relacional demora um período bem maior para criá-los, quando comparado com o banco orientado a grafos, tanto em disco sólido como rígido, assim como um período bem menor para excluí-los.

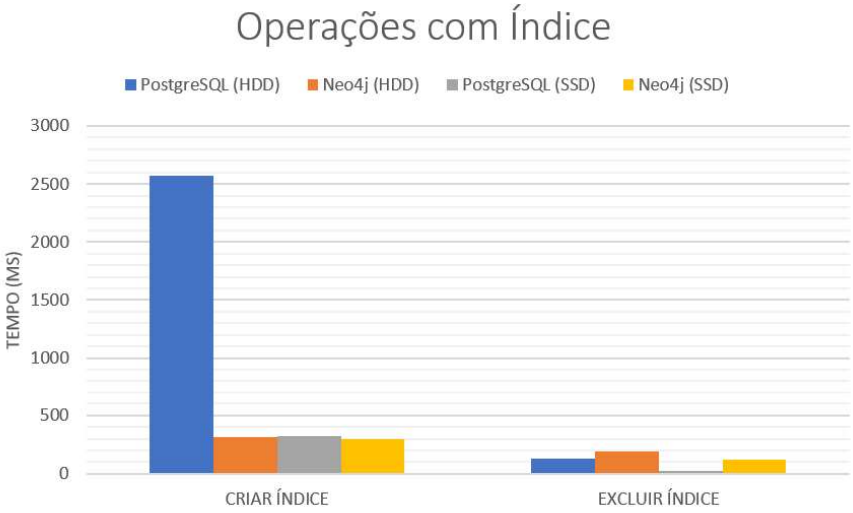

Figura 11. Gráfico comparativo das operações em CD5.

6.3. Junção

Nas operações de junção, o Neo4j obteve melhores resultados em SSD, com tempos menores de execução para todos os tamanhos de base estudados em ambos os casos — J1 e J2.

Em J1, o banco de dados orientado a grafos, em HDD, obteve melhor tempo para bases até 50 mil registros, porém, para 100 mil, o relacional teve um desempenho levemente superior, como mostra a Figura 12.

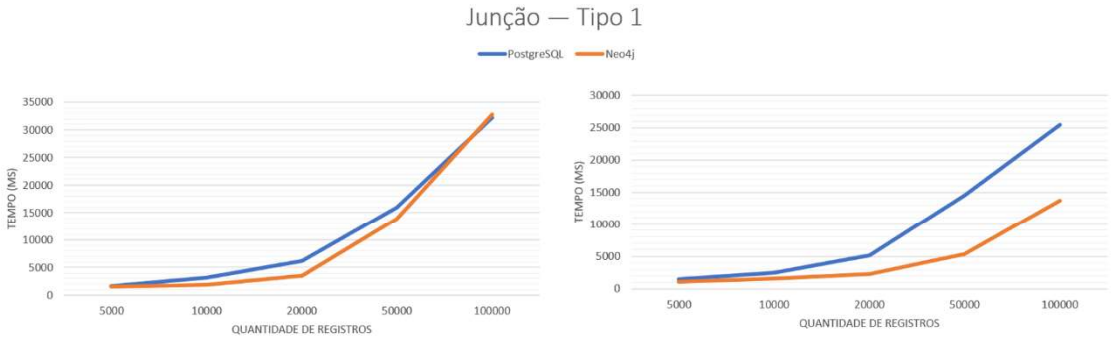

Figura 12. (a) Gráfico comparativo para o tempo da consulta J1 em HDD; (b) Gráfico comparativo para o tempo da consulta J1 em SSD.

Em J2, o Neo4j manteve o desempenho superior para bases até 20 mil registros, todavia, para 50 mil e 100 mil, os menores tempos foram apresentados pelo PostgreSQL (quando em HDD), de acordo com a Figura 13.

