REBECA SCHROEDER FREITAS

UMA ABORDAGEM PARA O PARTICIONAMENTO DE DADOS NA NUVEM BASEADA EM RELAÇÕES DE AFINIDADE EM GRAFOS

CURITIBA

2014

REBECA SCHROEDER FREITAS

UMA ABORDAGEM PARA O PARTICIONAMENTO DE DADOS NA NUVEM BASEADA EM RELAÇÕES DE AFINIDADE EM GRAFOS

Tese apresentada como requisito parcial à obtenção do título de Doutor em Ciência da Computação, no Programa de Pós-Graduação em Informática do Setor de Ciências Exatas da Universidade Federal do Paraná.
Orientadora: Prof. Dra. Carmem Satie Hara

CURITIBA
2014

RESUMO

Atualmente, a Web não é meramente uma plataforma para publicar dados, mas um meio para conectar e disponibilizar o conhecimento acumulado no mundo. A DBpedia, por exemplo, é capaz de modelar, representar e compartilhar informações provenientes da massa de dados compartilhada pela Wikipedia. Sobretudo, observa-se que uma quantidade cada vez maior de conjuntos de dados vem sendo disponibilizada. A versão 3.9 da DBpedia[1], por exemplo, atingiu um tamanho de 2.460 milhões de triplas RDF extraídas da Wikipedia. De acordo com o repositório *Large Triple Stores*[2], alguns *datasets* comerciais podem ser ainda maiores alcançando a casa de trilhões de triplas. Para suportar esta quantidade massiva de dados, sistemas gerenciadores de banco de dados necessitam particionar e distribuir estes grandes conjuntos de dados em servidores de um sistema distribuído. No projeto físico de banco de dados, uma distribuição adequada dos dados visa prover escalabilidade e disponibilidade das aplicações clientes. O advento da computação na *nuvem* e a disseminação de sistemas distribuídos altamente escaláveis têm mostrado que as soluções tradicionais para a distribuição de dados não são adequadas. Nesta tese o problema da distribuição de dados é abordado para o contexto de bancos de dados em nuvens computacionais. O foco deste trabalho é desenvolver abordagens para a fragmentação e alocação de dados que minimizam a execução de consultas distribuídas e, consequentemente, melhoram a vazão do sistema. Uma carga de trabalho é analisada para agrupar dados fortemente relacionados em um mesmo fragmento ou em um conjunto de fragmentos armazenados em um mesmo servidor. A metodologia proposta está focada em modelos em grafo, especificamente os modelos RDF e XML, dada a flexibilidade destes modelos para dar suporte a diversas aplicações. Resultados experimentais mostram que a solução proposta é efetiva para melhorar o desempenho de consultas em repositórios de dados em nuvem se comparada a abordagens alternativas.

Palavras-chave: RDF, Partitionamento, Fragmentação, Agrupamento, Banco de Dados Distribuído

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

ABSTRACT

Nowadays, the Web is no longer merely a platform for publishing data, but a means to connect and deliver the cumulative knowledge of the world. DBpedia, for instance, is able to model, represent the massive amount of data from Wikipedia. Above all, we have witnessed a ever-increasing amount of data made available. DBpedia[3] release 3.9, for instante, has reached a size of 2.46 billion RDF triples extracted from Wikipedia. According to the *Large Triple Stores*[4], some commercial datasets may be even bigger reaching the score of trillion triples. In order to support this massive amount of data, data management systems require data to be partitioned and distributed across multiple servers. In physical design of distributed databases, a suitable data distribution aims to provide scalable and available database services. The advent of cloud computing and the dissemination of large-scale distributed systems have shown that the traditional approaches for data distribution are not suitable. In this thesis, we tackle the data distribution problem for cloud computing environment. Our focus is on developing fragmentation and allocation approaches that avoid distributed execution of queries and, consequently, improve the system throughput. To this purpose, we analyze a transactional workload in order to pack the most correlated data in the same fragment or in a set of few fragments stored in the same server. We focus on graph models, i.e. the RDF and XML models, given the flexibility of these models to support several applications. Experimental results show that the proposed solution is effective for improving query performance in cloud data stores compared to alternative approaches.

Keywords: RDF, Partitioning, Fragmentation, Allocation, Clustering, Distributed databases

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

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

A computação nas nuvens, ou *cloud computing*, é o termo comumente associado à tendência tecnológica de hospedar um número crescente de aplicações, serviços e dados em *datacenters* de larga escala [Agrawal et al., 2013]. Esta tendência está a promover transformações em diversos aspectos da computação através de novas formas de prover serviços. Infraestruturas, plataformas e softwares são oferecidos como *commodities* através de um modelo de custo sob-demanda que os disponibiliza em tempo real e prontos para o consumo. Em especial, um número crescente de aplicações está se beneficiando de plataformas nas nuvens. Entretanto, esta proliferação tem favorecido ao aumento do volume de dados sendo produzidos e consumidos por estas aplicações. Diversos desafios estão associados a este cenário conhecido como *Big Data*, dentre os quais promover a escalabilidade de sistemas de gerenciamento de dados constitui parte fundamental para infraestruturas em nuvem. A importância deste desafio foi reconhecida em 2012 pelo governo dos EUA ao anunciar a *Big Data Initiative*, que destinou $200 milhões por 5 anos para projetos de pesquisa envolvendo *Big Data*, dentre os quais $25 milhões foram destinados ao desenvolvimento de novos métodos para escalar o gerenciamento de dados do Departamento de Energia americano [Weiss and Zgorski, 2012].

O termo *Big Data* foi inicialmente definido por Douglas Laney [Laney, 2012] como aplicações de grande volume, alta velocidade e grande variedade de informações que requerem novas formas de processamento para permitir a tomada de decisão, a descoberta de conhecimento e otimização de processos. Esta definição é conhecida como modelo *3 Vs*, pois estabelece as 3 dimensões características do *Big Data*, isto é, Volume, Velocidade e Variedade. Embora o termo tenha sido definido em 2012, a necessidade de escalar estas 3 dimensões do gerenciamento de dados foi identificada em 2001 [Laney, 2001] impulsionada

pelos efeitos do *e-commerce* e da integração de diversas aplicações. Uma década depois, o efeito *Big Data* ganhou nome e espaço especialmente devido a evolução e maturação de tecnologias envolvidas com a computação nas nuvens. Na prática, *Big Data* representa as características e desafios de um conjunto expressivo de aplicações, enquanto a computação nas nuvens atua como a infraestrutura a ser utilizada e desenvolvida para responder aos diversos desafios impostos por estas aplicações.

Em relação ao suporte da *Variedade* de informações, destacam-se os modelos de dados XML(*eXtensible Markup Language*) e RDF(*Resource Description Framework*). O modelo XML se tornou o padrão para o intercâmbio de informações em aplicações Web [Angles and Gutierrez, 2008]. Sua ascensão está relacionada a aplicações de *e-commerce*, onde XML passou a ser utilizado como formato de integração de documentos e aplicações. Com advento da *web semântica*, conjuntos de dados provenientes de diferentes aplicações passaram a ser vinculados através de uma pilha de padrões abertos chamados de *Resource Description Framework* (RDF). Atualmente, RDF é o padrão estabelecido pelo projeto *Linked Open Data*[1] para publicação de conjuntos de dados na Web, e assim habilitar o acesso a diferentes fontes que juntas formam uma base global de informações.

Uma quantidade cada vez maior de *datasets* vem sendo disponibilizada em diferentes domínios de aplicação. A DBpedia[2], por exemplo, atingiu um tamanho de 2.460 milhões de triplas RDF extraídas da Wikipedia. De acordo com o repositório *Large Triple Stores*[3], alguns *datasets* comerciais podem ser ainda maiores alcançando a casa de trilhões de triplas. Haja vista a tendência de crescimento constante destas fontes de dados, reconhece-se a necessidade eminente de sistemas de gerenciamento de dados capazes de suportar esta demanda. Embora existam resultados significativos em escalar esses sistemas em uma única máquina [Franz Inc, 2013], não correspondem a soluções realistas, em particular quando a quantidade de dados continua a expandir. Uma forma de gerenciar estes *datasets* é através de uma arquitetura baseada em repositórios em nuvem [Kossmann et al., 2010],

[1] http://linkeddata.org/
[2] http://wiki.dbpedia.org/Datasets
[3] http://www.w3.org/wiki/LargeTripleStores

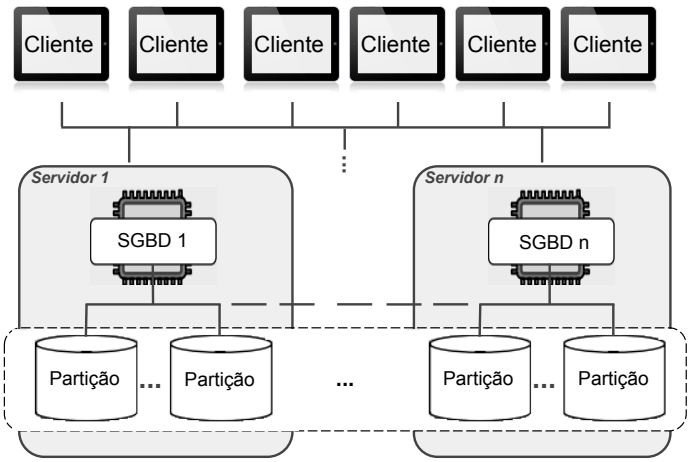


Figura 1.1: Arquitetura para o Gerenciamento de Dados Distribuídos

como a apresentada pela Figura 1. A elasticidade de nuvens computacionais prevê o ajuste da capacidade do sistema pela adição ou remoção de servidores. Desta forma, é dito que o sistema é capaz de escalar horizontalmente pela distribuição de sua carga de forma a não comprometer, ou até melhorar, o desempenho de suas aplicações. Para que esta distribuição seja possível, *datasets* devem ser particionados sobre servidores distribuídos.

