Um Estudo sobre Bancos de Dados em Grafos Nativos

Raqueline R. M. Penteado, Rebeca Schroeder, Diego Hoss, Jaqueline Nande, Ricardo M. Maeda, Walmir O. Couto, Carmem S. Hara

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

{rrmpenteado,rebecas,jfbn10,djhoss,rmmaeda,wocouto,carmem}@inf.ufpr.br

Abstract. *Graph databases support applications based on complex data models where the interconnectivity of data is an important aspect. The ever-increasing amount of data made available by these systems has been handled by graph databases in order to scale such systems. In spite of the straightforward way to represent complex data, native graph databases provide efficient query processing by avoiding costly joins. This paper presents a comparative analysis among native graph databases by considering some important features for data management system in this context.*

Resumo. *Sistemas de banco de dados em grafos suportam aplicações baseadas em modelos de dados cuja a interconectividade de dados é um aspecto importante. O volume de dados crescente envolvendo tais aplicações vem sendo tratado por uma série de soluções de gerenciamento de dados baseados em grafos visando a escalabilidade destes sistemas. Além de prover uma forma direta para a representação de dados complexos, bancos de dados em grafos classificados como nativos são capazes de executar consultas de forma eficiente por eliminar operações custosas de junções. Este artigo apresenta uma análise comparativa entre alguns sistemas nativos atuais considerando algumas características importantes para sistemas de gerenciamento neste contexto.*

1. Introdução

Durante décadas os Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) dominaram o meio empresarial e acadêmico por utilizarem um modelo de dados intuitivo, uma linguagem padronizada de consulta e manipulação de dados, e garantirem as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) em aplicações diversas. Apesar de todos os benefícios em se utilizar um SBGDR, aplicações baseadas em modelos de dados complexos podem arcar com problemas decorrentes desta adaptação ao modelo relacional, especialmente para bancos de dados que devem dar suporte a um grande volume de dados, de requisições e de usuários concorrentes.

O banco de dados em grafos surgiu como uma alternativa ao banco de dados relacional para dar suporte a sistemas cuja interconectividade de dados é um aspecto importante. Após seu surgimento nos anos 80, os bancos de dados em grafos rapidamente perderam espaço para os bancos de dados semi-estruturados em virtude da ascensão do modelo XML[Angles and Gutierrez 2008]. No entanto, o interesse pelo modelo de banco de dados em grafos ressurgiu juntamente com a web semântica e a disseminação das redes sociais[Liptchinsky et al. 2013]. A evidência recente deste modelo pode ser atestada pelo volume crescente das diversas bases de dados presentes no projeto *Linked Data*[1], assim

[1]http://linkeddata.org/

como no projeto *DBpedia*[2] e pelo conjunto de repositórios *Large Triple Stores*[3].

Um exemplo clássico do uso de banco de dados em grafos é o *Twitter* [4], uma rede social que oferece um serviço de microblog. Sua base é modelada diretamente como um grafo, onde os usuários são os vértices e os relacionamentos entre eles são as arestas. Outros exemplos de contextos são: sistemas que recomendam compras em lojas virtuais, sistemas que exploram dados químicos e biológicos para detecção de padrões, e sistemas web como o *PageRank* [Brin and Page 1998] da *Google* que analisa a importância dos sites computando o número de arestas incidentes em cada site.

Os sistemas de banco de dados em grafos modelam seus dados por meio de vértices e arestas facilitando a modelagem de contextos complexos e definindo naturalmente relações existentes entre as entidades de uma base. Nesta categoria os sistemas podem ser classificados como nativos ou não-nativos. Os nativos consideram a estrutura de grafo tanto no armazenamento físico dos dados quanto no processamento de consultas. Listas de adjacências `vértice-vértice` são usadas no armazenamento físico possibilitando a exploração do grafo de forma simples, direta e eficiente no processamento de consultas. Em contrapartida, os sistemas não-nativos usam outros modelos para o armazenamento e processamento de consultas. Por exemplo, por meio do modelo relacional, as relações de triplas `vértice-aresta-vértice` em um grafo são armazenadas como tuplas em tabelas. Este tipo de composição é prejudicial para o desempenho de consultas quando diversas junções são necessárias para executar uma consulta complexa envolvendo diversas triplas.

