Provendo Integridade de Dados em Bancos de Dados de Grafos

1st Fábio Reina
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Brasil
f.reina@grad.ufsc.br

2nd Aldelir Fernando Luiz
Campus Blumenau
Instituto Federal Catarinense (IFC)
Blumenau, Brasil
aldelir.luiz@ifc.edu.br

3rd Luciana de Oliveira Rech
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Brasil
luciana.rech@ufsc.br

4th Hylson Vescovi Netto
Campus Blumenau
Instituto Federal Catarinense (IFC)
Blumenau, Brasil
hylson.vescovi@ifc.edu.br

5th Frank Siqueira
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Brasil
frank.siqueira@ufsc.br

***Resumo*—A profusão da Internet no cotidiano das pessoas e organizações tem desencadeado a produção de uma massa de dados altamente conectada, o que estimula o uso de tecnologias de banco de dados não tradicionais. Neste contexto, os bancos de dados de grafos vêm ganhando notoriedade, por apresentarem facilidades quanto à modelagem de dados complexos, bem como por proverem um melhor desempenho acerca do processamento de consultas. Por outro lado, por se tratar de uma tecnologia relativamente nova, as implementações de sistemas de gerência de bancos de dados de grafo ainda carecem de mecanismos para lidar com a manutenção da integridade de dados. Neste sentido, este trabalho introduz uma solução para provimento de integridade (p. ex.: verificação e manutenção) dos dados armazenados em um banco de dados de grafos.**

***Index Terms*—banco de dados de grafos, integridade de dados, criptografia**

I. Introdução

O crescimento espetacular da Internet – que na atualidade interliga mais de metade da população global – trouxe grandes avanços tecnológicos nas áreas de computação e comunicação, principalmente no que tange aos sistemas de computação distribuída. Neste sentido, a alta utilização de tecnologias Web, p. ex.: as redes sociais online, tem produzido um imenso volume de dados que não pode ser ignorado, uma vez que não apenas os indivíduos, mas também as organizações têm se tornado cada vez mais dependentes destes dados. À vista disso, aplicações modernas demandam por uma gestão mais eficiente dos dados, uma vez que a tecnologia de banco de dados tradicional (i.e., relacional) não é completamente adequada para lidar com modelos de dados modernos e não convencionais.

É notável que os sistemas de gerenciamento de banco de dados relacionais (SGBDR), ao longo dos anos, se intensificaram como um padrão *de facto* no que se refere ao armazenamento e gerenciamento de dados [1]. Isto decorre, principalmente, pela maturidade da tecnologia de banco de dados em questão. Por outro lado, a ausência de regularidade (ou mesmo a dinamicidade) verificada nos modelos de dados de aplicações mais modernas, impõem requisitos que dificilmente conseguem ser atendidos pelos SGBDRs [2]. À vista disso, surge uma nova classe de bancos de dados denominada NoSQL (*Not Only SQL*) [3], no intuito de romper algumas barreiras encontradas nos SGBDRs. Em tempos onde estão em evidência modelos como os de computação intensiva de dados e computação em nuvem, os bancos de dados NoSQL consistem em alternativas bastante atrativas.

No cerne da tecnologia NoSQL estão as aplicações baseadas em grafos, as quais têm se tornado bastante populares, como é o caso das redes sociais online. Não obstante, o modelo de dados baseado em grafos tem sido adotado para modelar/especificar uma ampla gama de aplicações que visam a solução de problemas práticos, tal como mineração de dados em redes sociais. Tal modelo é composto por diferentes elementos que se relacionam através de objetos de conexão, de modo que o armazenamento de dados é realizado a partir de uma sequência de nodos e suas relações – opcionalmente contendo propriedades –, que quando combinadas dão origem a um grafo [4].

Por outro lado, novas tecnologias de bancos de dados introduzem novos desafios no que tange a manutenção consistente e correta dos dados, em face aos problemas reais que podem ser

manifestados no âmbito do sistema computacional. Um exemplo concreto são falhas de natureza arbitrária/bizantina [5] em sistemas de bancos de dados [6], as quais podem comprometer a integridade sem comprometer a disponibilidade dos dados. Um caso real da manifestação destas falhas são os *bit flips* [7], causados por interferências/perturbações eletromagnéticas, variações de tensão, etc. e induzidas principalmente pelas nuvens de radiação (i.e., raios cósmicos) que emanam em ambientes de *datacenters* [8]. No caso, tais falhas podem afetar a semântica dos dados ou mesmo ocasionar a perda destes – um feito que pode causar grande impacto para as aplicações e usuários.

Isto posto, é de grande relevância o desenvolvimento de mecanismos e técnicas para a construção de sistemas de bancos de dados modernos mais seguros, que sejam capazes de detectar alterações arbitrárias ou mal-intencionadas, a fim de garantir não somente a integridade, mas também a consistência dos dados. Neste contexto, este artigo descreve a proposição de um método inovador para verificar a integridade dos dados no âmbito de um banco de dados de grafo. Assim, o restante do artigo está organizado da seguinte maneira. A Seção II faz uma breve apresentação dos conceitos fundamentais para a compreensão do trabalho; uma breve discussão acerca dos trabalhos correlatos é realizada na Seção III; nas Seções IV e V é apresentado, em detalhes, a especificação do método que provê integridade sobre a manutenção de dados em um banco de dados de grafo; a Seção VI apresenta os aspectos de implementação de um protótipo, bem como uma avaliação de desempenho e discussão acerca dos resultados obtidos. Por fim, a Seção VII conclui o trabalho.

II. Aspectos Teóricos

Nesta seção é realizada uma revisão da literatura, a fim de elucidar alguns dos conceitos pertinentes ao trabalho em lide.

A. Conceitos Básicos do Modelo de Grafos

Um primeiro conceito a ser abordado é o de grafo, que por sua vez, é definido por um conjunto finito e não vazio de vértices V, tal que, sobre V há uma relação binária E [9]. Portanto, um conjunto V e uma relação binária sobre este é formalizada por um grafo $G(V, E)$. Os elementos de V são representados por pontos, enquanto que a relação binária E é representada por um conjunto de pares ordenados $(v, w) \in E$, de tal modo que, v, $w \in E$ é denotada por uma linha que liga v a w – a esta linha dá-se o nome de arco ou aresta.

No caso específico do modelo de grafos para representação em banco de dados, tal modelo é composto por três elementos básicos, são eles: (*i*) os nodos, que correspondem aos vértices; (*ii*) os relacionamentos, que denotam as arestas/arcos, e; (*iii*) as propriedades, que caracterizam os atributos especificados tanto para os nodos, como para os relacionamentos. À vista disso, um banco de dados pode ser representado por um multidígrafo rotulado [9], cujos pares de nodos podem ser ligados por uma ou mais arestas. Não obstante, os tipos de grafos suportados em um banco de dados são os grafos simples, hipergrafos, grafos aninhados e o grafo de propriedades.

Por sua vez, um sistema de gerência de banco de dados (SGBD) de grafo consiste numa componente de software designada a criar, recuperar, atualizar e excluir dados sob a representação de grafos. As duas principais características de um SBGD de grafo são o "armazenamento nativo de grafo" e o "processamento nativo de grafo'. O processamento nativo é possível, desde que cada elemento possua um apontador para o elemento adjacente. Por outro lado, o armazenamento nativo ocorre a partir de uma 3-tupla, onde cada campo armazena um dos elementos que constituem o grafo (i.e., o(s) nodo(s), o(s) relacionamento(s) e a(s) propriedade(s)). A Figura 1 – adaptada de [4] – ilustra a estrutura empregada em um sistema de armazenamento baseado em grafo.