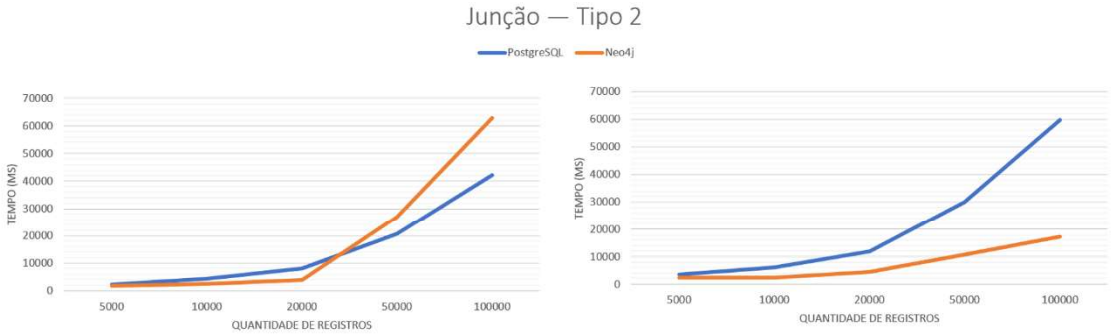

Figura 13. (a) Gráfico comparativo para o tempo da consulta J2 em HDD; (b) Gráfico comparativo para o tempo da consulta J2 em SSD.

6.4. Agrupamento

Nesta operação, o Neo4j obteve resultados melhores que o PostgreSQL apenas para bases de 10 mil e 20 mil registros, em disco rígido, como mostra a Figura 14. Além disso, seu desempenho foi inferior no SSD em comparação ao HDD para todos os casos analisados.

Em disco sólido, o banco de dados relacional obteve, em média, ganho de 64% em eficiência, ao contrário do banco orientado a grafos que perdeu 53%.

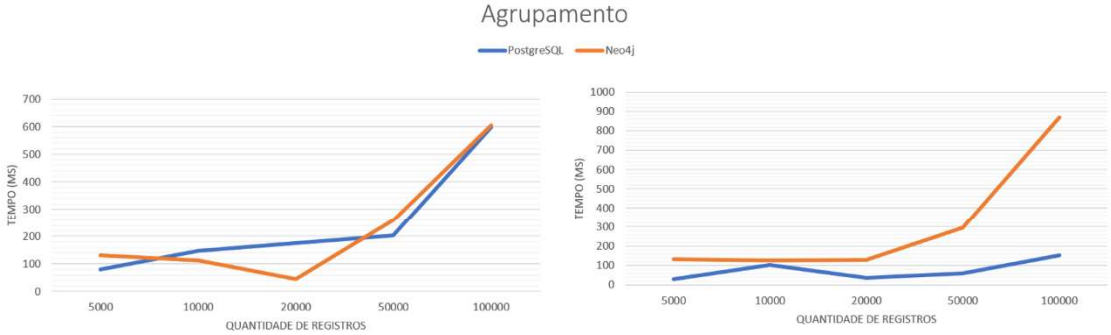

Figura 14. (a) Gráfico comparativo para o tempo da operação A em HDD; (b) Gráfico comparativo para o tempo da operação A em SSD.

6.5. Ordenação

Nesta operação, conforme mostrado na Figura 15, o Neo4j apresentou melhores resultados para todos os casos estudados. O PostgreSQL levou, em média, 3,7x mais tempo para processar a mesma requisição do que o Neo4j em HDD e 2,8x em SSD.

