Exploração de Grafos RDF com Distribuição Controlada

Raqueline R. M. Penteado[1,3], Rebeca Schroeder[2], Carmem S. Hara[1]

[1]Universidade Federal do Paraná (UFPR) – DInf – Curitiba, PR – Brasil

[2]Universidade do Estado de Santa Catarina (UDESC) – DCC – Joinville, SC – Brasil

[3]Universidade Estadual de Maringá (UEM) – DIN – Maringá, PR – Brasil

rrmpenteado@inf.ufpr.br, rebeca.schroeder@udesc.br, carmem@inf.ufpr.br

Abstract. *The communication costs involved in retrieving distributed data in SPARQL queries have a big impact on the system performance. In this paper, we define a parallel graph processing model that explores the existence of* allocation patterns*, which consist of information on how data has been distributed among servers. Based on this model, we define two types of communication schedules:* get-frag *and* send-result*. These strategies are of great interest to query optimizers for efficient query processing on distributed RDF stores.*

Resumo. *Grande parte do custo envolvido no processamento distribuído de consultas SPARQL resulta do custo de comunicação para a obtenção dos dados envolvidos na consulta. Neste trabalho é definido um modelo de exploração de grafos paralelo para consultas SPARQL que considera a existência de padrões de distribuição de dados. A partir deste modelo, são definidos dois modelos de escalonamento de comunicação entre servidores:* get-frag *e* send-result*. Estes modelos poderão ser explorados futuramente por um otimizador para a execução eficiente de consultas sobre bases RDF distribuídas.*

1. Introdução

O padrão RDF adota um modelo conceitual de grafos para representar dados e expressar relacionamentos entre entidades. A simplicidade e a flexibilidade do modelo motivou a proliferação de grandes bases de dados RDF como, por exemplo, a DBPedia, que representa de forma estruturada o conteúdo da Wikipedia. Segundo o consórcio *W3C*, algumas bases comerciais já alcançaram a marca de 1 trilhão de triplas RDF[1]. Este volume de dados e sua tendência de crescimento representam um grande desafio para a escalabilidade dos sistemas de gerenciamento de dados RDF atuais. Em resposta à explosão de dados RDF, pesquisadores têm investido esforços no desenvolvimento de técnicas escaláveis para o processamento de consultas SPARQL (*SPARQL Protocol and RDF Query Language*)[2].

Considerando o armazenamento de grandes bases de dados, sistemas tais como a proposta de Huang et al (2011) e *Triad* [Gurajada et al. 2014] armazenam seus dados de forma distribuída com o objetivo de prover uma maior escalabilidade às suas aplicações quando comparados com sistemas que adotam abordagens centralizadas, como o sistema *RDF-3X* [Neumann and Weikum 2010]. Porém, a distribuição de dados implica em custo de comunicação no processamento de consultas, uma vez que os dados envolvidos podem

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

[2]http://www.w3.org/TR/rdf-sparql-query/

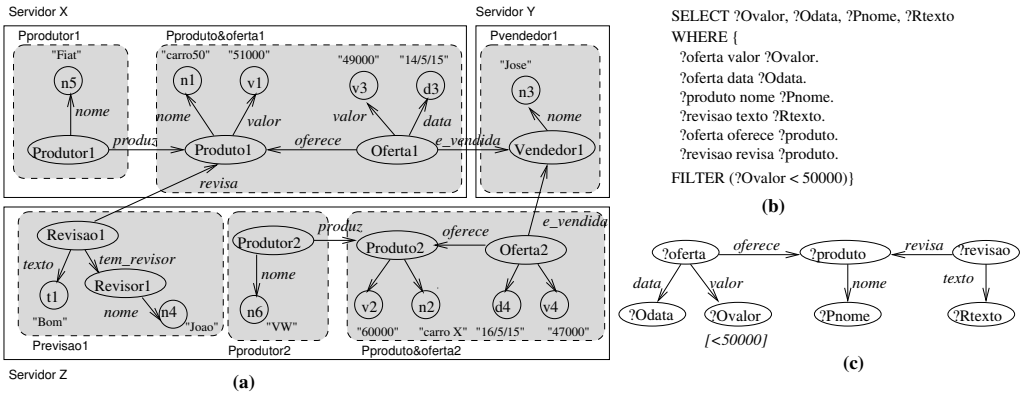

Figura 1. (a) Grafo RDF; (b) Consulta SPARQL; (c) Grafo de consulta

estar alocados em diversos servidores. Sendo assim, diferentes métodos de distribuição de dados têm sido propostos com o objetivo de minimizar este custo. Uma alternativa são os métodos de distribuição classificados como "controlados", uma vez que eles fragmentam e distribuem dados por meio de padrões de agrupamento que podem ser usados pelo processador de consultas. Na Figura 1(a), os fragmentos distribuídos estão representados por retângulos tracejados e cada fragmento de dados possui o seu padrão. Por exemplo, os fragmentos *Pproduto&oferta1* e *Pproduto&oferta2* possuem o padrão *Pproduto&oferta* que determina que um produto sempre estará armazenado fisicamente com suas ofertas.