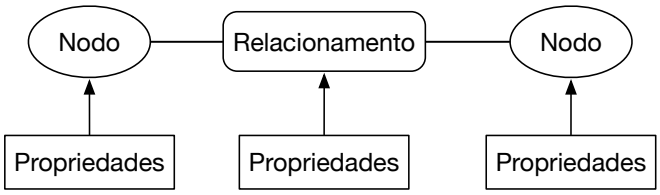


Figura 1. Representação de grafo como um modelo de dados.

Por outro lado, a despeito da designação de alguns SGBDs como sendo de grafos, o armazenamento (de dados) subjacente nativo (como um grafo) não é provido por todos os produtos. Logo, naqueles que não o fazem realiza-se um mapeamento do grafo para uma representação consonante com o modelo de dados adotado pelo mesmo. Alguns SGBDs que provêem suporte ao armazenamento nativo de grafos são Neo4J, InfoGrid, HyperGraphDB, Titan, Trinity e Amazon Neptune.

B. Um Panorama sobre Integridade de Dados

Integridade é um dos atributos requeridos para se alcançar a confiabilidade em sistemas computacionais (i.e., *dependability*) [10]. Embora a integridade de dados seja um problema inerente ao campo de segurança da informação, especificamente do campo de criptografia, mecanismos para o provimento de integridade e proteção de dados são tipicamente empregados não apenas em sistemas de bancos de dados, mas também em sistemas distribuídos. A violação de tal atributo pode culminar numa corrupção de dados. Inúmeras podem ser as causas da corrupção, sendo que as mais comuns são oriundas de causas benignas – p. ex.: problemas nos canais da rede de comunicação, corrupção em disco ou em memória RAM devido a efeitos físicos, ou devido a *bugs* no software [7], [8] – ou mesmo maliciosas – devido a ataques de segurança bem sucedidos [6].

Umas das soluções mais populares para provimento de integridade consiste na codificação da informação (ou dado), a partir do uso de uma cifra e uma chave, ambos conhecidos pelas partes interessadas na informação/dado. De um modo geral, a integridade é obtida na prática a partir do uso de códigos de autenticação de mensagens (do inglês, *Message Authentication Codes*, ou simplesmente MACs) [11]. Em suma, o MAC provê meios que possibilitam a verificação e detecção

de operações de manipulação de dados não autorizadas, tais como inclusão, exclusão e atualização.

Em sua essência, um MAC é construído a partir de uma função de resumo criptográfico (i.e., *hash* [12]), que por sua vez, consiste em um algoritmo que efetua o mapeamento de uma cadeia de *bytes* de tamanho arbitrário/variável (de entrada), numa cadeia de *bytes* de tamanho fixo (de saída). Um aspecto particularmente interessante acerca do resumo criptográfico, é que não é possível obter a cadeia de *bytes* de entrada, a partir da cadeia produzida na saída. À vista disso, uma função de resumo criptográfico é caracterizada como "função de caminho único" (ou *one-way function*).

Por sua vez, uma função de resumo criptográfico resistente a colisões, denotada por $\mathcal{H}$, consiste em uma função que mapeia uma entrada de tamanho arbitrário em uma saída de tamanho fixo [13]. Uma função de resumo criptográfico deve satisfazer a duas propriedades [13]:

- **Resistência a colisão**: é computacionalmente impraticável encontrar duas mensagens $m \neq m'$, tal que $\mathcal{H}(m) = \mathcal{H}(m')$;
- **Unidirecionalidade**: dada uma saída resultante da aplicação da função de resumo, é computacionalmente impraticável obter a entrada que produziu tal saída.

São exemplos de funções de resumo criptográfico a MD5, SHA1 e SHA256, as quais produzem saídas com tamanhos de 128 bits, 160 bits e 256 bits, respectivamente.

III. Revisão da Literatura

De um modo geral, no âmbito de sistemas de computação o modelo de grafos tem sido adotado para viabilizar o processamento de grandes volumes de dados. Exemplos concretos da aplicação de grafos em sistemas distribuídos são o Pregel [14] e o Greft [15]. O Pregel consiste em um modelo para o processamento de grafos desenvolvido para ser utilizado em *clusters* com milhares de máquinas, o que confere a ele a capacidade de processar grafos com vértices na ordem dos bilhões. O Greft, por sua vez, especifica e implementa um sistema para o processamento distribuído de grafos baseado no modelo do Pregel, com a peculiaridade de tolerar faltas arbitrárias acidentais de um modo bastante eficiente. Não menos importante, bancos de dados de grafos já foram alvo de incorporação de estratégias de replicação para tolerância a faltas [16].

De outro modo, questões relacionadas à segurança com aplicação em sistemas de banco de dados já foram discutidas na literatura, uma vez que as comunidades de banco de dados e de segurança da informação têm realizado pesquisas em torno do tema. A literatura dispõe de esquemas para verificação de integridade de dados, em que a maioria é desenvolvida para bancos de dados relacionais. Logo, a manutenção da integridade dos dados em um banco de dados de grafos ainda é um problema desafiador, que carece de investigação. Uma vez que a violação da integridade pode ser oriunda de diferentes causas (vide Seção II-B), o tema em lide consiste em algo desafiador no âmbito de sistemas de bancos de dados modernos.

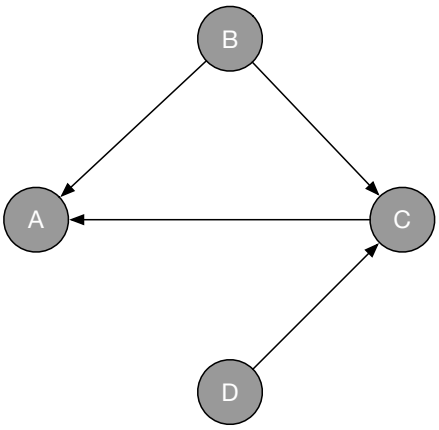
Uma proposta interessante encontrada na literatura é a técnica conhecida como *Merkle Hash* [17], a qual tem sido amplamente utilizada no contexto de árvores e grafos direcionados acíclicos. O resumo baseado na técnica *Merkle Hash* é computado por uma função $mh()$, da seguinte maneira. Para um nodo x de uma árvore $A(V, E)$, $mh(x) = \mathcal{H}(c_x)$, se x é uma folha; ou $mh(x) = \mathcal{H}(mh(y_1)||mh(y_2)||...||mh(y_n))$, se x for um nodo intermediário (i.e., não folha). No caso, $y_{1,..,n}$ são os filhos de x em A, visitados da esquerda para a direita, c_x denota o conteúdo do nodo x e $||$ uma operação de concatenação. É imperioso salientar que o processamento do resumo *Merkle Hash*, ocorre das folhas para a raiz.