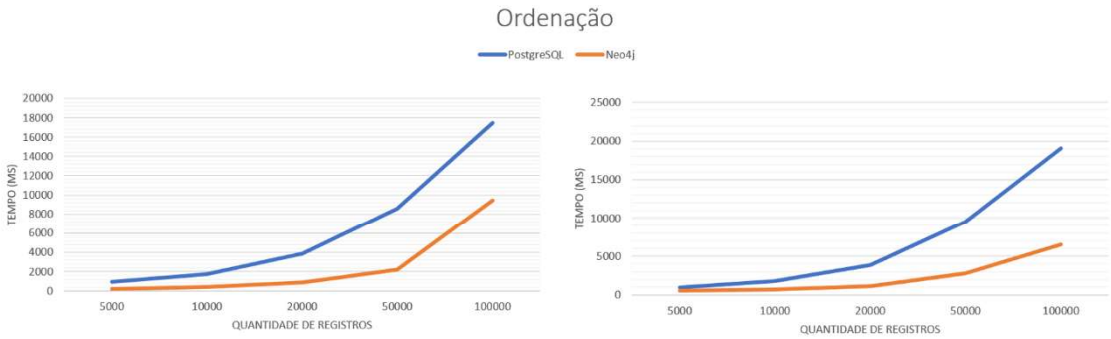

Figura 15. (a) Gráfico comparativo para o tempo da operação O em HDD; (b) Gráfico comparativo para o tempo da operação O em SSD.

6.6. HDD x SSD

Por fim, são feitas comparações entre todas as operações realizadas que não envolvem o uso de índice. A Figura 16 mostra os tempos para o disco rígido e para o disco sólido para o PostgreSQL e, na Figura 17, equivalentemente para o Neo4j. É importante notar que as escalas dos gráficos diferem.

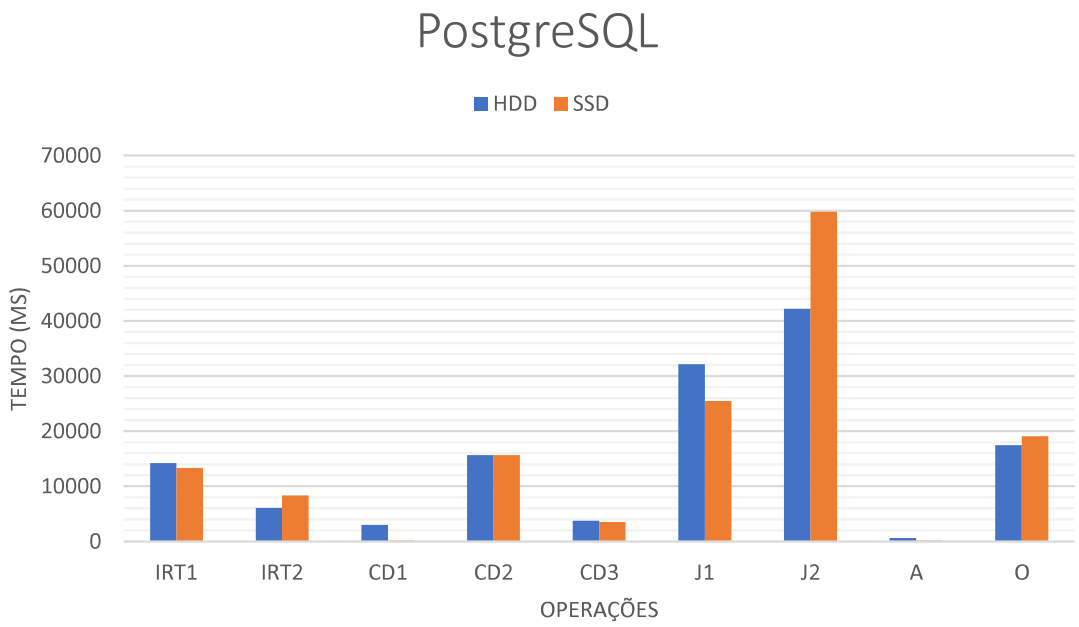

Figura 16. Gráfico comparativo com os tempos das operações em HDD e SSD no PostgreSQL com uma base de 100 mil registros.