Considerando um método de distribuição controlado e a ocorrência de trocas de mensagens entre servidores de um *cluster* durante a execução de consultas, duas estratégias comunicação entre servidores podem ser identificadas. A primeira baseia-se no envio de resultados intermediários (*send_result*) e a segunda na requisição de fragmentos de dados necessários (*get_frag*) para o processamento de consultas. Considere o grafo fragmentado da Figura 1(a) e a recuperação de dados envolvida no processamento da consulta da Figura 1(b). O servidor `X` é capaz de processar grande parte da consulta, pois possui os dados referentes a `produto` e `oferta`. Contudo, para obter os dados de `revisao`, é preciso acessar o servidor `Z`. No modelo *send_result*, o servidor `X` envia a `Z` o resultado intermediário do processamento (`oferta` e `produto`) para que `Z` finalize a execução da consulta. Já, no modelo *get_frag*, o fragmento de dados *Previsao1* é requisitado por `X`, enviado de `Z` para `X`, que continua a execução da consulta localmente.

O objetivo deste artigo é apresentar um modelo de exploração de grafos RDF para o processamento distribuído de consultas SPARQL, que considera estas duas estratégias. Este modelo difere da maioria dos trabalhos relacionados, que são baseados em um único modelo de comunicação. Assim, futuramente, otimizadores de consulta poderão explorar as estratégias propostas visando um processamento eficiente de consultas. Este artigo está organizado em mais 5 seções. A Seção 2 apresenta conceitos preliminares. A Seção 3 descreve o modelo de processamento de consultas e os dois modelos de escalonamento. A Seção 4 apresenta resultados experimentais iniciais, seguida pela Seção 5, que discute os trabalhos relacionados. Por fim, as conclusões são apresentadas na Seção 6.

2. Definições Preliminares

A sintaxe do modelo RDF é baseada no conceito de triplas da forma *(sujeito, predicado, objeto)*. Uma tripla pode ser interpretada como uma entidade *(sujeito)* que tem uma pro-

priedade ou um relacionamento *(predicado)* com um determinado valor literal ou entidade *(objeto)*. Uma base RDF pode ser representada por um grafo $G = (V, A)$, onde (1) V é o conjunto de vértices que representam os sujeitos e os objetos da base, e (2) A é o conjunto de arestas direcionadas e rotuladas que representam os predicados que relacionam pares de vértices. Fisicamente, tais arestas podem ser representadas na forma de listas de adjacências do vértice. Sendo assim, cada vértice v que representa uma entidade no grafo RDF pode ser representado pela tripla *(id,in,out)*, na qual: (1) *id(v)* denota o identificador da entidade, (2) *in(v)* é o conjunto de arestas incidentes em v, e (3) *out(v)* é o conjunto de arestas com origem em v. Por exemplo, a entidade `Produto1` da Figura 1(a) pode ser representada por $v_{Produto1}$ = (*id: http://exemplo* , *in:*{(*produz, Produtor1*), (*oferece, Oferta1*), (*revisa, Revisao1*)}, *out:*{ (*valor,* v_1), (*nome,* n_1)}).

O núcleo da linguagem SPARQL baseia-se em um conjunto de padrões de triplas semelhantes às triplas RDF, exceto pelo fato das triplas admitirem variáveis em seus membros. Existe uma variedade de operadores na linguagem [Pérez et al. 2009], mas neste trabalho são considerados os operadores *select*, *join* e *filter*. Dada uma consulta SPARQL, um padrão de grafo básico (PGB) é construído e pesquisado no grafo RDF. A contrução é feita a partir das junções definidas pelas variáveis em comum nos padrões de triplas, denominadas de *variáveis de junção*. A Figura 1(c) ilustra o PGB correspondente à consulta SPARQL da Figura 1(b), que obtém as entidades do tipo *Oferta* com seus respectivos *Produtos* e *Revisões*. Neste exemplo, são variáveis de junção: *?oferta, ?produto* e *?revisao*. O resultado da consulta corresponde à projeção dos mapeamentos sobre as variáveis de PGB. Um exemplo de mapeamento resultante do processamento da consulta da Figura 1(c) sobre o grafo da Figura 1(a) é: (*?oferta* $\mapsto$ *Oferta1*, *?produto* $\mapsto$ *Produto1*, *?revisao* $\mapsto$ *Revisao1*, *?Ovalor* $\mapsto$ *v3*, *?Odata* $\mapsto$ *d3*, *?Pnome* $\mapsto$ *n1*, *?Rtexto* $\mapsto$ *t1*).