No que concerne aos banco de dados de grafo, a literatura carece de propostas pertinentes à segurança/integridade. Os únicos trabalhos são os de Arshad *et al.* [18], [19], cuja autoria é dos mesmos pesquisadores. Estes trabalhos empregam elementos criptográficos, a fim de prover sigilo de informações, quando parte do grafo é compartilhado. No caso, os autores propõem uma técnica para cifrar o grafo, a partir da árvore de travessia gerada. Tal árvore é construída numa operação de busca em profundidade no grafo, onde são atribuídos valores de "pré-ordem" e "pós-ordem" para cada nodo visitado, e a partir daí é gerado um resumo criptográfico (i.e., *hash*) da estrutura resultante. Todavia, a despeito da funcionalidade, a técnica é deficiente, no sentido de que não lida com a dinamicidade em grafos, isto é, qualquer modificação no grafo requer a geração de um novo resumo criptográfico de todo o grafo.

Por fim, um trabalho desenvolvido para bancos de dados relacionais, porém, de particular interesse é o trabalho de Silvério *et al.* [20], onde os autores propõem uma solução bastante simples – porém útil – para a manutenção da integridade em bancos de dados relacionais. A ideia consiste na adição de uma nova coluna em cada tabela que integra o *schema* do banco de dados. Nesta coluna se armazena um valor de MAC (vide Seção II-B), que é calculado a partir da concatenação dos campos que compõem cada linha da tabela. Note que o uso desta coluna/atributo garante a integridade de cada linha, individualmente, o que, por conseguinte, incorre numa proteção contra atualizações indevidas/espúrias. De mesmo modo, o provimento de integridade em operações de inclusão e exclusão se dá a partir da criação de mais uma coluna, a fim de armazenar o valor de um *Chained-MAC* (CMAC). Nesta coluna, o valor atribuído é calculado a partir da fórmula $CMAC_n = MAC(k, (MAC_{n-1} \oplus MAC_n))$, de modo que n e $\oplus$ consistem, respectivamente, no número da linha atual da tabela e na operação "ou exclusivo" (i.e., XOR). Com isso, torna-se possível a detecção de inclusão e exclusão não autorizadas.

IV. Aspectos Gerais do Método Proposto

A despeito dos trabalhos correlatos mencionados (vide Seção III), em que a maioria se baseia na utilização de técnicas de resumos criptográficos (ou $hashing$), a construção destes resumos em grafos ainda é um problema desafiador, o que se deve, principalmente: (*i*) aos grafos poderem conter ciclos; (*ii*) as mudanças em nodos afetarem muitos outros nodos, e; (*iii*)



$$\mathcal{H}(A) = \mathcal{H}(\texttt{conteúdo de } A)$$
$$\mathcal{H}(B) = \mathcal{H}(\texttt{conteúdo de } B)$$
$$\mathcal{H}(C) = \mathcal{H}(\texttt{conteúdo de } C)$$
$$\mathcal{H}(D) = \mathcal{H}(\texttt{conteúdo de } D)$$

$$cHash(A) = \mathcal{H}(\mathcal{H}(A) \,||\, \mathcal{H}(B) \,||\, \mathcal{H}(C))$$
$$cHash(B) = \mathcal{H}(\mathcal{H}(B) \,||\, \mathcal{H}(A) \,||\, \mathcal{H}(C))$$
$$cHash(C) = \mathcal{H}(\mathcal{H}(C) \,||\, \mathcal{H}(A) \,||\, \mathcal{H}(B) \,||\, \mathcal{H}(D))$$
$$cHash(D) = \mathcal{H}(\mathcal{H}(D) \,||\, \mathcal{H}(C))$$

Figura 2. Exemplo de grafo com os respectivos cálculos dos valores de *hash* e *cHash*.

por um grafo poder ser compartilhado apenas parcialmente, em termos de subgrafos. Isto é, em um sistema real há a possibilidade de partilhar apenas um subgrafo de interesse, em vez de o grafo por completo. Deste modo, a aplicação de resumos criptográficos em grafos cíclicos consiste em algo demasiadamente complexo, já que um grafo não pode ser ordenado topologicamente. Logo, a dificuldade da aplicação de resumos criptográficos advém da preservação do(s) ciclo(s) presente(s) na estrutura.

Isto posto, esta seção descreve a especificação de um método para prover a manutenção da integridade de dados em um banco de dados de grafo. O método proposto é formalizado como um protocolo, que efetua a aplicação de um resumo criptográfico (i.e., $hash$), em que o cálculo é realizado a partir da concatenação dos valores dos atributos de um nodo do grafo. O resultado do resumo criptográfico é armazenado como um novo atributo (vide Figura 3), de modo a permitir a identificação de operações de atualização espúrias – i.e., não autorizadas.

Ademais, no intuito de evitar operações espúrias de inclusão e/ou exclusão, o protocolo efetua o cálculo de uma nova cadeia de resumos criptográficos (i.e., um *ChainedHash* [21]), a partir da concatenação do $\mathcal{H}_{nodo}$ do nodo atual com os $\mathcal{H}_{nodo}$ de cada um dos nodos vizinhos – vide equação abaixo. Com isso, é criada uma corrente que liga todos os nodos, de modo que qualquer operação de inclusão ou exclusão de nodo ou de relação entre nodos requer o recálculo do $cHash$ para todos os vértices/nodos vizinhos. No caso de uma operação de atualização regular sobre o grafo, precisam ser recalculados apenas os valores $cHash$ dos nodos vizinhos em relação ao nodo que fora atualizado. Similar ao que ocorre com o resumo criptográfico do nodo, o valor da cadeia/corrente de resumos criptográficos dos nodos envolvidos é armazenada em um novo atributo, que é declarado para tal finalidade. A Figura 3 ilustra a representação de um nodo para uso do método proposto.

$$cHash_a = \mathcal{H}(\mathcal{H}(a)||\mathcal{H}(b)||\mathcal{H}(a \rightarrow b)||...||\mathcal{H}(n)||\mathcal{H}(a \rightarrow n))$$

É digno de nota que o protocolo proposto, além de verificar a completude dos dados em cada nodo do grafo, elimina vulnerabilidades isoladas que poderiam ser exploradas e culminar na inclusão e/ou exclusão de nodos e relacionamentos (i.e., vértices e arcos/arestas). Isto é, ele evita que uma parte não autorizada venha a remover ou incluir uma nova entidade no banco de dados, sem que a operação possa ser devidamente detectada. Se assim não o fosse, a manutenção da integridade estaria comprometida. Note que a ideia por trás do uso de cadeia de resumos é análoga ao princípio empregado para a agregação de blocos na corrente de blocos do sistema *blockchain* [22]. A Figura 2 ilustra um exemplo de operação do protocolo proposto, sobre um grafo com quatro nodos e algumas relações entre eles.

Um aspecto crucial para o funcionamento do protocolo é a ordem de leitura dos nodos vizinhos. Tal aspecto decorre do fato de que cada sequência de nodos gera um resumo criptográfico distinto. Logo, é essencial que os nodos sejam ordenados por algum atributo, antes de serem submetidos ao protocolo. Para o caso de um banco de dados organizado por um grafo direcionado, deve-se considerar a informação pertinente ao sentido da(s) aresta(s), do contrário, uma inversão de direção pode vir a não ser detectada. Esta restrição é devidamente especificada na equação acima, em que, seja um nodo a cujos vizinhos são os nodos entre b e n; $a \rightarrow b$ denota uma aresta entre a e b e $a \rightarrow n$ uma relação entre a e n.

CHash: 0xF32..
Hash: 0xD7E..
Atributo_1: 153
Atributo_2: 'ab'
...
Atributo_N: 2,7

Figura 3. Representação de nodo empregada na estratégia proposta.