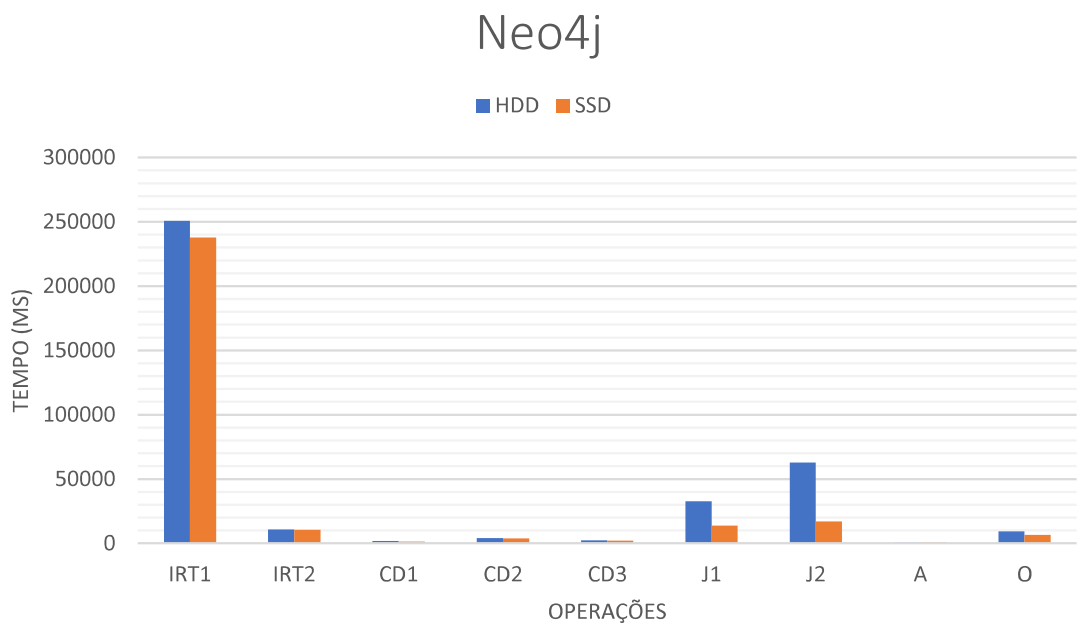

Figura 17. Gráfico comparativo com os tempos das operações em HDD e SSD no Neo4j com uma base de 100 mil registros.

6.7. Discussão dos resultados

Em comparação direta de desempenho, foi possível avaliar diversos aspectos do banco de dados relacional PostgreSQL e do banco de dados orientado a grafos Neo4j. Primeiro, ficou claro que o Neo4j requer mais espaço de armazenamento do que o PostgreSQL. Todavia, espaço de armazenamento não é uma questão mandatória atualmente, dado o baixíssimo custo de armazenamento. No que se refere a operações de inserção e remoção, o PostgreSQL se mostrou mais eficiente, o que o torna mais recomendado para sistemas em produção com alta taxa de entrada, como sistemas OLTP (*Online Transaction Processing*) que suportam operadoras de cartão de crédito, por exemplo. A diferença de desempenho se deve à maneira como o Neo4j organiza os dados em disco, com um arquivo para os nós, um para as propriedades, e um para os relacionamentos (arestas). Cada arquivo requer uma operação adicional em disco durante a escrita. Nota-se que, obviamente, manter arquivos abertos não causa impacto significativo em desempenho; o impacto vem do acesso (*disk seek*) decorrente do fato de se estar usando dados não contíguos.

A comparação com relação às operações de consulta conjuntivas e disjuntivas não mostraram diferenças de desempenho tão pronunciadas. Dado o fato de que os predicados das consultas se baseiam nas propriedades dos dados, tanto o PostgreSQL quanto o Neo4j resolvem o problema sobre um único arquivo, cujo acesso pode ser otimizado de diferentes maneiras. Com relação ao uso de índices, o Neo4j se mostrou significativamente mais eficiente tanto no uso quanto no gerenciamento.

Nas operações de junção, como se pode esperar, o Neo4j mostrou desempenho superior, especialmente quando usado sobre um disco de estado sólido. Este desempenho é esperado pois o banco orientado a grafos possui um arquivo dedicado ao registro dos relacionamentos entre as entidades do banco, não havendo custo de processamento para realizar a junção cartesiana. Assim, durante a junção, o Neo4j precisa apenas recuperar as entidades envolvidas a partir dos arquivos de nós, uma operação que se torna pronunciadamente mais eficiente em um disco de estado sólido.