Um projeto que tem recebido destaque no âmbito dos Sistemas Gerenciadores de Bancos de Dados em Grafos (SGBDGs) nativos é o *Tinkerpop* [5]. Este projeto foi criado em 2008 e tem o foco em estruturas de dados, consultas e linguagens de programação para grafos. Nele foi desenvolvida a *Blueprints*, uma interface básica para grafos de propriedades [6], que define uma série de métodos para interação com um grafo. Além disso, o *Tinkerpop* também oferece ferramentas para interagir com os SGBDGs que dão suporte à *Blueprints*, como o servidor de grafos *Rexster* que pode ser usado por um cliente para acessar uma base remotamente e a linguagem de consultas *Gremlin*.

Atualmente, alguns SGBDGs têm recebido destaque entre pesquisadores e empresas. Uma grande porção destes modelam os dados nativamente como grafos, porém, possuem características diversas que vão desde o tipo de grafo usado na modelagem até a forma de armazenamento de dados em disco. O objetivo deste artigo é apresentar e comparar algumas características de alguns sistemas de processamento nativo. As características usadas na comparação oferecem uma visão geral quanto à possibilidade dos usuários alterarem as funcionalidades dos sistemas, à facilidade do uso das funcionalidades dos sistemas e às variáveis envolvidas no desempenho dos sistemas, sendo elas: tipo de licença, suporte ao *Tinkerpop*, modelagem lógica, linguagem de consulta, modelagem física, armazenamento físico, suporte à transação, tipo de arquitetura e suporte à replicação.

---

[2] http://wiki.dbpedia.org/Datasets

[3] http://www.w3.org/wiki/LargeTripleStores

[4] twitter.com

[5] www.tinkerpop.com

[6] *property graph*

As contribuições deste artigo envolvem uma descrição do processo de construção e manipulação de uma base de dados em grafos considerando as características utilizadas na comparação objeto deste artigo e um estudo comparativo entre cinco sistemas de bancos de dados em grafos nativos. Os sistemas analisados foram: InfiniteGraph [7], Neo4j [8], OrientDB [9], Titan [10] e o Trinity [Shao et al. 2013].

O restante deste artigo está organizado da seguinte forma: os principais conceitos de um SGBDG são definidos pela Seção 2; nas seções 3 e 4 é apresentada uma visão geral sobre modelagem em grafos e consultas em grafos; os sistemas selecionados para a análise proposta são introduzidos pela Seção 5 e comparados na Seção 6; a Seção 7 apresenta os trabalhos relacionados deste artigo e; a Seção 8 apresenta as conclusões.

2. Sistema Gerenciador de Banco de Dados em Grafo

Na execução de um projeto de banco de dados, uma base é modelada logicamente e mapeada para uma forma de armazenamento físico para que seus dados possam ser manipulados por um sistema. A modelagem lógica abstrai detalhes do projeto facilitando o seu desenvolvimento. Já o mapeamento e o armazenamento físico envolvem detalhes técnicos e implicam diretamente no desempenho das aplicações.

Naturalmente, a modelagem lógica em um SGBDG é feita por meio do modelo de grafos. Basicamente, a topologia de um grafo G pode ser expressa como $G = (V, E)$, onde V é o conjunto de vértices (nós) e E é o conjunto de arestas. Cada aresta representa uma relação entre dois nós. Um conjunto de vértices conectados por meio de arestas definem um caminho no grafo. Diferentes tipos de grafos podem ser usados na modelagem, como mostra a próxima seção.

O mapeamento lógico-físico classifica os SGBDGs em não-nativos ou nativos. Os não-nativos modelam logicamente seus dados como grafos, porém armazenam-os por meio de outros modelos. Sistemas como RDF-3X [Neumann and Weikum 2010] e SW-Store [Abadi et al. 2009] armazenam suas triplas em tabelas relacionais, enquanto o HyperGraphDB [Iordanov 2010] armazena o grafo fisicamente em uma estrutura chave-valor. Já os nativos, em geral, usam listas de adjacência [Aho and Ullman 1995]. Em uma lista de adjacência, cada vértice mantém referências diretas para seus vértices adjacentes formando uma espécie de micro-índice para os vértices próximos. Esta propriedade é conhecida como adjacência livre de índices [11] [Robinson et al. 2013].