Embora a abordagem empregada na especificação do protocolo proposto permita identificar operações espúrias realizadas sobre o grafo no banco de dados, em um caso particular onde o grafo contém subestruturas desconexas (vide Figura 4(a)) não é possível prover garantias de integridade em completude. Neste

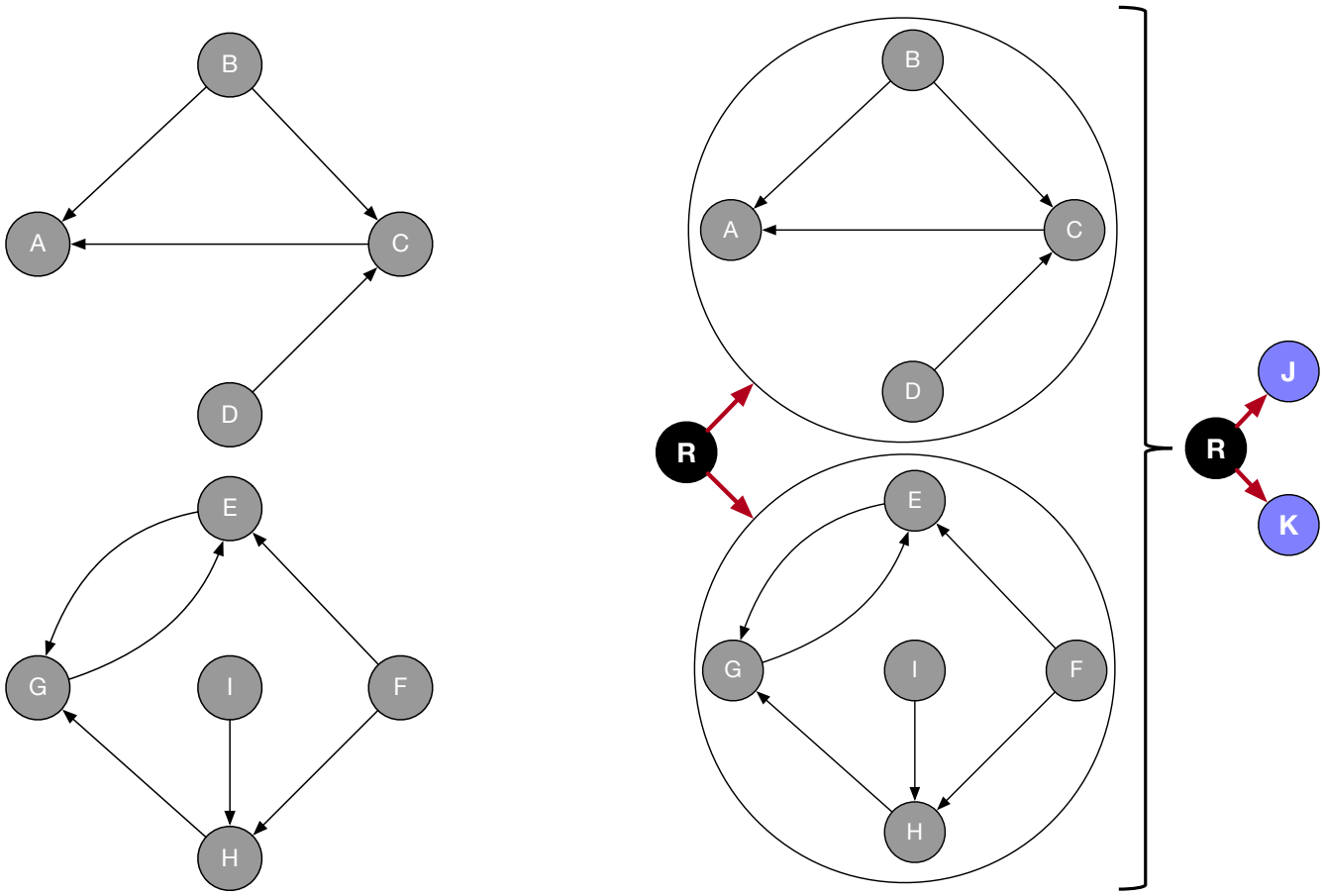


Figura 4. Exemplos de **(a)** grafo com subestruturas desconexas; e **(b)** grafo com um nodo raiz entre duas subestruturas, transformando-as num único grafo.

caso, seria permitido que uma subestrutura viesse a sofrer uma operação de remoção completa, sem ser devidamente identificada/detectada. Porém, o protocolo proposto lida com tal situação da seguinte maneira. É criado um novo nodo intermediário/central entre as duas estruturas, o qual atuará como uma espécie de "raiz" de toda a estrutura. Com isso, este nodo conecta-se aos grafos desconexos, de tal modo que as subestruturas passam a pertencer ao grafo cujo nodo central é o nodo raiz, recém criado. Tal situação é ilustrada pela Figura 4(b).

No que tange ao ponto ora explanado, a relação do nodo raiz com os demais é realizada a partir do uso de duas estratégias, denominadas por **LigaTodos** e **LigaMenor**. As estratégias se diferenciam pela maneira como se dá a ligação do nodo raiz, introduzido nos subgrafos, aos demais nodos das subestruturas. No primeiro caso, o nodo raiz se conecta a todos os nodos do grafo, enquanto que, no segundo caso, o nodo raiz se conecta apenas ao nodo de menor id (p. ex.: menor índice) de cada subestrutura que compõe o grafo. A partir daí, os valores referentes a cadeia de resumos criptográficos (i.e., $cHash$) são atualizados, onde é acrescentado à cadeia o conteúdo do nodo raiz ora introduzido. Esta estratégia cria um elo de ligação entre todas as partes do grafo, de modo a prover a integridade da(s) informação(ões) armazenadas no banco de dados. O cálculo do $cHash$ para o nodo raiz é especificado pela fórmula $cHash_{raiz} = \mathcal{H}(raiz)||\mathcal{H}(a)||...||\mathcal{H}(n)$, em que os nodos de a até n denotam todos os vizinhos ao nodo raiz.

V. Base Algorítmica do Método Proposto

A fim de corroborar para com a compreensão do método proposto, esta seção descreve em detalhes sua formalização, o que se dá a partir da especificação do(s) algoritmo(s), em que se descreve o modo pelo qual as operações se sucedem. No caso, o método é composto por cinco operações, das quais três são básicas para a manipulação dos dados e duas são operações de suporte às demais. O detalhamento das operações é realizado no Algoritmo 1.

A primeira operação básica consiste na adição de nodo, a qual é especificada entre as linhas 1 e 8. A criação de um novo nodo (i.e., vértice) ocorre de forma simples e direta. No caso, após os passos elementares requeridos para a criação do nodo, p. ex.: inicialização, atribuição dos valores aos respectivos atributos, etc. e ligação do mesmo aos seus respectivos vizinhos (linhas 2 e 3), o algoritmo produz um *hash* baseado nos valores presentes nos atributos deste nodo, a partir da invocação à função $generateHash$ – linha 4. Observe que, se

Algoritmo 1 Operações sobre o grafo com manutenção da integridade.