Já com relação à operação de agrupamento, o PostgreSQL mostrou melhor desempenho. Isto ocorre, possivelmente, pois no Neo4j os dados que definem os grupos do agrupamento estão em um arquivo, e os dados de sumarização estão em outro arquivo; ao passo que o PostgreSQL resolve o problema sobre um único arquivo de tuplas.

Na operação de ordenação, o Neo4j apresentou melhor desempenho, possivelmente porque o arquivo de nós, que é suficiente para a operação de ordenação, é menor do que o arquivo de tuplas do PostgreSQL, o que permite que o algoritmo de ordenação faço melhor uso da memória principal.

Com relação ao uso de SSD e de HDD, ficou claro que o SSD melhora o desempenho dos bancos em praticamente todos os casos. Todavia, o ganho em desempenho não é proporcional ao maior custo dos discos SSD. Com efeito, o GB em SSD ainda é em torno de 700% mais caro que o GB em um HDD, ao passo que o ganho em desempenho não alcança 200%.

7. Conclusão

Os experimentos mostraram que os dois tipos de bancos têm melhor desempenho em diferentes situações. O PostgreSQL se destaca em sistemas de alta taxa de escrita e em operações de agregação para análise de dados. Ao passo que o Neo4j se destaca em operações de consulta e de junção, o que o torna mais adequado a esquemas com muitos relacionamentos e constantes consultas envolvendo junção.

Um dos problemas de se usar o Neo4j vem do fato de que ele não usa SQL; ao passo que a linguagem Cypher não é muito complexa, seu uso por equipes de desenvolvimento pode exigir um tempo de adaptação ou, pior, a reescrita de sistemas já existentes.

Para trabalhos futuros, abrem-se várias possibilidades relacionadas a este assunto, como o estudo com bases de dados ainda maiores para verificar se o desempenho do Neo4j se mantém; a comparação com outros SGBDRs relevantes como Oracle, MySQL e Microsoft SQL Server; e a avaliação do impacto do tamanho da memória principal no tempo das consultas.

8. Referências

[Comunidade Brasileira de PostgreSQL 2017] COMUNIDADE BRASILEIRA DE POSTGRESQL. **Sobre o PostgreSQL | Comunidade Brasileira de PostgreSQL.** Disponível em: <https://www.postgresql.org.br/pages/sobre-o-postgresql.html>. Acesso em: 26 dez. 2016.

[DB-Engines 2017] DB-ENGINES. **DB-Engines Ranking – popularity ranking of database management systems.** Disponível em: <http://db-engines.com/en/ranking>. Acesso em: 26 dez. 2016.

[Feofiloff et al. 2011] FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. **Uma Introdução Sucinta à Teoria dos Grafos.** Disponível em: <https://www.ime.usp.br/~pf/teoriadosgrafos/texto/TeoriaDosGrafos.pdf>. Acesso em: 22 maio 2017.

[Free Data Generator 2014] FREE DATA GENERATOR. **SQL Data Generator.** Disponível em: <http://www.freedatagenerator.com/sql-data-generator>. Acesso em: 28 jan. 2016.

[Neo4j 2017] NEO4J. **Product At-A-Glance.** Disponível em: <https://info.neo4j.com/rs/neotechnology/images/Product%20At-A-Glance.pdf>. Acesso em: 26 dez. 2016.

[Sato 2014] SATO, P. M. **GraphDB Series: o que é um banco de dados de grafos.** Disponível em: <https://imasters.com.br/banco-de-dados/graphdb-series-o-que-e-um-banco-de-dados-de-grafos>. Acesso em: 22 maio 2017.

[Silberschatz 2006] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados.** Tradução de Daniel Vieira. Rio de Janeiro: Elsevier, 2006. p. 11.