Um modelo deve ser implementado para o armazenamento físico dos dados representados por meio de vértices e arestas no modelo lógico. Por exemplo, as listas de adjacência podem ser armazenadas em um repositório por meio do modelo chave-valor onde a chave identifica um vértice e o valor referencia a lista de adjacência. Outra característica importante no armazenamento físico é a forma de armazenamento, em memória ou em disco. Vários sistemas exploram a primeira opção visando melhorar seu desempenho descartando a necessidade de acessos ao disco. Porém, o armazenamento em memória trabalha com dados voláteis e limita a capacidade de armazenamento quando comparado

[7] www.objectivity.com/infinitegraph
[8] www.neo4j.org
[9] orientdb.org
[10] thinkaurelius.github.io/titan
[11] *index-free adjacency*

ao disco.

Ainda em relação ao armazenamento físico, um SGBD pode ter uma arquitetura centralizada ou distribuída. Na centralizada, os dados ficam em um único servidor responsável por receber/responder todas as requisições dos clientes. Na distribuída, o banco de dados é fragmentado e suas partes componentes são fisicamente armazenadas em diferentes servidores [Date 2004]. Em geral, sistemas que usam a arquitetura distribuída oferecem a funcionalidade de replicação de dados com o objetivo de melhorar a escalabilidade e a disponibilidade de suas aplicações. Os SGBDs distribuídos são a solução ideal para garantir um bom desempenho nas aplicações que demandam um grande volume de requisições e de dados.

3. Modelos de Grafos

Uma base pode ser modelada de diferentes formas dependendo do modelo de grafo escolhido. Um modelo bem simples e limitado é o grafo simples-relacional. Nele todos os vértices denotam o mesmo tipo de objeto e todas as arestas denotam o mesmo tipo de relacionamento. Já o grafo multi-relacional permite um conjunto variado de tipos de objetos e de relacionamentos possibilitando múltiplas relações e um maior poder de modelagem. Ainda há a opção de uma aresta ser direcionada e/ou rotulada e/ou valorada com um peso em um modelo. Adicionalmente, arestas e vértices podem ter propriedades com valores associados [Rodriguez and Shinavier 2008].

Angles[Angles 2012] cita quatro tipos de modelos:

1. Grafos simples: é a definição básica em que um grafo é definido por um conjunto de vértices conectados por arestas par a par;
2. Hipergrafos: uma aresta (hiper-aresta) pode se relacionar com um número arbitrário de vértices;
3. Grafos aninhados: um vértice (hiper-vértice)também pode ser um grafo;
4. Grafos de propriedades: vértices e arestas contêm atributos para descrever suas propriedades.

O modelo mais utilizado entre os SGBDGs atuais é o grafo de propriedades. Rodriguez e Neubauer [Rodriguez and Neubauer 2010] definem este modelo como um grafo multi-relacional com atributos e arestas direcionadas. A Figura 1 exibe um grafo de propriedades no contexto de uma rede social corporativa. Os vértices que representam as pessoas possuem as propriedades “nome” e “idade” e os vértices que representam as empresas possuem as propriedades “nome” e “cidade”. As arestas representam as relações “um empregado trabalha para uma empresa” e “uma pessoa conhece outra pessoa”.

4. Operações sobre Grafos

Quando uma base é armazenada de forma nativa, vários algoritmos podem ser utilizados para consultá-la. Angles[Angles 2012] considera quatro grupos de consultas essenciais em grafos, sendo elas:

- Consultas adjacentes: o princípio básico é a adjacência `vértice-aresta`. Dois vértices são adjacentes quando possuem uma aresta entre eles. Duas arestas são adjacentes quando compartilham um vértice em comum.
- Consultas de acessibilidade: este tipo de consulta considera o caminho ou a travessia em um grafo. Ele testa se dois vértices estão conectados por um caminho no grafo.