A estratégia de particionamento de dados adotada nestes ambientes determina a escalabilidade do sistema gerenciador de banco de dados [Cong et al., 2007]. O problema relacionado ao particionamento está em reduzir a latência de consultas através da minimização da comunicação e transferência de resultados intermediários entre servidores do sistema. Embora este seja um problema conhecido, a disseminação de sistemas distribuídos de grande escala mostra que as soluções tradicionais não são adequadas [Tatarowicz et al., 2012, Pavlo et al., 2012, Yang et al., 2012].

Considere como exemplo de introdução ao problema, o grafo da Figura 1.2 que representa as relações de produtos, seus representantes e as respectivas cidades e estados atendidos. Suponha que este *dataset* tenha sido particionado, e que a partição p_1 seja atribuída ao *Servidor 1* na arquitetura da Figura 1, e que a partição p_2 seja atribuída ao *Servidor n*. Considere ainda que uma consulta deseja recuperar o nome dos representantes de cada um dos produtos. Neste caso é possível formar sub-consultas para recuperar os dados de cada um dos produtos de forma que a recuperação possa ser desempenhada

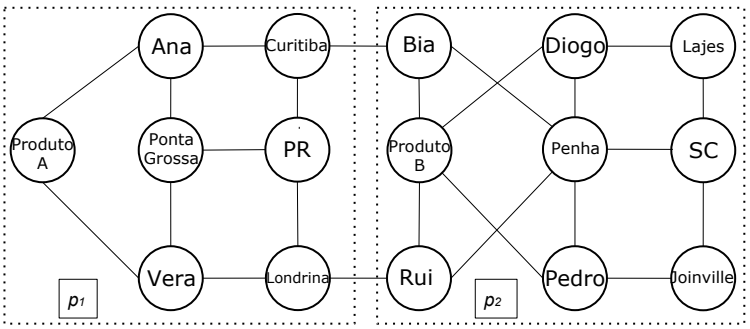


Figura 1.2: Um Exemplo do Particionamento de Dados

paralelamente e independentemente sobre os dois servidores. Assim, nenhuma troca de resultados se faz necessária entre os servidores, uma vez que os resultados parciais gerados por cada unidade poderão ser simplesmente unidos. O mesmo não seria possível se além dos nomes dos representantes se desejasse saber as cidades atendidas por eles. Neste caso, em especial a sub-consulta submetida ao *Servidor n*, as cidades atendidas por *Bia* e *Rui* não fazem parte da mesma partição do *Produto B*. A execução distribuída de sub-consultas como esta elevam o número de mensagens trocadas entre servidores e, por consequência, aumentam a latência, diminuem a vazão do sistema e podem levar a *deadlocks* caros. Minimizar este custo através de uma estratégia de particionamento adequada é o foco desta tese.

1.1 Definição do Problema

Seja D um *dataset* representado por um grafo (V, E), um particionamento $\mathcal{P} = \{P_1, ..., P_n\}$ a partir de D é assumido sobre n servidores de um repositório de dados distribuído. Dado um conjunto de consultas Q, $q \in Q$ representa uma consulta decomposta de forma que o processamento de consultas possa ser efetuado em paralelo sempre que possível, assim como mostrado no exemplo. A qualidade de um particionamento $\mathcal{P}$ é dada pela medida $\hat{P}$ em relação a q como definido a seguir:

Definition 1.1.1 *Dado um particionamento $\mathcal{P}$ de um grafo D, deixe $B(q)$ ser um conjunto de sub-grafos de D que representa os casamentos de uma consulta $q \in Q$ no grafo.*

A medida $\hat{P}$ de $\mathcal{P}$ com relação a q é definida por:

$$\hat{P}(q,\mathcal{P}) = 1 + \left|\{(D',P') \in B(q) \times \mathcal{P} | E(D') \cap E(P') \neq \oslash\}\right| - \left|B(q)\right| \tag{1.1}$$

A relação (D', P') representa o conjunto de sub-grafos nas partições de $\mathcal{P}$ que contenham arestas de $B(q)$. Em um particionamento ideal, a quantidade de elementos da relação (D', P') se equivale a quantidade de sub-grafos de $B(q)$ de forma que será necessário acessar apenas uma partição para responder a q. Assim, valores de $\hat{P}$ superiores 1 remetem à execução distribuída de q.

Considere que para cada consulta $q \in Q$ é atribuído um peso $f(q)$ que denota a quantidade de vezes em que q é acionada em um dado período de tempo. Tendo em vista um conjunto de consultas Q dado por uma carga de trabalho, o problema do particionamento de um *dataset* D corresponde a encontrar um particionamento $\mathcal{P}$ que minimiza a seguinte equação:

$$min \sum_{q_i \in Q}^{\mathcal{P}} f(q_i).\hat{P}(q_i,\mathcal{P}) \tag{1.2}$$

Caso a execução distribuída de uma consulta não possa ser evitada, por razões evidentes o tamanho das mensagens trocadas entre servidores constitui um fator a ser controlado. Para tanto, o tamanho de fragmentos gerados no processo de particionamento deve refletir um tamanho adequado para troca de mensagens na rede.

Existem diversos trabalhos que propõem soluções para o problema da escalabilidade do processamento de consultas sobre repositórios distribuídos e particionados. Porém, os métodos propostos para particionar baseiam-se em algoritmos que requerem analisar todo o grafo que corresponde ao conjunto de dados. Grafos volumosos são difíceis de particionar. De acordo com [Jiewen Huang, 2011], um grafo RDF de 270 milhões de triplas leva aproximadamente uma hora para ser particionado. Processos exaustivos podem se tornar impraticáveis em um contexto em que *petabytes* de dados são acessados. Sobretudo, o raciocínio dos principais métodos relacionados baseiam-se na estrutura do grafo

para gerar partições que não expressam padrões de consulta da carga de trabalho. Como resultado, o desempenho da consulta diminui quando dados requeridos pela mesma consulta é distribuído por servidores diferentes. Adicionalmente, algumas soluções tentam resolver este problema através de um método de replicação de dados agressivo. Assim, grandes conjuntos de dados tornam-se ainda maiores. Com o intuito de superar estas deficiências de soluções correntes, esta tese propõe um método para o particionamento de dados baseado na hipótese de que é possível promover a escalabilidade do processamento de consultas sobre bases de dados particionadas de acordo com informações de carga de trabalho aplicadas a estruturas de bancos de dados.

1.2 Objetivos

O objetivo geral desta tese é definir uma metodologia de particionamento de dados capaz de promover a escalabilidade de sistemas de bancos de dados em ambientes sujeitos a grandes volume de dados e dispostos sobre a arquitetura em nuvem idealizada pela Figura 1.2.

Para atingir este objetivo geral, foram definidos os seguintes objetivos específicos:

— Propor uma abordagem de caracterização da carga de trabalho sobre a qual um banco de dados está sujeita;

— Prover uma solução não-exaustiva para o particionamento de dados baseado em informações de carga de trabalho. A solução deve dar suporte ao particionamento dos modelos XML e RDF destacados anteriormente;

— O processo de particionamento deve prever a fragmentação dos dados em unidades de armazenamento com tamanhos compatíveis ao tamanho adequado para mensagens na rede.

— Avaliar a solução produzida através de estudos experimentais.

1.3 Contribuições

As principais contribuições desta tese são:

— Uma abordagem de caracterização da carga de trabalho capaz de sumarizar a carga prevista sobre um esquema de banco de dados em termos de frequências de execução de consultas e seus respectivos padrões de acesso;

— Um método para o particionamento de dados baseado em heurísticas sobre a caracterização de uma carga de trabalho. O problema do particionamento é tratado em duas etapas. Na primeira, o processo considera a carga de trabalho juntamente com um limiar de armazenamento para produzir fragmentos de dados compatíveis com tamanhos adequados para mensagens na rede. No segundo e último passo, fragmentos que estão relacionados pela carga de trabalho são co-alocados nos servidores do repositório de dados distribuído;

— Um sistema, nomeado *ClusterRDF*, que implementa o método de particionamento proposto sobre *datasets* RDF sobre a arquitetura de processamento idealizada;

— Estudos experimentais que validam o método proposto sobre *datasets* XML e RDF.

1.4 Estrutura do Documento

Este trabalho está organizado em mais 5 capítulos. No capítulo 2 são apresentadas características de repositórios de dados na *nuvem*, arquiteturas de sistemas de banco de dados neste contexto e maiores detalhes sobre o problema do particionamento de dados. O capítulo seguinte dedica-se à análise de abordagens relacionadas à fragmentação e alocação de dados, juntamente com a apresentação de modelos e técnicas de particionamento de grafos adequadas ao contexto deste trabalho. O capítulo 4 apresenta o projeto da metodologia de particionamento de dados que atende aos objetivos anteriormente citados. O sistema *ClusterRDF* desenvolvido para simular a execução de consultas sobre a arquitetura idealizada é apresentado no capítulo 5. No capítulo 6 são apresentados estudos experimentais

que validam a eficácia da solução proposta por esta tese. Por fim, o capítulo 7 dedica-se às conclusões deste trabalho, trabalhos futuros e a lista de publicações realizadas durante o doutorado.