{ **Adição de Nodo** }

entrada: nodeID, nodeData {dados que irão compor o nodo, p. ex.: atributos e respectivos valores}
executa um passo que compreende todos os comandos requeridos para a criação do nodo
$nodes := \{\text{nodeID}\} \cup createRelationships()$ {função cria os relacionamentos do nodo e retorna a lista dos vizinhos}
`call` $generateHash(\text{nodeID})$ {cálculo do resumo criptográfico do nodo}
for all $node \in nodes$ **do**
  if $exists\_properties(\text{nodeID} \rightarrow node)$ **then**
    $hash := generateHash(getProperties(\text{nodeID} \rightarrow node))$ {se há propriedade(s), calcula o resumo criptográfico}
  `call` $generateCHash(node)$ {cálculo da cadeia de resumos criptográficos do subgrafo}

{ **Modificação/Atualização de Nodo** }

entrada: nodeID
executa o(s) passo(s) que consiste(m) na(s) modificação(ões) efetuada(s) sobre o nodo
`call` $generateHash(nodeID)$
$nodes := \{\text{nodeID}\} \cup getNodeNeighbors(nodeID)$ {função retorna a lista dos vizinhos de nodeID}
for all $node \in nodes$ **do**
  if $exists\_properties(\text{nodeID} \rightarrow node)$ **then**
    $hash := generateHash(getProperties(\text{nodeID} \rightarrow node))$ {se há propriedade(s), calcula o resumo criptográfico}
  `call` $generateCHash(node)$

{ **Exclusão de Nodo** }

entrada: nodeID
$neighbors := getNodeNeighbors(nodeID)$ {retorna todos os vizinhos do nodo fornecido como argumento}
executa um passo que consiste no comando que remove o nodo, tal que $id = nodeID$
if $neighbors \neq \perp$ **then**
  for all $neighbor \in neighbors$ **do**
    `call` $generateCHash(neighbor)$

{ **Geração/Atualização do Resumo Criptográfico (*Hash*)** }

function $generateHash(elementID)$
entrada: elementID
$attributes :=$ lista que contém os atributos do elemento fornecido como entrada, i.e., nodo ou propriedade
$values := \perp$
for all $attribute \in attributes$ **do**
  $values := values \mid\mid valueOf(attribute)$ {concatenação dos valores dos atributos}
$digest := hash(values)$ {cálculo do resumo criptográfico}
executa um passo que consiste no comando que atualiza o nodo/propriedade, com o $hash$ computado no passo anterior

{ **Geração/Atualização da Cadeia de Resumos Criptográficos (*cHash*)** }

function $generateCHash(nodeID)$
entrada: nodeID
$neighbors :=$ lista contendo os vizinhos do Nodo fornecido como entrada, i.e., nodeID
$hashes := hash_{nodeID}$ {inicializa a cadeia com o *hash* do nodo fornecido na entrada, i.e., nodeID}
for all $neighbor \in neighbors$ **do**
  if $exists\_properties(\text{nodeID} \rightarrow neighbor)$ **then**
    $hash_{properties} := getHashProperties(\text{nodeID} \rightarrow neighbor)$
    $hashes := hashes \mid\mid hash_{properties}$ {concatenação dos valores de *hashes* da(s) propriedade(s)}
  $hashes := hashes \mid\mid hash_{neighbor}$ {concatenação dos valores de *hashes* de cada nodo vizinho}
$cHash := hash(hash_a \mid\mid hashes)$ {cálculo do resumo criptográfico da cadeia}
executa um passo que consiste no comando que atualiza o nodo, com o valor do $cHash$ calculado no passo anterior

houver vizinhos ligados ao nodo ora incluído (i.e., nodeID), é necessário constatar se há propriedade(s) no relacionamento e, se houver, calcular também o resumo criptográfico para esta(s). Do mesmo modo, caso haja vizinhos, as cadeias de resumos criptográficos destes também deverão ser atualizadas, uma vez que a disposição do grafo foi modificada (vide Figura 2). Estas duas ações são efetuadas no código entre as linhas 5 e 8.

O código entre as linhas 9 e 16 trata da especificação da operação de modificação/atualização de nodo. É importante salientar que, entende-se por atualização de nodo, qualquer modificação de valor de algum atributo daquele nodo. Uma vez que há modificação de valores, há reflexo no valor do resumo criptográfico daquele nodo, bem como no valor da cadeia de resumos criptográficos no nodo modificado e dos vizinhos daquele nodo. Logo, é requerido o recálculo tanto do resumo como da cadeia de resumos, a fim de computar os valores modificados. O cálculo do resumo do nodo modificado é realizado na linha 11, enquanto que o cálculo da cadeia de resumos, tanto para o nodo modificado quanto para os nodos vizinhos é efetuado entre as linhas 13 e 16. É digno de nota que, assim como ocorre na adição do nodo, se houver propriedade(s) entre o nodo modificado e seus vizinhos, estes também são considerados na construção da cadeia (linhas 14 e 15).

Um aspecto importante quanto ao código que trata das operações de inclusão e de modificação do nodo é que, em ambos os casos, se o nodo em questão não possuir relacionamentos com outros nodos (i.e., vizinhos), o cálculo da cadeia de resumos criptográficos será construído apenas pelo resumo criptográfico do nodo em questão. Note que, nos dois casos, mesmo que não haja nodos vizinhos, os algoritmos entram nos laços das linhas 5 e 13, já que o nodo atual também faz parte das listas onde ocorrem as iterações pelos laços (i.e., variável $nodes$ que é inicializada nas linhas 3 e 12. Quanto à operação no código entre as linhas 17 e 22, ele trata da exclusão de um nodo. Da mesma forma como ocorre na inclusão e modificação, a exclusão requer a atualização das cadeias de resumos (i.e., *cHash*) daqueles nodos que mantém ligação com aquele que se deseja excluir – linhas 20 a 22. A mesma situação ocorre quando apenas se pretende desfazer uma relação entre nodos (i.e., uma aresta). Todavia, no caso de um nodo, todos os vizinhos devem ser identificados, enquanto que para uma relação, apenas os dois nodos conectados/relacionados são identificados. Tal identificação, que ocorre na linha 18, é necessária para orientar a atualização da cadeia de resumos criptográficos, uma vez que a composição do grafo sofre modificação.

No que segue, as duas operações restantes são aquelas que proveem suporte às demais apresentadas, que são as gerações do resumo criptográfico do nodo (*hash*) – linhas 23 a 29 – e da corrente/cadeia de resumos criptográficos (*cHash*), especificada nas linhas 30 a 39. De maneira simples e intuitiva, a operação $generateHash$ calcula o valor de *hash* a partir da concatenação dos valores dos atributos do elemento (p. ex.: nodo ou propriedade(s)) fornecido como argumento (linhas 26 a 28). Por sua vez, a função $generateCHash$ é responsável pela construção da cadeia de resumos (*cHash*), de modo que o valor é calculado a partir do valor do resumo criptográfico do nodo fornecido como argumento, concatenado aos valores dos resumos criptográficos de cada um dos seus vizinhos, bem como aos resumos criptográficos das propriedades dos relacionamentos, quando houver – código entre as linhas 32 e 37. Note que, a cada iteração do laço, a cadeia de resumos é alimentada com o resumo do nodo/vértice vizinho (linha 37), e/ou da(s) propriedade(s) existente(s) no relacionamento/aresta entre o nodo atual e seu(s) vizinho(s) – linhas 34 a 36.