Este trabalho considera grafos de consulta conexos. Logo, a geração de mapeamentos pode ser processada pelo método de exploração de grafos. Algoritmos de isomorfismo de subgrafos podem ser utilizados na exploração. Ullmann (1976) propõe um algoritmo centralizado clássico desta categoria. A exploração requer a definição de uma ordem de caminhamento no grafo para a pesquisa de subgrafos isomórficos a PGB. A ordem, em nosso modelo, corresponde à definição de uma sequência das arestas do PGB.

Schroeder (2014) propõe a técnica *ClusterRDF* para a distribuição controlada de dados em um *cluster* de servidores. Esta técnica introduz os conceitos de *padrões de alocação (PA)* e *templates*. Os PAs definem quais vértices de um grafo RDF devem ser alocados em um mesmo servidor. Cada PA é dividido em um conjunto de *templates*, que definem unidades de comunicação. Os *templates* tem como objetivo minimizar a quantidade de comunicação entre servidores, empacotando dados que são usualmente utilizados em conjunto em uma mesma transmissão. Considerando o grafo RDF da Figura 1(a), a alocação dos vértices nos servidores `X`, `Y` e `Z` é o resultado da definição de três PAs: PA_1 : {`produto`, `oferta`, `produtor`}, PA_2:{`revisao`, `revisor`} e PA_3:{`vendedor`}. No exemplo, o PA_1 é dividido em dois *templates* (representados por linhas tracejadas): `Produto&oferta` e `Produtor`. Instâncias de *templates* são denominados de *fragmentos*. Uma aresta que relaciona dois fragmentos é replicada em ambos os fragmentos e denominada de aresta de corte.

3. Processamento distribuído de consultas

O objetivo desta seção é apresentar o modelo de processamento distribuído e paralelo e os dois modelos de comunicação considerados neste artigo. Basicamente, o processamento de consultas envolve as etapas de análise, planejamento e execução.

Na etapa de **análise**, dada um consulta SPARQL, PAs e *templates* são detectados a partir das arestas do PGB da consulta. Por exemplo, para o grafo de consulta da Figura 1(c), detecta-se que os padrões de triplas pertencem ao *template Produto&oferta* e PA_1 e os padrões restantes ao *template Revisão* e PA_2.

Durante o **planejamento**, define-se uma sequência de exploração que agrupe os padrões de tripla de um mesmo *template* e/ou PA para que a quantidade de comunicação seja minimizada. Observe que a ordenação de exploração em PAs distintos, bem como de padrões dentro de um mesmo *template* ou PA tem impacto sobre o tempo de processamento da consulta. Contudo, a geração de uma ordenação adequada depende de informações de cardinalidade e seletividade e está fora do escopo deste trabalho. Um plano de execução Q é uma sequência de passos $[q_1, \ldots, q_n]$, onde q_i é uma quintúpla (a, t, p, f, dir), onde (1) a é um padrão de tripla, (2) t é uma *template* que contém a, (3) p é um PA que contém a, (4) f é o conjunto de filtros definidos sobre variáveis de a, e (5) $dir \in \{in, out\}$ tal que se $dir(q_i) = out$ o percurso no grafo RDF é na direção $(sujeito, predicado, objeto)$ em a; caso contrário, a direção é $(objeto, predicado, sujeito)$ em a. Dando continuidade ao exemplo acima, um possível plano para a consulta da Figura 1(b) consiste dos seguintes passos:

[q_1 : ((?oferta, valor, ?Ovalor), Produto&oferta, PA_1, {(?Ovalor < 50000)}, out),
q_2 : ((?oferta, data, ?Odata), Produto&oferta, PA_1, {}, out),
q_3 : ((?oferta, oferece, ?produto), Produto&oferta, PA_1, {}, out),
q_4 : ((?produto, nome, ?Pnome), Produto&oferta, PA_1, {}, out),
q_5 : ((?revisao, revisa, ?produto), Revisao, PA_1, {}, in),
q_6 : ((?revisao, texto, ?Rtexto), Revisao, PA_2, {}, out)]