CAPÍTULO 2

GERENCIAMENTO DE DADOS NA NUVEM

O amadurecimento dos diversos conceitos de computação em *nuvem* iniciou uma mudança nos aspectos operacionais e econômicos da computação, principalmente devido à introdução de serviços com custos acessíveis, sob demanda e em tempo real. Um semblante desta mudança de paradigma se deve a novas formas de manusear e disponibilizar dados através de arquiteturas distribuídas e baseadas em serviços, de forma que o acesso aos dados seja facilitado e ubíquo [Buyya et al., 2008]. Esse modelo de computação habilita uma escalabilidade maciça de serviços e realça oportunidades de colaboração, integração e análise sobre uma plataforma comum e compartilhada. Para viabilizar e dar suporte a esse novo modelo surgiu a necessidade de sistemas de banco de dados centrados em escalabilidade a custa de alguns benefícios presentes em SGBDs tradicionais. Apesar de um SGBD tradicional oferecer uma mistura de simplicidade, flexibilidade, robustez e confiabilidade, algumas dessas funcionalidades resultam no aumento significativo da complexidade quando se tenta assegurá-las em um sistema distribuído em centenas ou milhares de nós. Remover esta complexidade para alavancar sistemas escaláveis é o princípio do tipo de sistema comumente chamado de *NoSQL*, pois abdica principalmente da interface SQL de alto-nível e da garantia de um nível de consistência forte em transações comuns a SGBDs tradicionais. Esta condição é favorável a alguns tipos de aplicações que baseiam-se em um nível de consistência relaxada e que são pouco orientadas à interface. Este é o caso, por exemplo, de sistemas OLAP que se beneficiam da distribuição da carga de trabalho na *nuvem* para torná-los mais escaláveis que em SGBDs paralelos tradicionais.

Como alternativa aos bancos *NoSQL*, uma classe de bancos de dados na *nuvem* tem sido denominada *NewSQL* [Aslett, 2011]. A promessa desta classe é continuar provendo escalabilidade e flexibilidade, enquanto mantem o suporte a uma interface de alto-nível

e garantias ACID. O objetivo é dar suporte a cargas de trabalho do tipo OLTP, fundamentais para a indústria de banco de dados. No entanto, existem alguns desafios em assegurar tais propriedades. Em geral, repositórios na *nuvem* baseiam-se em modelos de implementação bastante simples. Além disto, é difícil manter a consistência em um ambiente distribuído sobre regiões geográficas distantes. Neste capítulo estes e outros desafios são contextualizados com relação aos aspectos fundamentais de repositórios de dados na *nuvem* e da distribuição de dados para estes sistemas.

2.1 Repositório de Dados na *Nuvem*

Segundo Cattell [Cattell, 2011], existem quatro características comuns a repositórios de dados que são escaláveis: (i) são baseados em uma interface simples; (ii) conferem a habilidade de escalar horizontalmente um sistema sobre muitos servidores; (iii) dispõem de uso eficiente de índices distribuídos e de memória; e (iv) permitem um ajuste dinâmico da distribuição da carga de trabalho. Todas estas características estão associadas à natureza *shared-nothing* de redes par-a-par (p2p) empregadas por estes repositórios. Em especial, neste contexto as redes de sobreposições p2p são estruturadas através de uma tabela de dispersão distribuída (DHT - *Distributed Hash Table*) capaz de prover uma interface de uso geral para a indexação, armazenamento e distribuição de dados [Rowstron and Druschel, 2001]. A principal função de uma DHT está associada à sua função de dispersão responsável por particionar um espaço de chaves entre um conjunto de nós distribuídos, semelhante à associação entre chaves e valores de uma tabela de dispersão tradicional. Aplicações distribuídas baseadas em DHT herdam aspectos de escalabilidade, robustez e facilidade de operação [Stoica et al., 2003].

Sistemas DHT provêem uma interface genérica com três operações. A operação $put(chave, valor)$, armazena um valor associado a uma dada chave em um nó da rede p2p. A recuperação de um valor associado a uma determinada chave é possível através da operação $get(chave)$. A terceira operação, $remove(chave)$, remove um par chave-valor as-

sociado à chave dada. Sistemas p2p estruturados formam redes de sobreposição nas quais os tempos de busca e manutenção da DHT necessitam apenas de um número de acessos de ordem logarítmica sobre os nodos que compõem a rede. Por esta razão, aplicações baseadas em redes p2p estruturadas estão entre as aplicações mais escaláveis [Valduriez and Pacitti, 2004]. No entanto, existem variações nos modelos de dados empregados e nas arquiteturas para repositórios na *nuvem*.

2.1.1 Modelos de Dados

Embora a grande maioria dos repositórios de dados na *nuvem* dá suporte ao armazenamento de dados complexos, eles diferem na estrutura de armazenamento. Os modelos de dados podem ser classificados em modelos chave-valor, documentos ou registros extensíveis [Cattell, 2011]. O modelo *chave-valor* é a forma de representação mais aproximada àquela imposta por uma rede de sobreposição baseada em DHT. Neste caso, a unidade de armazenamento é um par chave-valor, onde a chave atua como um índice para endereçamento do valor associado. O modelo de *documentos* é similar ao modelo chave-valor, mas neste caso o valor pode ser estruturado, por exemplo através um conjunto de pares chave-valor. Os documentos são indexados por uma única chave e organizados em domínios. Por fim, os *registros extensíveis* são organizados através do aninhamento de colunas e super-colunas. A Figura 2.1 mostra exemplos para os modelos de dados apresentados. Os sistemas Voldemort [Voldemort, 2012], Dynamo [DeCandia et al., 2007], Riak [Klophaus, 2010], Redis [red, 2012], Scalaris [Zuse Institute Berlin, 2012] e Bamboo [bam, 2012] são exemplos de repositórios chave-valor. Dentre os repositórios que aderem ao modelo de documentos, pode-se citar o SimpleDB [AWS, 2012], CouchDB [Apache, 2012a] e MongoDB [mon, 2012]. Por fim, HBase [Apache, 2012b], HyperTable [hyp, 2012] e Cassandra [Lakshman and Malik, 2009] são exemplos de repositórios com registros extensíveis.

O modelo chave-valor é de fato o mais simples e flexível, o que é ideal para um modelo físico neste contexto. Repositórios chave-valor são frequentemente associados ao termo *NoSQL* por apresentarem um modelo simples para garantir melhor desempenho e flexi-

```
(1, "(RG, 123),(nome, João)")
(2, "(RG, 456),(nome, Maria)")
(2, "(RG, 789),(nome, Pedro)")
```

(a) Chave-valor

```
Domínio Aluno
(1, {(RG,123),(nome, João)})
(2, {(RG, 456),(nome, Maria)})
(3, {(RG, 789),(nome, Pedro)})

Domínio Curso
(1, {(nome, CC)})
(2, {(sigla, EE)})
```

(b) Documento

Aluno	coluna		super-colunas			
A1	nome	curso	A1BD		A1EE	
	João	CC	disc	nota	disc	nota
			BD	95	EE	78
An	...					

(c) Registro Extensível

Figura 2.1: Modelos de Dados para Repositórios na *Nuvem*

bilidade de representação. No entanto, como um modelo lógico, ele é pouco expressivo e inadequado para representar a complexidade dos dados provenientes da camada de aplicação. A simplicidade da representação do modelo *chave-valor* também se estende para os baseados em *documentos* e *registros extensíveis*, pois repositórios baseados nestes modelos apresentam uma interface bastante simples e de pouca expressividade. Esta simplicidade gera a necessidade de implementar certas operações no nível da aplicação. Para maior expressividade das operações sobre estes repositórios, alguns trabalhos propõem o mapeamento destes modelos de implementação e suas respectivas operações para um modelo lógico de alto nível. A ideia de desenvolver uma interface sobre repositórios existentes tem sido adotada por diversos trabalhos [Arnaut et al., 2011,Curino and et al, 2011,Bunch and et al, 2010, Egger, 2009, Cooper and et al, 2008]. Em [Arnaut et al., 2011] é demonstrado que o custo em se manter uma camada relacional aplicada sobre o repositório chave-valor Scalaris é mínimo.