VI. Implementação de Protótipo, Avaliação e Resultados

No intuito de averiguar não apenas a viabilidade e desempenho, mas sobretudo, de elaborar uma prova de conceito acerca do mecanismo para provimento de integridade ora proposto, um protótipo foi implementado a partir das especificações do algoritmo formalizado na Seção V. A base de código deste protótipo foi implementada com o uso da linguagem Java, a qual está disponível no endereço (`https://github.com/fmreina/neo4j`). Os experimentos foram realizados no SGBD de grafos Neo4J, versão 3.1.4. Para tanto, o ambiente consistiu da seguinte configuração: um máquina física equipada com 1 (um) microprocessador Intel-Core i7-4510U dual-core 2.0 GHz – Hyper Threading – 64bit; 16GB de memória RAM. As execuções foram realizadas sobre o sistema operacional Linux Mint 18.1 Cinnamon 64-bit, kernel 4.4.0-63. Durante a realização dos experimentos, apenas o servidor Neo4J e o protótipo desenvolvido eram as únicas aplicações em execução, além dos processos normais da máquina. É digno de nota que o principal objetivo por trás da avaliação é mensurar o custo associado ao provimento de integridade em um banco de dados de grafo. À vista disso, a fim de nortear a condução dos experimentos e verificação dos resultados, algumas perguntas relevantes acerca da avaliação foram elencadas, são elas:

1) Qual o consumo adicional de recursos, em termos de tempo de processamento e espaço ocupado, para aplicar o método proposto em um SGBD de grafos?
2) Qual o impacto do algoritmo criptográfico adotado para prover integridade no método proposto?
3) Qual a influência do tamanho do banco de dados original, em termos de tamanho e número de nodos, na aplicação da solução?

Neste sentido, tendo em conta as perguntas de avaliação ora elencadas, foram definidas as variações necessárias para a execução dos experimentos. No caso, foram utilizados grafos com tamanhos variáveis entre dez (mínimo) e cem mil nodos (máximo), sendo que o tamanho dos grafos foi multiplicado por dez a cada incremento de tamanho. Os algoritmos criptográficos empregados nos experimentos foram o MD5 [23], o SHA1 [24] e o SHA256 [25]. Isto posto, a partir das perguntas de investigações, bem como das variações definidas, algumas hipóteses foram criadas, a fim de guiar a realização dos experimentos. Logo, é importante salientar que as avaliações

realizadas sobre o protótipo, foram voltadas a responder as hipóteses levantadas – apresentadas e explanadas no que segue.

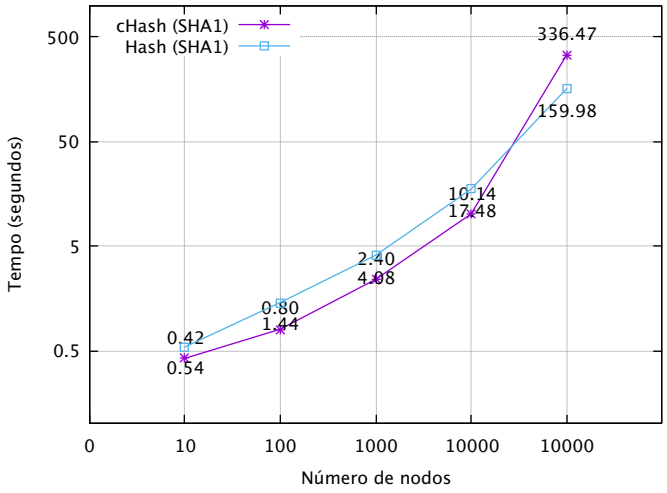


Figura 5. Latência de execução com diferentes números de nodos.

A primeira hipótese, doravante referenciada por **H1**, afirma de que *o tempo de aplicação do método proposto em um banco de dados (BD) é linearmente proporcional ao número de nodos armazenados no BD*. Os resultados para esta hipótese são reportados na Figura 5. Embora esta hipótese seja intuitiva (no sentido de que ambos os valores crescem, i.e., são diretamente proporcionais), os experimentos realizados, no entanto, mostraram que o tempo de aplicação do método proposto sobre um grafo por inteiro foi exponencial. Apesar desse resultado, pode-se afirmar que o tempo de aplicação é aceitável, considerando o número de nodos e que a atualização dos valores de *hash* e *cHash* do grafo por inteiro ocorre apenas na primeira execução do método. As demais atualizações desses valores ocorrem apenas em nodos que sofrem modificações e seus respectivos vizinhos.

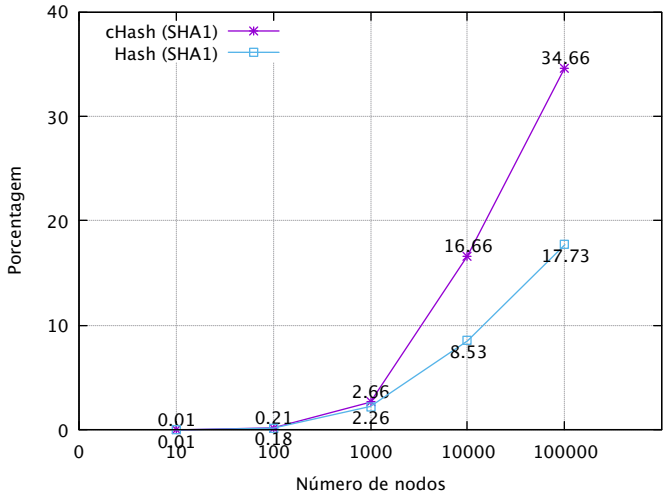


Figura 6. Espaço adicional após a aplicação do método proposto.

Cabe salientar que embora a execução tenha ocorrido em tempo não linear, a soma dos tempos de geração dos valores de *hash* e *cHash* ainda são aceitáveis, considerando-se o tamanho do grafo e que a atualização do banco inteiro ocorrerá apenas uma vez. Logo, é possível afirmar que a aceitabilidade é mensurável de acordo com o tamanho do grafo. Note que este mesmo comportamento se repetiu independente do algoritmo criptográfico aplicado. Isto posto, é possível concluir que a hipótese é falsa, já que o tempo de aplicação do método proposto não é linearmente proporcional ao número de nodos.

Como segunda hipótese – denotada por **H2** –, presume-se que *após a aplicação do método em um BD o tamanho adicional resultante da geração dos resumos criptográficos é menor do que 5%*. Esse valor usado como referência foi arbitrado como um valor pequeno, uma vez que há razoabilidade em estimar que não apenas o tamanho dos BDs, mas dos nodos, de um modo em geral, seja superior ao tamanho dos resumos criptográficos gerados. Por exemplo, resumos criptográficos gerados pelos algoritmos mais populares, tais como o MD5, SHA1 e SHA256, possuem tamanhos de 128 bits, 160 bits e 256 bits, respectivamente. À vista disso, a Figura 6 expõe o resultado do tamanho adicional após a geração dos valores de *hash* e *cHash*, em grafos de diferentes tamanhos, à luz da hipótese **H2**. Neste sentido, a partir dos valores reportados na Figura 6 observa-se que o espaço adicional do grafo foi maior que 5%, para grafos com mais de 10.000 nodos. Logo, a hipótese pode ser refutada, já que os resultados conduzem a conclusão de que a mesma é falsa. Tal afirmação decorre do fato de que o tamanho adicional gerado pela aplicação do *hash*/*cHash* cresce conforme o número de nodos do grafo. Por conseguinte, não é possível afirmar que o custo de armazenamento para o método proposto manterá a proporção de até 5% em relação tamanho total do BD, em todos os casos, mas apenas naqueles em que o tamanho do BD é módico.