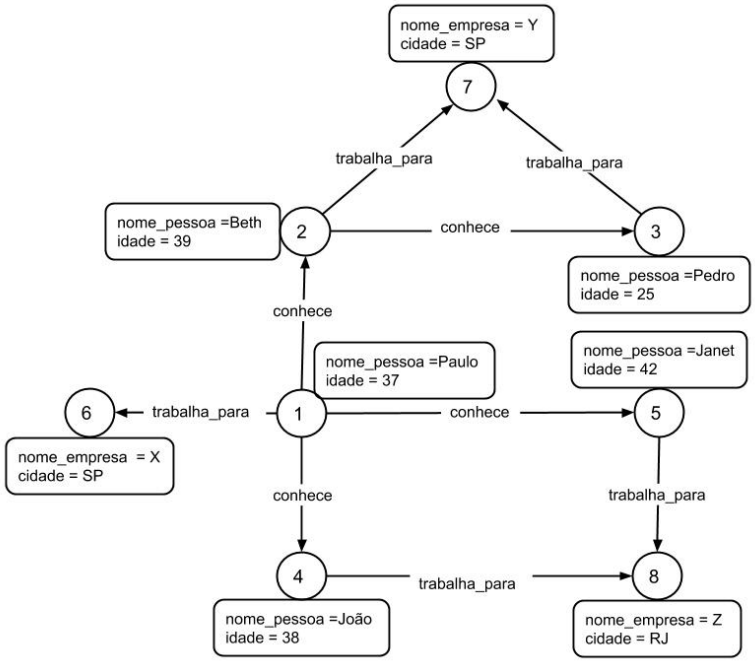

Figura 1. Uma rede social corporativa

- Consultas de combinações de padrões: este tipo de consulta procura todos os subgrafos isomórficos de um grafo padrão de uma determinada consulta.
- Consultas de sumarização: este tipo de consulta usa funções especiais que permitem sumarizar ou operar nos resultados das consultas, normalmente retornando um valor simples.

Em contrapartida, os SGBDGs não-nativos não conseguem explorar os mesmos algoritmos de consulta que os nativos. Logo, o processamento de suas consultas dependem diretamente da forma de armazenamento escolhida.

Os exemplos da Tabela 1 exibem consultas básicas no grafo da Figura 1 com o *Gremlin*[12], ferramenta desenvolvida pelo *Tinkerpop*. No quesito consulta, *Gremlin* é classificada como uma linguagem de travessia [13] que usa algoritmos de busca em profundidade e em largura. *Gremlin* atende os tipos de consultas listadas anteriormente por meio de diferentes funções de manipulação de dados.

Em todos os exemplos "g" referencia o grafo exemplo e "V" é uma variável iteradora para referenciar os vértices do grafo "g". Cada passo da execução (denotado por um ponto) é uma função que opera nos resultados obtidos no passo anterior. O operador "out" busca as arestas que partem de um vértice e o "in" busca as arestas incidentes em um vértice. A operação "map" captura todas as propriedades do vértice ou aresta em questão.

Além das operações de leitura, uma base também pode receber operações de escrita. Alterações em dados que estão sendo utilizados concorrentemente por diversos usuários exigem um gerenciamento mais complexo por parte dos SGBDs. Para isso al-

[12]https://github.com/tinkerpop/gremlin/wiki
[13]*graph traversal language*

#	Descrição da Consulta	Expressão em *Gremlin*
1	Nomes dos integrantes da rede social	g.V.nome_pessoa
2	Idade do Paulo	g.V('nome_pessoa', 'Paulo').idade
3	Pessoa que o Paulo conhece	g.V('nome_pessoa', 'Paulo').out('conhece'). nome_pessoa
4	Empresa onde trabalham os conhecidos do Paulo	g.V('nome_pessoa', 'Paulo').out('conhece'). out('trabalha_para').nome_empresa
5	Nome e idade dos funcionários da empresa X	g.V('nome_empresa', 'X').in('trabalha_para'). map()

Tabela 1. Consultas em *Gremlin*

guns SGBDGs dão suporte a transações. Uma transação é um conjunto de operações que possui um objetivo específico [Date 2004]. Um SGBD transacional controla a concorrência das operações e também dá suporte à recuperação de dados caso ocorra alguma falha no processamento das transações. Diferentes modelos de consistência em transações podem ser usados nos SGBDs, como o ACID e o BASE [Vogels 2009]. O segundo modelo permite uma consistência dos dados mais relaxada quando comparado com o primeiro.

5. Sistemas Analisados

Todos os sistemas analisados nesse artigo são nativos, ou seja, todos usam listas de adjacências para armazenar grafos e executam algoritmos de exploração de grafos em suas consultas. Apesar deles apresentarem várias características em comum, cada um possui suas particularidades tornando-se adequado ou não para uma determinada aplicação. Nas descrições apresentadas a seguir foram consideradas nove características que podem influenciar na escolha de um sistema, sendo elas:

1. Tipo de licença: há basicamente dois grandes grupos, os sistemas abertos que disponibilizam seus códigos-fonte permitindo alterações, e os proprietários que possuem os códigos fechados permitindo somente o uso dos sistemas.
2. Suporte ao *Tinkerpop*: permite o uso de diversas ferramentas para manipulação de grafos desenvolvidas no projeto *Tinkerpop*.
3. Modelagem lógica: o modelo lógico usado no sistema influencia na facilidade de modelagem de uma aplicação pelo usuário e diz respeito ao modelo de grafo empregado.
4. Linguagem de consulta: define as operações e a forma de manipulação da base por meio de sua sintaxe.
5. Modelagem física: o modelo físico influencia no desempenho e implementação das operações sobre grafos.
6. Armazenamento físico: há dois tipos, em memória e em disco. O armazenamento físico influencia na capacidade de armazenamento e no desempenho das aplicações.
7. Suporte à transação: constitui uma característica essencial em aplicações que precisam garantir recuperação e controle de concorrência em suas requisições.
8. Tipo de arquitetura: há dois tipos, a centralizada e a distribuída. Em geral, a primeira atende às necessidades de aplicações simples e de pequeno porte e, a segunda, às de aplicações que demandam grandes volumes de requisições e de dados.

9. Suporte à replicação: importante para garantir a disponibilidade e a escalabilidade nas aplicações distribuídas.

As características listadas acima oferecem uma visão geral dos sistemas quanto à possibilidade de alteração das suas funcionalidades (característica 1), quanto à facilidade do uso destas funcionalidades (características 2, 3 e 4) e quanto às variáveis envolvidas no desempenho dos sistemas (características 5, 6, 7, 8 e 9). A seguir são apresentados os cinco SGBDGs selecionados para a análise.

5.1. InfiniteGraph

InfiniteGraph é um banco de dados proprietário da Objectivity [14]. Sua implementação foi feita em Java, com *core* em C++, e sua primeira versão foi lançada em 2010. InfiniteGraph é um banco transacional que oferece a arquitetura centralizada e a distribuída com um mecanismo de replicação com consistência relaxada dos dados.

O modelo lógico suportado pelo InfiniteGraph é o grafo de propriedades. Ele dá suporte ao *Tinkerpop* e também fornece uma API para consultas em Java. O modelo físico usado é orientado a objetos e, sendo assim, duas classes são usadas para armazenar os vértices e os nós de uma base, a *BaseVertex* e a *BaseEdge*, respectivamente. Ele oferece somente o armazenamento físico em disco.

5.2. Neo4j

A primeira versão do Neo4j foi lançada em fevereiro de 2010 pela empresa *Neo Technology* [15]. Sua implementação foi feita em Java e ele possui duas versões de licenciamento, uma aberta e outra proprietária. A diferença entre elas é que a proprietária tem código fechado, oferece suporte aos seus usuários e possui mais recursos para garantir o desempenho de grandes aplicações. O Neo4j é transacional e oferece os dois tipos de arquitetura, a centralizada e a distribuída com suporte à replicação.

O Neo4j também utiliza o modelo de grafos de propriedade e oferece suporte ao *Tinkerpop*. Dessa forma, tanto a definição dos esquemas das bases quanto a manipulação dos dados (carregamento e consulta) podem ser feitas por meio da linguagem *Gremlin*. Além disso, o Neo4j possui a linguagem de consulta proprietária *Cypher* e também dá suporte à SPARQL (SQL *for linked data*). O armazenamento físico pode ser tanto em memória quanto em disco e o modelo físico é baseado em repositórios chave-valor.

5.3. OrientDB

O OrientDB é um banco de dados em grafo aberto lançado comercialmente em 2011 pela empresa OrienTechnologies [16]. Ele foi implementado em Java, é transacional e dá suporte à arquitetura centralizada e distribuída com replicação.

O OrientDB oferece grafos de propriedades para a modelagem lógica e dá suporte ao *Tinkerpop*. A manipulação da base pode ser feita com o *Gremlin*, com o Java ou com uma adaptação do SQL desenvolvida no próprio projeto. O armazenamento físico dos dados pode ser feito em memória e em disco. Como todos os sistemas, ele usa a lista

[14] www.objectivity.cm
[15] neotechnology.com
[16] http://www.orientechnologies.com

livre de adjacências para viabilizar o processamento nativo das consultas, porém, diferentemente dos outros ele usa recursos de banco de dados de documentos e de orientação a objetos para o armazenamento físico dos vértices para garantir uma maior flexibilidade na estrutura dos dados a serem armazenados na base.