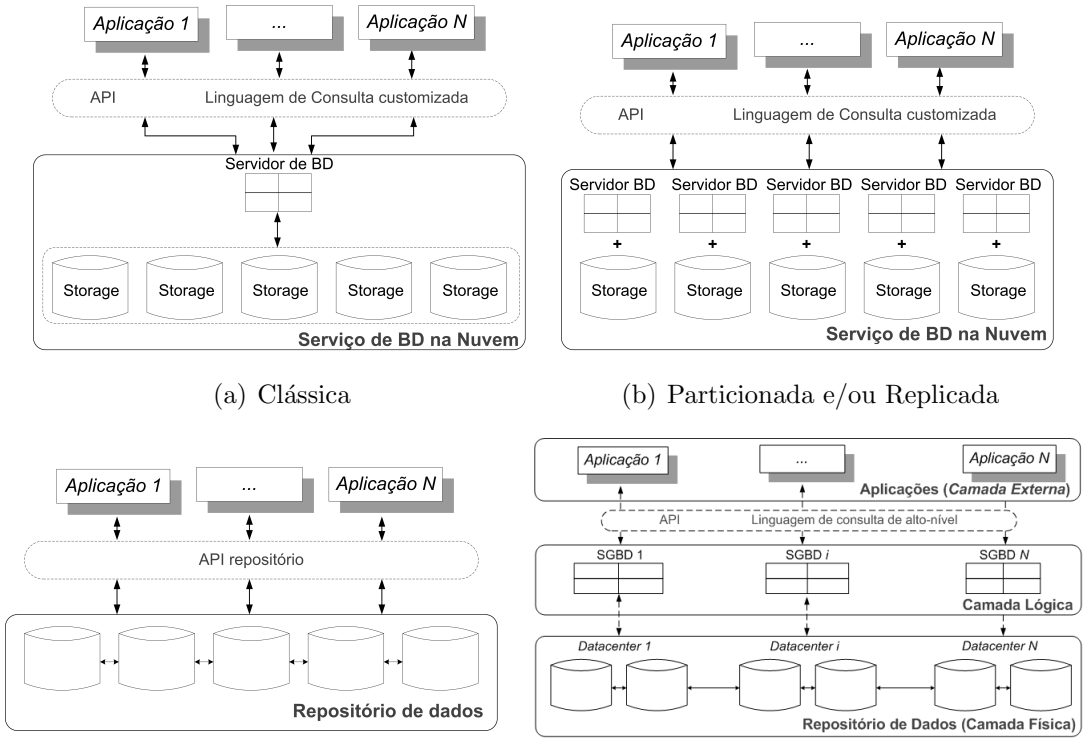
Considerando que grande parte das aplicações que utilizam SGBDs escaláveis são orientadas à Web, assume-se que XML e RDF sejam modelos ideais para atuar no nível lógico. Além de constituírem padrões para o intercâmbio e publicação de dados na Web, a flexibilidade destes modelos permite representar uma grande variedade de informações. Assim, para manter a simplicidade do modelo físico e a expressividade e flexibilidade no nível da aplicação, considera-se que dados definidos através destes modelos podem ser adequadamente convertidos para um dos modelos de dados empregados pelos repositórios

e vice-versa.

2.1.2 Arquiteturas de Processamento

Segundo [Kossmann et al., 2010], existem 3 variações de arquiteturas para o processamento de transações na *nuvem*. As três primeiras ilustrações da Figura 2.2 representam as arquiteturas definidas em [Kossmann et al., 2010]. A primeira delas corresponde a arquitetura *clássica* em que um servidor de BD centralizado é responsável por processar e submeter as operações a um sistema de armazenamento escalável. Embora as principais propriedades de um SGBD tradicional possam ser asseguradas sobre um sistema de armazenamento escalável através do controle centralizado do SGBD, o servidor de BD representa um gargalo quando submetido a sobrecargas. Neste caso, a solução para evitar a sobrecarga é tentar escalar o sistema verticalmente, isto é, substituir o servidor por uma máquina de maior poder computacional. No entanto, este provisionamento pode se tornar caro e ineficiente, uma vez que sistemas na *nuvem* estão sujeitos a picos constantes na carga de trabalho. O AWS MySQL e o AWS RDS [Amazon, 2012] são exemplos de sistemas que empregam esta arquitetura clássica.

Uma arquitetura alternativa para eliminar este ponto de contenção é a utilização de técnicas de particionamento e replicação de bancos de dados distribuídos sobre diversos SGBDs. Nesta solução, cada SGBD poderá executar sobre máquinas menos custosas para gerenciar um sub-conjunto do BD e assim sustentar a carga. No entanto, variações na carga de trabalho ou alterações na composição da rede podem levar a um remanejamento custoso das partições de dados entre os SGBDs. Além disto, operações de atualização envolvendo bases replicadas poderão requerer um controle centralizado de réplicas principais para manter as demais réplicas consistentes. Neste caso, ainda assim existe um ponto de contenção capaz de limitar a escalabilidade do sistema. Como exemplos de sistemas que aplicam este tipo de arquitetura, pode-se citar o MySQL/R e SimpleDB da Amazon Web Services [Amazon, 2012], assim como o Google AppEng [Google, 2012] e o MS Azure [Sengupta, 2008].



(a) Clássica (b) Particionada e/ou Replicada

(c) Controle Distribuído (d) Arquitetura Ideal

Figura 2.2: Arquiteturas para o Processamento de Transações na *Nuvem*

A terceira e última arquitetura relacionada em [Kossmann et al., 2010], refere-se a uma solução encontrada para escalar estas aplicações através de uma composição do tipo *shared-nothing*, onde o controle do sistema ocorre de forma completamente distribuída sobre nós independentes e auto-suficientes. Através deste tipo de composição é possível atingir escalabilidade quase ilimitada pela adição de novos nós à rede [Baker et al., 2011]. Este tipo de escalabilidade é conhecida como escalabilidade horizontal e contrasta com escalabilidade vertical, que representa a substituição de máquinas por outras de maior poder computacional. Escalar um sistema horizontalmente é mais barato do que substituir os recursos computacionais vigentes. Em sistemas em *nuvem*, este conceito é interessante pois adere ao conceito *on-demand* e *pay-as-you-go* proposto por estes ambientes ou serviços. No entanto, os sistemas existentes que são baseados em controle distribuído apresentam um conjunto de funcionalidades bastante limitada. O S3 [Brantner et al., 2008] é um exemplo deste tipo de sistema de armazenamento escalável.

A quarta arquitetura da Figura 2.2 representa uma arquitetura ideal para o pro-

cessamento de transações neste contexto. Nesta composição, as características de um SGBD tradicional, como uma interface de alto-nível, são asseguradas por um conjunto de SGBDs distribuídos e que gerenciam de forma distribuída e independente um sistema de armazenamento escalável. A arquitetura estratificada proposta oferece independência de dados física, permitindo que diferentes abordagens para mapeamento lógico-físico sejam empregadas, enquanto o repositório de dados distribuído é responsável por fornecer escalabilidade, disponibilidade, replicação de dados e garantias ACID. Esta independência de modelos também habilita o emprego de diferentes estratégias de fragmentação e alocação de dados sobre os modelos lógico e físico adotados.

Repositórios de dados na *nuvem* comumente baseiam-se em um nível de consistência relaxada, conhecido como BASE (*Basically Available, Soft state, Eventually consistent*). A razão disto está associada à escolha de duas propriedades do teorema CAP (*Consistency, Availability e Partition Tolerance*). Segundo [Brewer, 2000], apenas duas dessas propriedades podem ser asseguradas em sistemas distribuídos que compartilham dados. Assim, grande parte dos repositórios asseguram disponibilidade e tolerância a partições da rede em detrimento à consistência. Por esta razão, em sistemas como estes a consistência dos dados é eventual, possibilitando que existam duas cópias de um mesmo item assumindo valores diferentes. No entanto, existe um grupo de sistemas propondo favorecer a consistência em detrimento à disponibilidade dos dados ou tolerâncias a partições na rede. Os repositórios Scalaris [Schütt et al., 2008] e G-Store [Das et al., 2010] são exemplos destes sistemas. O Scalaris, por exemplo, assegura consistência e tolerância a partições na rede enquanto sacrifica a disponibilidade dos dados. Em caso de partições na rede, as operações somente serão bem sucedidas na partição que engloba a maioria das réplicas de um determinado item. Para viabilizar esta estratégia são utilizados algoritmos baseados em quorum, dentre os quais destaca-se o protocolo de consenso Paxos [Lamport, 1998] que tem se mostrado efetivo em assegurar níveis fortes de consistência para repositórios que o utilizam [Rao et al., 2011].

No contexto de uma transação, a habilidade de dar suporte a níveis fortes de con-

sistência também está fortemente relacionado à habilidade de manter a localidade dos dados. A grande maioria de repositórios baseados em DHT não detém qualquer controle sobre o local de armazenamento dos dados visto que o nó da rede que deve armazenar um par chave-valor é determinado pelo resultado de uma função de espalhamento. Entretanto, no projeto físico de banco de dados o controle sobre a localidade de dados é um fator determinante para a formação de *clusters* de dados. Em um BD tradicional, estratégias de agrupamentos são adotadas para minimizar o número de acesso a discos, enquanto que em um sistema distribuído elas são aplicadas para minimizar o número de transações distribuídas e, consequentemente, o custo da comunicação de dados na rede. O controle de localidade de dados, presente em alguns repositórios, é dado pelo particionamento por intervalos de chaves ou pela manutenção da ordem lexicográfica de chaves. O particionamento por intervalo é adotado por sistemas como PNUTS [Cooper and et al, 2008], BigTable [Chang and et al, 2006] e Spinnaker [Rao et al., 2011], enquanto a ordem lexicográfica é mantida pelo Scalaris [Schütt et al., 2008]. Em ambas as abordagens, valores associados com chaves similares são mantidos no mesmo servidor ou em servidores próximos. Neste contexto, a habilidade de conferir a garantia de propriedades ACID está associada à distribuição de dados no sistema de armazenamento. Para tanto, o projeto de distribuição de dados deve idealmente permitir que dados relacionados por uma mesma transação sejam armazenados em um mesmo servidor. De outro modo, torna-se difícil ou até impraticável manter um nível forte de consistência sobre dados distribuídos através de localidades geograficamente distantes.