A terceira hipótese **H3** trata do impacto do algoritmo criptográfico utilizado junto ao método proposto, no que tange o tempo de processamento consumido pelas operações de manipulação do grafo. Especificamente, a hipótese **H3** afirma que *o algoritmo criptográfico utilizado influencia significativamente no tempo requerido para aplicar o método proposto em um BD*. Para verificar a indagação exarada na hipótese em questão, também foi realizado um experimento, cujo resultado é reportado na Figura 7. Nesta figura é exibida uma comparação entre os tempos consumidos pela aplicação do método proposto, com a variação/alternância do algoritmo criptográfico. No caso, se observou que a variação/alternância do algoritmo criptográfico não causa influência significativa no tempo de aplicação da solução. Logo, pode-se inferir que a escolha do algoritmo criptográfico para realizar o cálculo do *hash* e *cHash* pouco interfere no tempo requerido para a implantação do método proposto. Outro aspecto que corrobora para com tal afirmação, é que o tempo consumido cresce quase que linearmente proporcional ao número de nodos existentes no BD. Uma análise dos resultados sob outra perspectiva evidencia que a alternância do algoritmo criptográfico incide em uma pequena variação de latência, sobre um mesmo grafo – i.e., com o mesmo número de nodos. A diferença de valores é

pouco significativa, o que, portanto, falseia a hipótese.

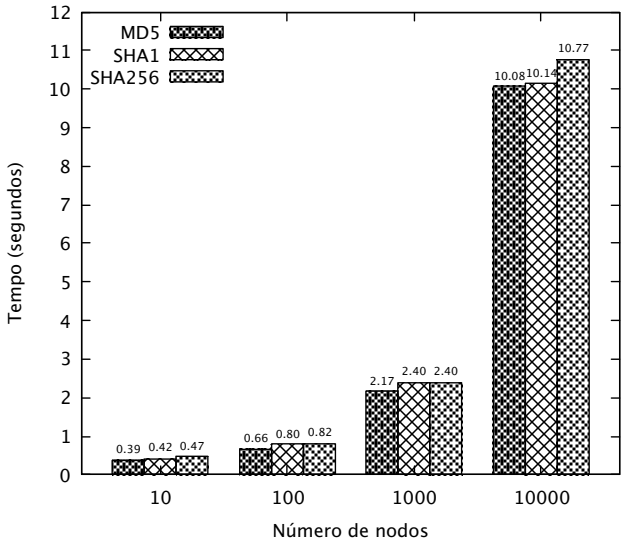


Figura 7. Latência de execução em função do algoritmo criptográfico.

A quarta hipótese **H4** discorre sobre a incidência do algoritmo criptográfico empregado no método proposto, sobre o espaço ocupado no BD. Diante disso, essa hipótese presume que *o algoritmo criptográfico utilizado influencia significativamente no espaço ocupado pelo BD que utiliza o método proposto*. Note que ambas as hipóteses **H3** e **H4** originaram-se do fato de que os resumos criptográficos gerados/obtidos possuem tamanhos diretamente relacionados aos algoritmos criptográficos utilizados. Na Figura 8 tem-se o resultado dos experimentos realizados à luz da hipótese **H4**. Nessa figura é exibida uma comparação dos valores, em percentuais – exibidos no topo das barras –, quanto ao espaço adicional gerado por cada um dos algoritmos criptográficos empregados naquele experimento. Neste cenário, é possível observar claramente uma variação dos valores, de acordo com o algoritmo criptográfico utilizado. Ademais, a diferença se torna mais evidente e significativa quando o número de nodos do grafo é superior à 10.000. Isto posto, pode-se concluir que a hipótese em questão é verdadeira.

No que segue, a quinta hipótese **H5** aborda um aspecto pertinente ao tamanho do grafo que forma o banco de dados – em termos de nodos e relacionamentos – e sua relação com o tempo de aplicação do método proposto. À vista disso, ela afirma que *nodos de maior tamanho incorrem em maior tempo de aplicação do método proposto, no BD*. Para verificar esta hipótese, comparou-se dados gerados pelo grafo com o mesmo número de nodos, porém, com dez vezes mais informações nos nodos. O gráfico da Figura 9 ilustra a comparação do tempo consumido pela aplicação do método proposto, em grafos do mesmo tamanho. No caso, o grafo (B) possui nodos com tamanho dez vezes maior que os nodos do grafo (A). Os experimentos evidenciaram claramente que o tamanho do nodo influencia no tempo de aplicação da solução, isto é, quanto maior a quantidade de informação no nodo, maior o

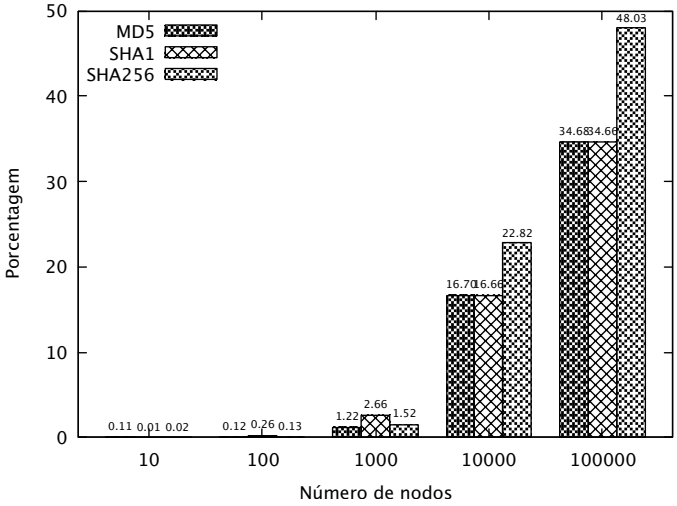


Figura 8. Custo adicional de armazenamento em escala exponencial.

seu tamanho e maior o tempo requerido para geração do *hash* e *cHash*. Isto decorre da necessidade de concatenação dos dados de todos os atributos do nodo, para a geração do resumo criptográfico daquele nodo. Deste modo, o tempo necessário para agrupar/concatenar toda(s) a(s) informação(ões) requerida(s) para produzir o resumo criptográfico, cresce de maneira proporcional ao número de atributos especificados para o nodo. Com isso, é demonstrado que a hipótese **H5** é verdadeira.

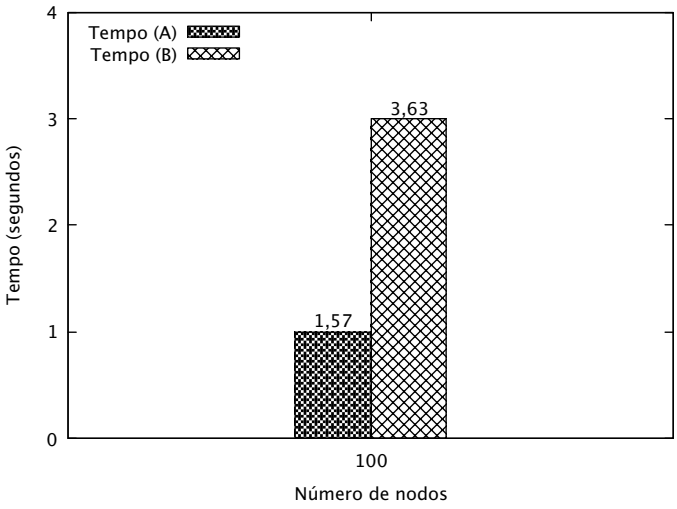


Figura 9. Latência de execução em um BD com nodos de tamanho 10 vezes maiores.