Para a **execução** de um plano Q sobre um grafo RDF $G_b = (V_b, A_b)$, o sujeito de q_1 é mapeado em V_b de acordo com as restrições de q_1 para definir um ou mais pontos iniciais de exploração. De acordo com dir e f, triplas RDF são instanciadas gerando o conjunto de mapeamento de q_1. Na sequência, a entrada para um passo q_i consiste de um conjunto de mapeamentos das variáveis de junção dos passos anteriores e o resultado é um conjunto de mapeamentos de triplas RDF instanciadas em $[q_1, \ldots, q_{i-1}] \cup q_i$. Assim, para a execução de Q no grafo RDF da Figura 1(a), os pontos iniciais de exploração começam com os mapeamentos $\{(?oferta \mapsto oferta1), (?oferta \mapsto oferta2)\}$. Após a execução de q_1 o resultado é $\{(?oferta \mapsto oferta1, ?Ovalor \mapsto v_3), (?oferta \mapsto oferta2, ?Ovalor \mapsto v_4)\}$. O passo q_2 busca triplas de acordo com sua variável de junção e os mapeamento de $?oferta$ em q_1, e assim sucessivamente. Quando o $pa(q_i) \neq pa(q_{i-1})$ e as instâncias dos PAs estão em servidores diferentes, a comunicação pode ser feita utilizando dois modelos de comunicação, sendo eles: *1)* ***send_result***: neste modelo, os mapeamentos gerados até q_{i-1} são enviados aos servidores capazes de instanciarem a variável de junção de q_i dando continuidade à exploração; e, *2)* ***get_frag***: neste modelo, o servidor requisita os fragmentos capazes de instanciarem o vértice de junção de q_i para dar continuidade ao processamento da consulta localmente. Em Q, o passo q_5 representa uma aresta de corte que implica na comunicação para a execução do passo q_6 com a variável de junção $?revisao$. Uma função de custo deve ser elaborada para que o otimizador escolha

a estratégia de comunicação que implique em um menor tempo de processamento de Q.

4. Experimentos

Uma análise inicial do modelo de processamento foi feita a partir de um protótipo preliminar baseado em Ullmann (1976), implementado em Java e que utiliza os mecanismos de *thread* e RMI para viabilizar o processamento distribuído e paralelo. Foi adotada uma arquitetura mestre-escravo composta por 3 instâncias Amazon EC2 64-bit do tipo t1.micro e o repositório em memória Berkeley DB[3] para o armazenamento dos dados RDF e do modelo de distribuição. Na arquitetura, o servidor mestre recebe e analisa requisições de consulta, define planos de execução e requisita o processamento paralelo a todos os servidores escravos; cada escravo recebe paralelamente as requisições e inicializa a exploração com a sua base local. Quando um passo do plano não pode ser processado localmente, é utilizada uma estratégia de comunicação para dar continuidade à execução da consulta.

A base de dados foi gerada a partir do *Berlin SPARQL Benchmark*[4], que permite gerar bases para um domínio *e-commerce* que estabelece os relacionamentos entre produtos e seus produtores. No experimento realizado, foi utilizada uma base com 20 produtos que corresponde a um total de 2.203 triplas (524KB). Na base todos os produtos estavam relacionados com um único produtor e, conforme a distribuição gerada, os fragmentos relacionados aos produtos foram alocados no servidor denominado de escravo 2, enquanto que os fragmentos do produtor, no servidor escravo 1. Para a distribuição da base foi adotado o método de distribuição *ClusterRDF*. Devido à distribuição, as listas de adjacência dos vértices consideram o endereço físico dos vértices adjacentes que não estão armazenados em um mesmo PA. Além disso, tornaram-se necessários índices para a recuperação de: *(i)* uma instância de um tipo de PA dado um vértice v; *(ii)* fragmentos de um tipo de *template* dada uma instância de um PA; e *(iii)* vértices v dado um fragmento.