2.2 Distribuição de Dados

A estratégia de distribuição de dados em sistemas largamente distribuídos está fortemente relacionada ao tipo de sistema sobre o qual está sujeito. Não obstante ao tipo de carga de trabalho, o objetivo da distribuição de dados na *nuvem* para qualquer tipo de carga de trabalho visa a escalabilidade das aplicações. Por exemplo, para sistemas OLAP

o fator determinante para torná-los escaláveis é conseguir um nível de paralelização que confira a eficiência de suas operações sobre um grande volume de dados. Estas operações correspondem a transações longas com níveis de consistência relaxados e fortemente orientadas à leitura. Por outro lado, em sistemas OLTP a paralelização de uma transação não é o foco principal, uma vez que correspondem a transações curtas e que, em geral, acessam um conjunto pequeno de dados. A escalabilidade destes sistemas está associada a tempos de resposta baixos para suas transações. A obtenção deste bom desempenho envolve principalmente evitar a execução de transações distribuídas e reduzir o número de requisições à rede de sobreposição.

Um projeto de distribuição de dados adequado deve idealmente manter itens de dados requeridos por uma mesma transação em um mesmo nó da rede. Para este fim, a base de dados deve ser fragmentada para manter os itens de dados fortemente relacionados em um mesmo fragmento de acordo com a carga de trabalho sob a qual está sujeita. Quando não for possível manter todos os itens relacionados por uma transação em um mesmo fragmento, a estratégia de alocação dos fragmentos deverá manter fragmentos com itens relacionados em nós próximos, preferencialmente pertencentes a um mesmo servidor.

Considere como um exemplo, o esquema de banco de dados introduzido pela Figura 2.3. No exemplo, as entidades envolvidas em um sistema de suprimentos estão relacionadas, onde os nós representam os itens de dados e o valor das arestas denotam a frequência na qual pares de itens são pesquisados em conjunto pelo sistema. Assuma que o sistema é mantido por um banco de dados distribuído que deverá ser fragmentado e distribuído pelas unidades de armazenamento da rede. Para viabilizar o acesso ao banco, o tamanho das requisições deve se enquadrar a um limiar que representa uma unidade de transferência na rede. Neste exemplo, assume-se que o limiar é definido de forma que cada transferência poderá conter até 4 itens de dados. Considera-se que o limiar de transferência também determina o tamanho máximo de cada fragmento uma vez que ele representa a unidade de armazenamento do repositório. Logo, cada fragmento poderá conter até 4 itens de dados. Neste exemplo, duas opções de

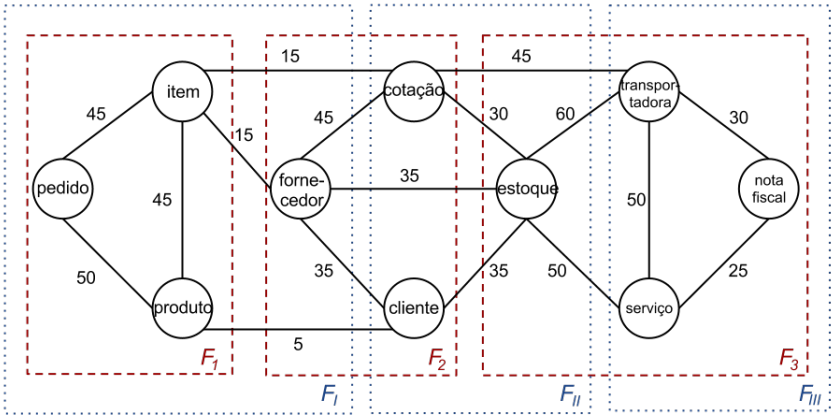


Figura 2.3: Problema da Distribuição de Dados

fragmentação dos dados são apresentadas. A opção representada pela linha tracejada mantem nós fortemente relacionados em um mesmo fragmento de acordo com as frequências das arestas ($F_1 = \{item, pedido, produto\}$, $F_2 = \{fornecedor, cotacao, cliente\}$ e $F_3 = \{estoque, transportadora, notafiscal, servico\}$), enquanto o esquema representado pelas linhas pontilhadas desconsidera as informações da carga de trabalho ($F_I = \{item, pedido, produto, fornecedor\}$, $F_{II} = \{cotacao, cliente, estoque\}$ e $F_{III} = \{transportadora, notafiscal, servico\}$).

Embora a primeira opção de fragmentação considera a carga de trabalho para definir o esquema de fragmentação, ainda assim os fragmentos F_2 e F_3 apresentam dados fortemente relacionados. Uma forma de mantê-los agrupados é alocar ambos em um mesmo servidor na rede. Para definir a estratégia de alocação, assuma que existem 2 servidores na rede (S_1 e S_2) e que cada um pode armazenar até dois fragmentos. Considere também 2 esquemas de alocação, sendo que o primeiro define $S_1 = \{F_1\}$ e $S_2 = \{F_2, F_3\}$, enquanto o segundo define $S_1 = \{F_I, F_{II}\}$ e $S_2 = \{F_{III}\}$. Uma transação envolvendo os itens $estoque - transportadora - servico$ requer acessar apenas o fragmento F_3 em S_1 pela primeira opção de distribuição dos dados. Em contrapartida, a segunda opção resultará em uma execução de transação distribuída visto que os fragmentos F_{II} e F_{III} serão requeridos a partir dos servidores S_1 e S_2. Se esta mesma transação necessitar ainda acessar o item *cliente*, mais um fragmento (F_2) seria requerido pela primeira opção de distribuição.

Embora uma nova requisição a rede seja necessária para acessar mais este fragmento, a transação continuará a ser executada localmente pelo servidor S_2 visto que ambos os fragmentos requeridos estão co-alocados nele.

Mesmo quando os dados de uma mesma transação estão dispostos em fragmentos de um mesmo servidor, verifica-se que o número de requisições a rede necessárias para recuperar tais fragmentos pode degradar o desempenho do sistema [Arnaut et al., 2011, Ribas et al., 2011]. Em alguns repositórios de dados na *nuvem*, existe um recurso que habilita o empacotamento de requisições endereçadas para um mesmo servidor da rede [Zuse Institute Berlin, 2012]. Na presença e uso deste recurso, o custo relacionado ao número de fragmentos requeridos de um mesmo servidor pode ser reduzido ou até desprezado.

Existem diversos desafios associados ao problema de distribuição de dados em sistemas de bancos de dados na *nuvem*. Inicialmente destaca-se o método de caracterização da carga de trabalho. Semelhantemente ao grafo da Figura 2.3, o método deve ser capaz de representar as projeções e seleções de consultas que determinam as conexões entre os nodos. Esta representação, bem como os métodos de fragmentação e alocação de dados deverão considerar a não-exaustão do processo em um ambiente sujeito a grandes volumes de dados. No Capítulo 4, estes desafios são enquadrados em uma solução de projeto distribuição de dados adequada a este contexto.

CAPÍTULO 3

FRAGMENTAÇÃO E ALOCAÇÃO DE DADOS

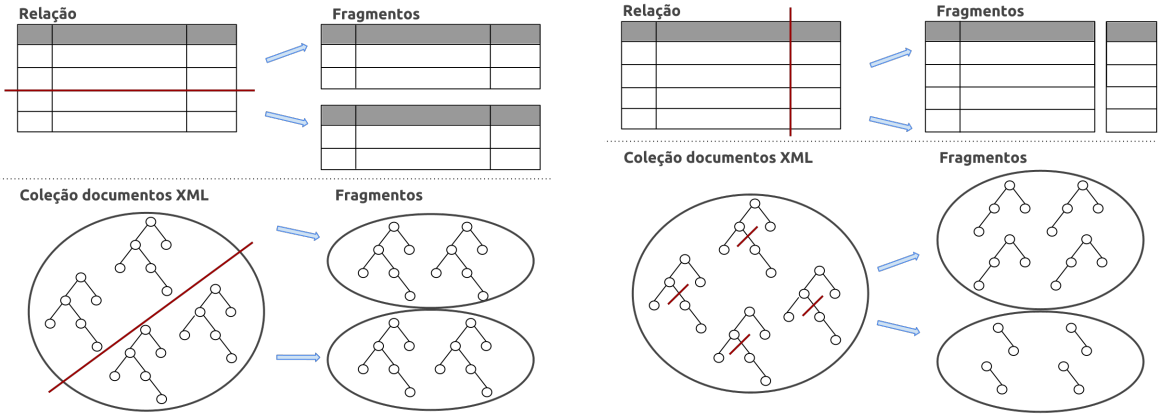
Segundo [Zilio, 1998], os primeiros objetivos do projeto físico de um BD distribuído são o particionamento das relações em fragmentos e a posterior alocação dos mesmos nos servidores do sistema. Conforme introduzido pelo capítulo anterior, estes objetivos definem o esquema da distribuição de dados do sistema. Primeiramente, a fragmentação de dados divide o banco de dados em fragmentos. Em seguida, a alocação de dados envolve encontrar uma distribuição ótima para os fragmentos nos servidores da rede enquanto satisfaz restrições envolvendo o tempo de resposta, armazenamento e processamento. A razão para esta divisão em dois passos está relacionada ao melhor tratamento da complexidade do problema [Ozsu and Valduriez, 2011]. Neste capítulo são apresentadas abordagens existentes para solucionar o problema da fragmentação e da alocação de dados.