5.4. Titan

Titan é um sistema aberto mantido pela empresa Aurelius [17] desde 2012. Ele foi implementado em Java e pode ser classificado como um framework transacional para banco de dados em grafos. Ele dá suporte tanto à arquitetura centralizada quanto à distribuída.

O projeto do Titan é completamente dependente do *Tinkerpop*. Tanto a definição dos esquemas das bases quanto a manipulação dos dados (carregamento e consultas) são feitos por meio da linguagem *Gremlin*. Logo, a modelagem lógica é feita por meio de grafos de propriedades e o mapeamento por meio de listas de adjacência, que são essenciais ao *Gremlin*. Quanto a forma de armazenamento, Titan oferece a opção de armazenar os dados em memória, desde que a base seja relativamente pequena, e em disco. Os três principais banco de dados utilizados para armazenamento em disco são: Apache Cassandra, Apache HBase e Oracle Berkeley DB. Basicamente, os três utilizam pares chave-valor para armazenar fisicamente os dados. Cada banco tem suas características particulares e a escolha do banco determina a garantia transacional, o suporte à replicação e a escalabilidade de uma aplicação.

5.5. Trinity

O projeto Trinity encontra-se em fase de desenvolvimento pela Microsoft Research. Trinity [Shao et al. 2013] denomina-se um banco de dados distribuído em memória que também constitui uma plataforma computacional para suporte do processamento de consultas sobre grafos.

Na modelagem lógica, Trinity suporta grafos direcionados, não-direcionados e hipergrafos. Diferente dos outros sistemas, ele não dá suporte ao *Tinkerpop*. O processamento bem como a linguagem de consultas é dependente do *engine* de processamento constituído sobre a plataforma Trinity. Trinity.RDF [Zeng et al. 2013] é um exemplo de engine constituído sobre esta plataforma para o processamento de consultas SPARQL. O modelo físico adotado por Trinity corresponde ao modelo de um repositório chave-valor onde um par chave-valor representa um nó do grafo RDF e sua lista de adjacência.

6. Análise dos SGBDGs

A Tabela 2 apresenta os sistemas e as características usadas na comparação objeto deste artigo. Alguns sistemas possuem o tipo de licença aberto, mas, infelizmente, eles possuem códigos complexos e não disponibilizam documentação suficiente dos projetos inviabilizando alterações em suas funcionalidades. Quanto ao *Tinkerpop*, pode-se notar a sua importância nos sistemas atuais. Como consequência, predominam o grafo de propriedades como modelo lógico e o *Gremlin* como linguagem de consulta. Esse suporte amplo ao *Tinkerpop* facilita o uso dos sistemas pela comunidade de banco de dados em grafos. Como já citado nas seções anteriores, todos os sistemas fazem o mapeamento lógico-físico por meio de listas livre de adjacências, porém o modelo físico varia entre

[17] thinkaurelius.com

eles. Nesse quesito a maioria usa repositórios chave-valor devido à sua simplicidade e desempenho, porém, o OrientDB destaca-se pelo uso do modelo de documentos que dá suporte às aplicações que não têm esquemas de dados pré-definidos. A maioria dá suporte ao armazenamento físico dos dados tanto em memória quanto em disco, lembrando que o uso somente da memória trata de dados voláteis e limita o espaço de armazenamento dos dados, mas, em contrapartida, melhora o desempenho de consultas em grafos. Todos os sistemas são transacionais e dão suporte à distribuição de dados e à replicação com o objetivo de melhorar, respectivamente, a escalabilidade e a disponibilidade de suas aplicações.

SGBDG	Aberto/ Proprietário	*Tinkerpop*	Modelo lógico	Linguagem de consulta	Modelo físico	Memória/ Disco	Centralizada/ Distribuída	Transação/ Replicação
InfiniteGraph	proprietário	✓	propriedades	Gremlin	objetos	disco	ambas	ambas
Neo4j	ambos	✓	propriedades	Gremlin/Cypher/ SPARQL	chave-valor	ambos	ambas	ambas
OrientDB	aberto	✓	propriedades	Gremlin/SQL	documentos	ambos	ambas	ambas
Titan	aberto	✓	propriedades	Gremlin	chave-valor	ambos	ambas	ambas
Trinity	proprietário		hipergrafos	SPARQL	chave-valor	memória	distribuída	ambas