Por fim, a hipótese **H6** também estabelece uma relação entre o tamanho do grafo com o tempo de aplicação do método proposto. De forma similar a hipótese **H5**, a hipótese **H6** antevê que *quanto mais nodos existirem no grafo, maior será o tempo de aplicação do método proposto, no BD*. Essas duas hipóteses são coerentes e intuitivas, logo, seus objetivos consistem em reafirmar estas características no método proposto. Os resultados para os experimentos pertinentes a esta hipótese são reportados na Figura 10. De forma análoga ao que ocorre com o resultado do experimento para atestar a hipótese **H5**,

a Figura 10 mostra como o aumento do número de nodos, e consequente aumento do número de relacionamentos/arestas, faz com que a proporção do tempo consumido pela geração do *cHash* também se eleve. Porém, no caso da hipótese **H5** a proporção é verificada em função do número de atributos, enquanto que na hipótese **H6** ela é verificada em relação ao número de vértices/nodos e suas relações existentes, já que eles serão empregados para a geração da cadeia de resumos criptográficos.

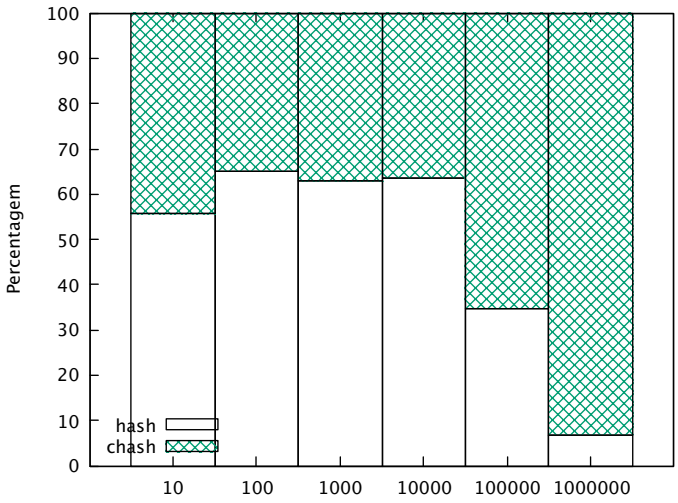


Figura 10. Dispêndio de tempo em termos da latência de execução do hash e cHash.

No caso particular dos resultados verificados para hipótese **H6**, a partir da Figura 10, o impacto sobre a latência decorre da necessidade de efetuar, para cada nodo, uma consulta a todos os seus vizinhos, a fim de obter todas as informações requeridas para produzir a cadeia de resumos criptográficos. O que termina por resultar em um aumento do tempo. Logo, conclui-se que a hipótese é verdadeira.

VII. Conclusões

Mediante a crescente possibilidade de corrupção de dados oriunda de ataques cibernéticos e as nuvens de radiação que emanam nos datacenters, este trabalho apresentou um método para prover integridade às informações armazenadas em bancos de dados de grafos. Resumos criptográficos asseguram as informações dos nodos, e a inclusão de elementos nos nodos expande gradualmente a integridade ao restante do grafo. Alterações em nodos do grafo implicam na atualização dos resumos de maneira localizada e reduzida. As avaliações mostraram que o consumo de recursos necessários ao provimento da integridade é viável para BDs de tamanho módico. Embora os experimentos tenham considerado apenas o SGBD Neo4J, SGBDs como OrientDB, AllegroGraph, Titan, Trinity, e outros, são passíveis de incorporação dos algoritmos especificados para o método proposto, sem o dispêndio de demasiado esforço.

Referências

[1] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The end of an architectural era: (it's time for a complete rewrite)," in **Proceedings of the 33rd International Conference on Very Large Data Bases**, 2007, pp. 1150–1160.

[2] J. M. Hellerstein, M. Stonebraker, and J. Hamilton, "Architecture of a Database System," **Foundations and Trends in Databases**, vol. 1, no. 2, pp. 141–259, 2007.

[3] R. Cattell, "Scalable SQL and NoSQL data stores," **SIGMOD Record**, vol. 39, no. 4, pp. 12–27, 2011.

[4] I. Robinson, J. Webber, and E. Eifrem, **Graph Databases: New Opportunities for Connected Data**, 2nd ed. O'Reilly Media, Inc., 2015.

[5] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," **ACM Transactions on Programming Languages and Systems**, vol. 4, no. 3, pp. 382–401, 1982.

[6] A. F. Luiz, L. C. Lung, and M. Correia, "Mitra: Byzantine fault-tolerant middleware for transaction processing on replicated databases," **SIGMOD Record**, vol. 43, no. 1, pp. 32–38, 2014.

[7] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: A large-scale field study," in **Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems**, 2009, pp. 193–204.

[8] E. B. Nightingale, J. R. Douceur, and V. Orgovan, "Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer PCs," in **Proceedings of the 6th European Conference on Computer Systems**, 2011, pp. 343–356.

[9] M. A. Rabuske, **Introdução à Teoria dos Grafos**. Florianópolis, SC, Brasil: Editora da UFSC, 1992.

[10] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," **IEEE Transactions on Dependable and Secure Computing**, vol. 1, no. 1, pp. 11–33, 2004.

[11] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in **Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology**, 1996, pp. 1–15.

[12] W. Mao, **Modern Cryptography: Theory and Practice**. Pearson Education, 2003.

[13] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, **Handbook of Applied Cryptography**. CRC Press, Inc., 1996.

[14] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in **Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data**, 2010, pp. 135–146.

[15] D. Presser, L. C. Lung, and M. P. Correia, "Tolerância a faltas arbitrária em processamento distribuído de grafos," in **Anais do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2015, pp. 1–14.

[16] R. Neiheiser, L. C. Lung, A. F. Luiz, and H. V. Netto, "Replicação tolerante a faltas eficiente em bancos de dados orientados a grafos," in **Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2016, pp. 1–14.

[17] R. C. Merkle, "A certified digital signature," in **Proceedings on Advances in Cryptology (CRYPTO' 89)**, 1989, pp. 218–238.

[18] M. U. Arshad, A. Kundu, E. Bertino, K. Madhavan, and A. Ghafoor, "Security of graph data: Hashing schemes and definitions," in **Proceedings of the 4th ACM Conference on Data and Application Security and Privacy**, 2014, pp. 223–234.

[19] M. U. Arshad, A. Kundu, E. Bertino, A. Ghafoor, and C. Kundu, "Efficient and scalable integrity verification of data and query results for graph databases," **IEEE Transactions on Knowledge and Data Engineering**, vol. 30, no. 5, pp. 866–879, 2018.

[20] A. L. Silvério, R. F. Custódio, M. C. Carlos, and R. dos Santos Mello, "Efficient data integrity checking for untrusted database systems," in **Proceedings of the 6th International Conference on Advances in Databases, Knowledge, and Data Applications**, 2014, pp. 118–124.

[21] L. Lamport, "Password authentication with insecure communication," **Communications of the ACM**, vol. 24, no. 11, pp. 770–772, 1981.

[22] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[23] R. L. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, 1992.

[24] D. Eastlake 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, 2001.

[25] National Institute of Standards and Technology (NIST), "Secure hash standard," FIPS Publication 180-4, 2015.