3.1 Fragmentação de Dados

A fragmentação de dados em sistemas distribuídos pode melhorar a vazão do sistema e o desempenho de consultas. Este objetivo pode ser atingido de duas maneiras [Son and Kim, 2004]. Primeiramente, o número de fragmentos requeridos pelas consultas deve ser reduzido. Em segundo lugar, a quantidade de dados desnecessários recuperados deve ser reduzida [Ceri et al., 1987]. Em um ambiente distribuído, a primeira forma é a mais importante visto que ela pode reduzir a transmissão de dados entre nodos. Além disto, com uma alocação apropriada destes fragmentos, a vazão do sistema pode ser melhorada em termos do processamento paralelo de consultas sobre fragmentos alocados em servidores diferentes [Ozsu and Valduriez, 2011, Sacca and Wiederhold, 1985].

Os modelos de distribuição de dados geralmente dão suporte à fragmentação horizontal e vertical. Em bancos de dados relacionais, a abordagem tradicional gera fragmentos que

possuem um sub-conjunto de tuplas, enquanto a fragmentação vertical produz fragmentos que contêm um sub-conjunto de atributos das relações. Na fragmentação horizontal, o agrupamento de tuplas em fragmentos se dá pela análise de predicados, assim como na fragmentação vertical atributos são agrupados pela projeção das consultas. Observa-se que os mesmos critérios de agrupamento podem ser aplicados a outros modelos. Para o modelo XML, por exemplo, fragmentos verticais correspondem a sub-árvores de um esquema XML e fragmentos horizontais correspondem a conjuntos de documentos de uma coleção homogênea de documentos [Figueiredo et al., 2010, Kling et al., 2010]. Os exemplos da Figura 3.1 ilustram as estratégias horizontal e vertical para BDs relacionais e XML.



(a) Fragmentação Horizontal (b) Fragmentação Vertical

Figura 3.1: Fragmentação de Dados XML

Dados em RDF são comumente armazenados por bancos de dados relacionais conhecidos como *Triple Stores*. Em um repositório *triple store*, cada conjunto de dados é representado por uma tabela que contem 3 campos, cada qual representando os elementos de uma tripla RDF (sujeito, predicado e objeto). Um exemplo desta forma de armazenamento é apresentado pela Figura 3.2(a). A fragmentação horizontal e vertical de RDF foi inicialmente proposta sobre esta composição. Fragmentos horizontais correspondem a um agrupamento baseado em valores de sujeitos ou objetos das triplas. No caso deste exemplo, uma chave de particionamento poderia ser tomada sobre os valores do sujeito

identificador. Assim, uma tabela poderia ser criada para um conjunto de triplas de um mesmo valor de ID, ou até mesmo de um intervalo de valores. Fragmentos verticais foram introduzidos pelo projeto Jena [Wilkinson et al., 2003] através da criação de tabelas de propriedades que ficou conhecido como *Property Tables*. Assim como ilustrado pela Figura 3.2(b), propriedades acessadas frequentemente em conjunto são agrupadas em uma única tabela juntamente com seus respectivos sujeitos e predicados. Propriedades que não foram atribuídas a nenhum agrupamento são mantidas na estrutura de uma *property table* original. Uma extensão deste modelo, chamada de *Property-class Tables* [Chong et al., 2005], corresponde a uma fragmentação híbrida como apresentado pela Figura 3.2(c). Neste caso, cada tabela corresponde a uma classe que agrupa triplas com os mesmos valores para o predicado *type*. Os campos destas tabelas de classe correspondem a propriedades utilizadas por triplas destas classes. Da mesma forma que ocorria nas *property tables*, triplas que não foram atribuídas a nenhuma das classes são mantidas na estrutura original.

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

(a) RDF Triple Store

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

(b) Fragmentação Vertical

Class: BookType

Subj.	Title	Author	copyright
ID1	"XYZ"	"Fox, Joe"	"2001"
ID3	"MNP"	NULL	NULL
ID6	NULL	NULL	"2004"

Class: CDType

Subj.	Title	Artist	copyright
ID2	"ABC"	"Orr, Tim"	"1985"
ID5	"GHI"	NULL	"1995"

Subj.	Prop.	Obj.
ID2	language	"French"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

(c) Fragmentação Híbrida

Figura 3.2: Fragmentação de Dados RDF

Observe que para propor estratégias de fragmentação adequadas é necessária a análise da carga de trabalho sob a qual um BD está sujeito. Dependendo da quantidade de informações de carga disponíveis, é possível que exista um vasto número de possibilidades de esquemas de fragmentação. Estas possibilidades, em geral, são avaliadas por uma função objetivo que reflete as necessidades que a fragmentação deve atender. Além da

função objetivo, as abordagens de fragmentação diferenciam-se pelo processo aplicado na busca por uma solução ótima. Estes processos podem ser classificados como *exaustivos* ou *heurísticos* [Zilio, 1998]. Um processo de busca exaustivo tem a vantagem de encontrar uma composição ótima com respeito a uma função objetivo. Porém, neste caso, o tempo consumido pelo processo não é determinado em tempo polinomial visto tratar-se de um problema NP-difícil. A maioria dos métodos exaustivos utiliza uma solução conhecida como *what if?*, onde diversas possibilidades de fragmentação são simuladas pelo otimizador de um SGBD, produzindo os respectivos custos para assim determinar as recomendações mais adequadas. Dentre as abordagens que aplicam este método, destacam-se as ferramentas AutoAdmin [Agrawal et al., 2004] da Microsoft, o DB2 Database Advisor da IBM [Zilio et al., 2004] e a AutoPart [Papadomanolakis and Ailamaki, 2004]. Se comparados a processos exaustivos, os processos heurísticos reduzem o número de possibilidades avaliadas e podem (ou não) levar a soluções ótimas ou próximas ao ótimo.

Existem diversos trabalhos que propõem soluções de fragmentação e alocação de dados. A Tabela 3.1 sumariza um conjunto destes trabalhos propostos tanto para o modelo de dados *Relacional* quanto para os modelos *XML* e *RDF*. As abordagens são inicialmente classificadas pelo processo de busca das soluções de fragmentação e/ou alocação de dados, que pode ser *Heurístico*, *Exaustivo* ou *Semi-exaustivo*. No contexto do processo de busca, algumas soluções definem um modelo de custo para classificar as soluções, como ocorre em [Zilio et al., 2004], [Papadomanolakis and Ailamaki, 2004], [Pavlo et al., 2012] e em [Nehme and Bruno, 2011]. O método de fragmentação adotado pode ser *Horizontal*, *Vertical* ou *Híbrido*. Soluções classificadas como híbridas equivalem a abordagens que empregam ambas as estratégias horizontais e verticais de fragmentação. Juntamente com estratégias de fragmentação de dados, os trabalhos podem propor ainda soluções para a alocação de dados e para tipos de cargas de trabalho específicas. Estes critérios de classificação são detalhados nas seções a seguir.

Abordagem	Modelo	Processo	Modelo Custo	Fragmentação	Alocação	Carga
[Ceri and Pelagatti, 1984]	Relacional	Heurístico		Horizontal	√	
[Navathe and Ra, 1989]	Relacional	Heurístico		Vertical		
[Zilio, 1998]	Relacional	Heurístico		Horizontal	√	OLTP
[Zilio et al., 2004]	Relacional	Exaustivo	DB2	Horizontal	√	
[Agrawal et al., 2004]	Relacional	Semi-exaustivo		Híbrido	√	
[Papadomanolakis and Ailamaki, 2004]	Relacional	Semi-exaustivo	SQL Server	Híbrido	√	
[Curino et al., 2010]	Relacional	Heurístico		Híbrido		
[Nehme and Bruno, 2011]	Relacional	Semi-exaustivo	SQL Server	Horizontal		OLAP
[Pavlo et al., 2012]	Relacional	Heurístico	próprio	Horizontal		OLTP
[Ma and Schewe, 2003]	XML	Heurístico		Híbrido		
[Gertz and Bremer, 2003]	XML	Heurístico		Híbrido		
[Bonifati et al., 2004]	XML	Heurístico		Híbrido	√	
[Kanne and Moerkotte, 2006]	XML	Heurístico		Híbrido		
[Abiteboul et al., 2008]	XML	Heurístico		Horizontal	√	
[Shnaiderman et al., 2008]	XML	Heurístico		Híbrido		
[Bordawekar and Shmueli, 2008]	XML	Heurístico		Híbrido		
[Shnaiderman and Shmueli, 2009]	XML	Heurístico		Híbrido		
[Figueiredo et al., 2010]	XML	Heurístico		Híbrido		OLAP
[Abadi et al., 2009]	RDF	Heurístico		Vertical		
[Jiewen Huang, 2011]	RDF	Exaustivo		Híbrido	√	
[Mulay and Kumar, 2012]	RDF	Heurístico		Vertical		
[Hose and Schenkel, 2013]	RDF	Exaustivo		Híbrido	√	
[Yang et al., 2013]	RDF	Exaustivo		Híbrido	√	
[Yang and Wu, 2013]	RDF	Heurístico		Horizontal	√	OLTP
[Ozsu et al., 2013]	RDF	Heurístico	próprio	Híbrido		
[Zeng et al., 2013]	RDF	Heurístico		Híbrido	√	OLAP e OLTP
[Galárraga et al., 2014]	RDF	Exaustivo		Horizontal	√	

Tabela 3.1: Abordagens de Fragmentação de Dados

3.1.1 Fragmentação Horizontal

O princípio que classifica soluções de fragmentação como horizontais é o particionamento de unidades lógicas de dados em conjuntos homogêneos de dados. Por exemplo, no modelo relacional as unidades lógicas representadas pelas relações são particionadas de forma que cada fragmento contenha um conjunto homogêneo de dados, neste caso um sub-conjunto de tuplas da relação. Existem diversas estratégias para determinar o agrupamento de dados nestes conjuntos. Grande parte destas soluções definem sua estratégia baseando-se em chaves de particionamento, enquanto outras provêem soluções mais refinadas tendo como base a análise de predicados.