Tabela 2. SGBDGs e características

7. Trabalhos Relacionados

Alguns trabalhos na literatura comparam bancos de dados em grafos considerando características diversas. Angles apresentou dois estudos comparativos com diversos sistemas da época, um em 2008 [Angles and Gutierrez 2008] e outro em 2012[Angles 2012]. Os estudos apresentam uma análises em nível lógico e comparam, segundo Angles, características de modelos de dados, sendo elas: estruturas de dados, formas de consultas e restrições de integridade. Por outro lado, com foco em desempenho, [Ciglan et al. 2012] apresentou uma análise de cinco sistemas por meio de um *benchmark* e, [McColl et al. 2013] publicou um breve relatório técnico considerando diversos sistemas abertos. O primeiro considerou somente o processamento de consultas e, o segundo, consultas e atualizações. Nesse artigo o critério de seleção dos sistemas foi o processamento de consultas nativo sobre grafos e o objetivo da comparação apresentada é mostrar uma visão geral quanto à possibilidade dos usuários alterarem as funcionalidades dos sistemas, quanto à facilidade do uso das funcionalidades dos sistemas e quanto às variáveis envolvidas no desempenho dos sistemas.

8. Conclusão

Em geral, os sistemas nativos oferecem um melhor desempenho no processamento de consultas quando comparado com os não-nativos, porém, várias características diferenciam esses sistemas. Na comparação objeto desse artigo pode-se notar que o suporte ao *Tinkerpop* é uma tendência entre eles devido à facilidade de manipulação de dados que as ferramentas do projeto oferecem aos usuários. Desta forma, o modelo de grafos de propriedades e a linguagem *Gremlin* estão presentes na maioria dos sistemas analisados, apesar de ainda não existir um padrão de linguagem de consulta para banco de dados em grafos. Outro destaque é o suporte à distribuição de dados. Todos procuram atender às necessidades das aplicações atuais que demandam grandes volumes de requisições e de dados. O suporte à distribuição e à replicação oferece uma maior escalabilidade e disponibilidade às aplicações.

Agradecimentos Este trabalho foi parcialmente financiado pela Capes, Araucária e Microsoft Azure.

Referências

Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2009). Sw-store: A vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18(2):385–406.

Aho, A. and Ullman, J. (1995). *Foundations of Computer Science: C Edition*. Principles of Computer Science Series. W. H. Freeman.

Angles, R. (2012). A comparison of current graph database models. In *ICDE Workshops*, pages 171–177.

Angles, R. and Gutierrez, C. (2008). Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.

Ciglan, M., Averbuch, A., and Hluchý, L. (2012). Benchmarking traversal operations over graph databases. In Kementsietsidis, A. and Salles, M. A. V., editors, *ICDE Workshops*, pages 186–189. IEEE Computer Society.

Date, C. (2004). *Introdução a Sistemas de Bancos de Dados*. Campus.

Iordanov, B. (2010). Hypergraphdb: A generalized graph database. In Shen, H., Pei, J., Özsu, M., Zou, L., Lu, J., Ling, T.-W., Yu, G., Zhuang, Y., and Shao, J., editors, *Web-Age Information Management*, volume 6185 of *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin Heidelberg.

Liptchinsky, V., Satzger, B., Zabolotnyi, R., and Dustdar, S. (2013). Expressive Languages for Selecting Groups from Graph-structured Data. In *Proceedings of the 22Nd International Conference on World Wide Web*, pages 761–770.

McColl, R., Ediger, D., Poovey, J., Campbell, D., and Bader, D. A. (2013). A brief study of open source graph databases. *CoRR*, abs/1309.2675.

Neumann, T. and Weikum, G. (2010). The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19(1):91–113.

Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases*. O'Reilly Media.

Rodriguez, M. A. and Neubauer, P. (2010). The graph traversal pattern. *CoRR*, abs/1004.1001.

Rodriguez, M. A. and Shinavier, J. (2008). Exposing multi-relational networks to single-relational network analysis algorithms. *CoRR*, abs/0806.2274.

Shao, B., Wang, H., and Li, Y. (2013). The Trinity Graph Engine. In *SIGMOD international conference on Management of data*.

Vogels, W. (2009). Eventually consistent. *Commun. ACM*, 52(1):40–44.

Zeng, K., Yang, J., Wang, H., Shao, B., and Wang, Z. (2013). A Distributed Graph Engine for Web Scale RDF Data. *VLDB Endowment*.