Uma consulta foi analisada a partir de dois planos diferentes. O objetivo da consulta era o de recuperar produtos e seus respectivos produtores. No primeiro plano (P1) obteve-se `produtos` e na sequência `produtor`. No segundo (P2), a ordem dos padrões de triplas foi inversa a P1. Em ambos, o tamanho do resultado da consulta é o mesmo, isto é, 1.116 bytes. Três variáveis foram analisadas: número de mensagens trocado entre servidores (M), volume de bytes trafegado na rede (B) e tempo total de resposta (T). Em P1, com *get-frag* obteve-se: M = 1 (recuperação do fragmento do `produtor` pelo escravo 2), B = 6.277 bytes (tamanho do fragmento recuperado) e T = 0,417 s. Com *send-result* obteve-se: M = 20 (envio dos resultados intermediários do escravo 2 ao escravo 1), B = 3.384 bytes (volume total dos resultados intermediários) e T = 0,548 s. Pode-se notar que nesse contexto o valor de M influenciou diretamente em T. Em P2, *get-frag* exige a troca de mais mensagens e trafega um maior volume de bytes que *send-result*. Sendo assim, o tempo de *get-frag* foi de 0,276 s e o de *send-result*, 0,179 s. Esta breve análise mostrou que dependendo do plano de execução de uma mesma consulta, uma estratégia de comunicação torna-se mais adequada que outra.

5. Trabalhos Relacionados

O modelo de processamento de consultas e a estratégia de comunicação entre servidores influenciam no custo de comunicação. Para o processamento, sistemas optam por uma

[3]http://www.oracle.com/technetwork/database/database-technologies/berkeleydb

[4]http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

abordagem centralizada ou distribuída. Na primeira, uma consulta é alocada a um único servidor de processamento, que requisita os dados necessários aos servidores de armazenamento. Punnoose et al (2012) faz uso desta abordagem em *Rya*. Na distribuída, que é adotada neste trabalho, a carga de processamento é distribuída entre diversos servidores. Diversos sistemas tais como as propostas de Huang et al (2011), Yang et al (2013) e Gurajada et al (2014) adotam esta abordagem. Já, *Trinity.RDF* [Zeng et al. 2013] adota uma abordagem mista onde um servidor central finaliza o processamento da consulta. Apesar destes sistemas adotarem métodos de distribuição específicos para minimizar o custo de comunicação, a maioria deles adotam somente uma estratégia de comunicação. O modelo de processamento apresentado neste artigo propõe o uso de duas estratégias de *comunicação*. Essa flexibilidade visa possibilitar a geração de planos de consulta que adotem o modelo considerado mais adequado para um determinado contexto. Tal flexibilidade não foi encontrada em nenhum outro trabalho na literatura.

6. Conclusão

Este artigo propõe um modelo de exploração de grafos que explora a padronização de agrupamento de dados gerada por métodos de distribuição de dados classificados como "controlados". A partir deste modelo, foram definidas duas estratégias de comunicação entre servidores que podem ser usadas durante o processamento distribuído de consultas. Uma análise detalhada sobre um conjunto de consultas SPARQL encontra-se em andamento para uma comparação completa dos modelos. O objetivo da análise é o de fornecer heurísticas para a escolha do melhor modelo de comunicação durante a exploração distribuída de grafos. Este trabalho traz contribuições para o contexto de bancos de dados de grande escala, uma vez que os modelos aqui apresentados poderão ser explorados por otimizadores de consultas para prover escalabilidade ao processamento de consultas sobre fontes de dados RDF em crescente expansão.

Agradecimentos Este trabalho foi parcialmente financiado pela Capes e pelo *AWS* na Educação.

Referências

Gurajada, S., Seufert, S., Miliaraki, I., and Theobald, M. (2014). TriAD: A Distributed Shared-nothing RDF Engine Based on Asynchronous Message Passing. In *ACM SIGMOD*, pages 289–300.

Huang, J., Abadi, D. J., and Ren, K. (2011). Scalable SPARQL Querying of Large RDF Graphs. *PVLDB*, 4(11):1123–1134.

Neumann, T. and Weikum, G. (2010). The RDF-3X Engine for Scalable Management of RDF Data. *The VLDB Journal*, 19(1):91–113.

Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and Complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45.

Punnoose, R., Crainiceanu, A., and Rapp, D. (2012). Rya: A scalable rdf triple store for the clouds. In *Proceedings of the 1st International Workshop on Cloud Intelligence*, Cloud-I '12, pages 4:1–4:8, New York, NY, USA. ACM.

Schroeder, R. (2014). *Uma Abordagem para o Particionamento de Dados na Nuvem Baseada em Relações de Afinidade em Grafos*. PhD thesis, Universidade Federal do Paraná.

Ullmann, J. R. (1976). An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42.

Yang, T., Chen, J., Wang, X., Chen, Y., and Du, X. (2013). Efficient SPARQL Query Evaluation via Automatic Data Partitioning. In *DASFAA (2)*, pages 244–258. Springer.

Zeng, K., Yang, J., Wang, H., Shao, B., and Wang, Z. (2013). A Distributed Graph Engine for Web Scale RDF Data. In *VLDB*, PVLDB'13, pages 265–276. VLDB Endowment.