3.1.1.1 Abordagens baseadas em Chaves de Particionamento

No modelo relacional, uma chave de particionamento de uma relação é um sub-conjunto de seus atributos cujos respectivos valores são usados para mapear cada tupla para uma das unidades de armazenamento, chamadas de fragmentos [Zilio, 1998]. Os métodos *hash*

partitioning, *range partitioning* e *round-robin* são os mais elementares dentre os métodos de fragmentação horizontal e que tem em comum a escolha por uma chave de particionamento. No método baseado em *hash*, uma função de espalhamento é aplicada sobre a chave de particionamento para agrupar os dados em fragmentos. Quando o método baseado em *range* é aplicado, o conjunto de dados é agrupado por intervalos de valores assumidos pela chave de particionamento. A abordagem *round-robin* é um método aleatório para distribuir os dados nos fragmentos gerados.

O DB2 Design Advisor [Zilio et al., 2004] é uma ferramenta para determinar, de forma automática, grande parte dos aspectos do projeto físico de BDs, que incluem escolha de índices, criação de visões materializadas, fragmentação e agrupamento de dados. A ferramenta explora o otimizador do DB2 para avaliar e recomendar soluções para uma dada carga de trabalho. A carga de trabalho é obtida através de ferramentas adicionais como o *Dynamic Statement Cache* e o *Query Patroller*. O *Dynamic Statement Cache* armazena um histórico de planos utilizados por consultas executadas junto com as respectivas frequências de execução. O *Query Patroller* permite ao usuário classificar as consultas, dando prioridade a algumas delas através de uma fila de prioridades. Sua estratégia de fragmentação é essencialmente baseada na escolha de chaves de particionamento dentre um vasto número de recomendações que são exaustivamente avaliadas por simulações sobre o otimizador do DB2.

A ferramenta AutoAdmin [Agrawal et al., 2004] possui objetivos e um processo similar ao DB2 Database Advisor. O processo é composto de três passos principais: (i) seleção de chaves de particionamento candidatas, (ii) *Merging* e (iii) Enumeração. No primeiro passo cada consulta é analisada para identificar atributos candidatos a chaves de particionamento sobre cada uma das tabelas utilizadas pela consulta. No passo de *Merging*, os relacionamentos entre tabelas utilizadas são considerados para produzir novas recomendações a fim de determinar particionamentos horizontais, índices e visões materializadas adequadas. O objetivo do último passo (Enumeração) é produzir uma solução final das configurações candidatas levantadas pelos passos anteriores. Neste trabalho são aplicadas

algumas heurísticas para podar o conjunto de recomendações candidatas e reduzir o tempo de resposta da ferramenta. Embora as técnicas de fragmentação aplicadas pelo AutoAdmin e pelo DB Advisor possam ser aplicadas no projeto de BDs distribuídos, é importante ressaltar que elas foram inicialmente propostas para ambientes centralizados. Este fato torna justificável a escolha por métodos exaustivos.

O *AutoPart* [Papadomanolakis and Ailamaki, 2004] utiliza, fundamentalmente, particionamento de dados no projeto automatizado de esquemas para grandes bancos de dados científicos. O objetivo da ferramenta é evitar ao máximo redundâncias e a sobrecarga da utilização de índices. Segundo os autores, a utilização de uma boa estratégia de particionamento pode reduzir drasticamente a necessidade de indexação que tende a sobrecarregar SGBDs largamente distribuídos. O algoritmo utilizado efetua a fragmentação horizontal que é chamada de *particionamento categorizado* seguida pela fragmentação vertical de tabelas relacionais. No *particionamento categorizado* atributos que possuem uma pequena quantidade de valores capazes de identificar classes de objetos são considerados como chaves de particionamento para determinar fragmentos horizontais sobre as tabelas. Em seguida, as projeções das consultas são utilizadas para determinar a fragmentação vertical das relações. Este último ponto trata-se na verdade de uma fragmentação híbrida visto que o resultado da fragmentação horizontal já está sendo considerado. O esquema de fragmentação inicial gerado deve conter fragmentos completamente atômicos, isto é, os fragmentos que são recuperados por determinadas consultas devem conter apenas valores que são requisitados por elas. Isto torna possível que um fragmento nesta fase possa conter apenas um único atributo. A partir deste ponto, o algoritmo tenta melhorar o esquema de fragmentação inicial efetuando a junção de fragmentos para assim evitar transações distribuídas. O custo de cada composição é avaliado através de uma abordagem do tipo *what-if* que simula o desempenho do SQL Server sobre o esquema de fragmentação em avaliação. Embora o processo seja essencialmente exaustivo, é aplicada uma heurística para reduzir o número de recomendações avaliadas. Esta heurística se baseia em resultados de iterações prévias para gerar fragmentos com um número de atributos maior a

cada nova iteração. Embora um dos objetivos seja evitar a redundância, o AutoPart pode gerar fragmentos que não são completamente disjuntos pois utiliza replicação em alguns casos para evitar junções excessivas.

O grande problema de abordagens exaustivas é o tempo que as ferramentas levam para produzir as recomendações. Como cada expressão de consulta considerada na carga de trabalho deve ser compilada pelo otimizador, quanto maior for a carga de trabalho menos escalável será a ferramenta. No entanto, o método *what-if?* é ainda aplicado em trabalhos mais recentes como o Shinobi [Wu et al., 2011], pois corresponde a um método efetivo para identificar tendências na carga de trabalho. Acredita-se que soluções como estas são melhor aplicadas para SGBDs centralizados. Em razão de tratar-se de um problem NP-difícil [Kling et al., 2010], soluções heurísticas tornam-se mais atrativas para a fragmentação em bancos de dados distribuídos.

Na área de BDs largamente distribuídos, destaca-se a abordagem *Horticulture* [Pavlo et al., 2012] do projeto H-Store. Horticulture é uma ferramenta automática que particiona um banco de dados relacional *shared-nothing* paralelo e que tem por objetivo minimizar o número de transações distribuídas. Esta solução explora um conjunto de possíveis soluções, onde para cada tabela a ferramenta decide entre particioná-la horizontalmente e/ou replicá-la em todas as partições. Cada tupla é enviada a um determinado fragmento baseado nos valores da sua chave de particionamento usando particionamento por *hash* ou *range*. A replicação é utilizada para melhorar o desempenho de consultas sobre dados que são pouco atualizados. Holticulture aplica uma técnica baseada em *Large-Neighborhood Search* (LNS) que explora um espaço de soluções possíveis para encontrar uma solução de particionamento adequada. O LNS compara soluções em potencial com um modelo de custo que estima a qualidade de um particionamento para uma dada carga de trabalho. Esta estimativa estabelece uma proporção entre o número de transações distribuídas e o grau de uniformidade da sua carga distribuída entre as partições para determinar o desempenho de uma carga de trabalho com relação à uma solução em potencial.

MESA [Nehme and Bruno, 2011] é uma solução similar ao Horticulture que baseia-

se em uma integração com o otimizador do *Microsoft SQL Server 2008 Parallel Data Warehouse*. Apesar de caracterizar-se como uma abordagem do tipo *what-if*, a exaustão do processo é evitada pela heurística *branch-and-bound* na redução de soluções candidatas. Diferentemente do Horticulture, MESA é específica para cargas de trabalho OLAP.

3.1.1.2 Abordagens baseadas em Predicados

No modelo relacional, a fragmentação horizontal baseada na análise de predicados é classificada em *fragmentação primária* e *fragmentação derivada* [Ceri and Pelagatti, 1984]. Na *fragmentação primária*, uma relação é particionada usando apenas predicados definidos sobre ela. Já a *fragmentação derivada* corresponde ao particionamento de uma relação de acordo com o resultado de predicados definidos sobre outra relação. Na abordagem clássica definida por [Ceri and Pelagatti, 1984], a fragmentação primária de uma relação é definida pelo conjunto de predicados *minterm* sobre ela. Um predicado *minterm* corresponde à conjunção de predicados simples. Assim, dada uma relação $R(a_1, ..., a_n)$, onde a_i é um atributo definido sobre o domínio D_i, um predicado simples p_j sobre R é definido por $p_j := a_i\ \theta\ valor$, $\theta \in \{=, \neq, >, \leqslant, <, \geqslant\}$ e $valor \in D_i$. Dado um conjunto $Pr_i = \{p_{i1}, ..., p_{im}\}$ de predicados simples de uma relação R_i, o conjunto de predicados *minterm* $M_i = \{m_{i1}, ..., m_{iz}\}$ é definido por $M_i = \{m_{ij} | m_{ij} = \bigwedge_{p_{ik}} p^*_{ik}\}$, $1 \leqslant k \leqslant m$, $1 \leqslant j \leqslant z$. Assim, cada fragmento primário de uma relação corresponde a um conjunto de tuplas que atendem a um dos seus predicados *minterm*. Segundo [Ceri and Pelagatti, 1984], a fragmentação primária deve ainda atender ao critério de completude e minimalidade dos fragmentos. Um fragmento é dito *completo* se cada tupla tem a mesma probabilidade de ser acessada dentro do fragmento. Em outras palavras, a propriedade de completude garante o acesso uniforme sobre as tuplas de um fragmento o que, intuitivamente, pode indicar que elas são acessadas em conjunto por uma ou mais transações da carga de trabalho. A minimalidade do fragmento é uma propriedade que estabelece que o conjunto de valores resultante de cada predicado simples em Pr_i deve contribuir para determinar um conjunto de fragmentos disjuntos de R. Na *fragmentação derivada*

existe um relacionamento de junção de igualdade entre uma relação proprietária e uma relação membro. A fragmentação derivada aplica uma semi-junção entre as duas relações, sendo que a relação membro é então fragmentada de acordo com uma seleção aplicada sobre a relação proprietária. Os conceitos e algoritmos propostos em [Ceri and Pelagatti, 1984] são a base dos algoritmos de fragmentação horizontal que os sucederam [Noaman and Barker, 1999, Bellatreche and Boukhalfa, 2005, Wehrle et al., 2005].

A técnica de fragmentaçao horizontal baseada em predicados *minterm* foi recentemente proposta para o particionamento de dados RDF [Galárraga et al., 2014]. Neste trabalho, cada fragmento corresponde a um *minterm*, que posteriormente é alocado em conjunto com fragmentos relacionados por este predicado. Outra abordagem focada na fragmentação horizontal para RDF é o trabalho de [Yang and Wu, 2013]. O foco deste trabalho é extrair padrões de consultas RDF para identificar padrões de triplas que conferem de alta seletividade em predicados. Assim, valores de sujeito ou objeto destas triplas são usadas como chave de particionamento do grafo RDF.

Diversas técnicas de fragmentação para dados XML também foram propostas com base nas soluções clássicas para o modelo relacional [Kling et al., 2010]. No entanto, a avaliação de predicados neste contexto se dá pela análise de caminhos sobre a árvore de documentos XML. A técnica introduzida por [Ma and Schewe, 2003] define como os esquemas de fragmentação devem ser estruturados sobre um documento XML. Neste contexto, a fragmentação horizontal é definida pelo agrupamento de elementos de acordo com um critério de seleção possibilitando a geração de fragmentos não-homogêneos. Os autores sugerem a aplicação do método proposto em [Ceri and Pelagatti, 1984] para identificar os critérios de seleção e efetuar o particionamento do documento. O conceito de fragmentação horizontal aplicado por este trabalho difere de outras soluções, que aplicam os critérios de seleção sobre uma coleção de documentos XML para gerar conjuntos de documentos homogêneos após o particionamento.

Os autores de [Gertz and Bremer, 2003] também definem estratégias de fragmentação sobre repositórios XML. Neste caso os fragmentos são definidos baseando-se em

uma linguagem de caminho chamada XF derivada do XPath. Os fragmentos são obtidos aplicando-se uma expressão do tipo XF sobre uma árvore que representa os dados XML. O método equivale a uma abordagem de fragmentação híbrida em que a fragmentação horizontal e vertical são consideradas ao mesmo tempo. Esta solução provê expressões de exclusão de alguns caminhos para garantir a coerência dos fragmentos e eliminar caminhos contraditórios, bem como garantir a disjunção entre os fragmentos gerados. Uma solução bastante similar foi introduzida por [Bonifati et al., 2004], que adicionalmente define como alocar os fragmentos em redes p2p. A proposta de [Abiteboul et al., 2008] armazena uma coleção de dados XML em uma rede p2p baseada em DHT. Cada documento XML é associado a uma palavra-chave que atua como uma chave para a DHT. O conjunto de documentos é então particionado horizontalmente por intervalos de palavras-chave.

O método proposto em [Andrade et al., 2006, Figueiredo et al., 2010] propõe aplicar a fragmentação sobre uma coleção homogênea de documentos XML. Neste trabalho, os autores definem a estrutura de fragmentos horizontais, verticais e híbridos. Fragmentos são definidos através de operações da álgebra TLC (*Tree Logical Class*) [Paparizos et al., 2004]. Isto permite que consultas sejam decompostas sobre os fragmentos quando se utiliza a TLC para representar consultas XQuery. O trabalho sugere a adaptação das técnicas bem sucedidas do modelo relacional para o modelo XML. Na realidade, o foco do trabalho está na definição da estrutura dos fragmentos, na decomposição de consultas e em uma proposta de fragmentação virtual para o contexto de aplicações OLAP. Em resumo, a abordagem replica completamente um BD sobre um conjunto de nodos e, em seguida, divide cada consulta em sub-consultas de intervalo de acordo com seus predicados. Assim, cada BD recebe uma sub-consulta e é forçado a processar um subconjunto diferente dos itens de dados. Cada subconjunto desses é chamado de partição virtual.

Em resumo, as abordagens propostas para XML baseiam-se em métodos de fragmentação clássicos para dividir um BD XML em uma coleção de fragmentos XML. Um fragmento XML é definido por uma expressão de caminho ([Gertz and Bremer, 2003], [Bonifati et al., 2004]), ou por um operador de álgebra XML ([Andrade et al., 2006]). Além

disto, a fragmentação é executada sobre um único documento XML [Ma and Schewe, 2003], ou sobre uma coleção de documentos homogêneos [Andrade et al., 2006]. Existem outras abordagens propostas para XML que definem estratégias de fragmentação baseadas em restrições estruturais de esquemas XML, como profundidade e largura da árvore [Bonifati and Cuzzocrea, 2007], ou ainda restrições envolvendo o número de fragmentos gerados [Cuzzocrea et al., 2009]. Considera-se que estas abordagens são ortogonais ao problema de fragmentação de dados. Existem ainda soluções não empíricas como [Khan and Hoque, 2010], que baseiam-se na capacidade e desempenho de recursos disponíveis na rede. No entanto, em ambientes de alta volatilidade e *shared-nothing* como sistemas na *nuvem*, estas informações podem não estar disponíveis ou podem mudar a todo instante.

3.1.2 Fragmentação Vertical e Híbrida

O princípio da fragmentação vertical é a identificação de itens de dados acessados em conjunto pela carga de trabalho para definir como o esquema do BD deve ser fragmentado [Ozsu and Valduriez, 2011]. Assim, pode-se observar que a fragmentação vertical é aplicada sobre um esquema de BD enquanto a fragmentação horizontal é aplicada sobre todas as instâncias de itens de dados. Assim, o volume de dados a ser analisado na fragmentação vertical é menor que na fragmentação horizontal. No modelo relacional, uma relação é dividida em fragmentos que contém um sub-conjunto de seus atributos. As abordagens exaustivas propostas para este modelo [Agrawal et al., 2004,Papadomanolakis and Ailamaki, 2004] simulam as possíveis soluções de agrupamento sobre o modelo de custo do otimizador do BD para determinar o melhor agrupamento. As soluções tradicionais que utilizam heurísticas são descritas em [Navathe et al., 1984] e em [Navathe and Ra, 1989]. Nelas, um algoritmo define como os atributos de uma relação devem ser agrupados baseado na afinidade entre eles. Na solução inicial, uma relação era fragmentada de acordo com estas afinidades pelo algoritmo *Bond Energy* [McCormick et al., 1972]. Na sequência uma solução com menor complexidade computacional foi proposta em [Navathe and Ra, 1989]. Nela, um grafo de afinidades é criado para relacionar os atributos de uma tabela,

onde as arestas denotam o grau de afinidade entre pares de atributos. Esta afinidade é dada pela soma das frequências de transações que acessam um par de atributos em conjunto. Os fragmentos são gerados baseados na identificação de ciclos de afinidade neste grafo. A ideia é que afinidades altas devam ser encontradas entre atributos de um mesmo fragmento, e afinidades mais baixas entre atributos de fragmentos diferentes.

Existem algumas propostas que baseiam-se em grafos para determinar a fragmentação de BDs relacionais. Em geral, estas propostas aplicam uma fragmentação híbrida pois, diferentemente do que foi proposto em [Navathe and Ra, 1989], não somente as relações entre itens do esquema do BD são consideradas, mas também a relação entre as instâncias de itens de dados. Um exemplo deste tipo de abordagem é a proposta da ferramenta Schism [Curino et al., 2010]. A principal ideia deste trabalho é representar o BD e sua carga de trabalho como um grafo, onde as tuplas são representadas por nós e transações por arestas conectando tuplas acessadas em conjunto em uma transação. Um algoritmo de particionamento de grafos é aplicado para encontrar partições balanceadas e que minimizam o peso das arestas de corte. Intuitivamente, o peso das arestas de corte está diretamente associado à quantidade de execuções de transações distribuídas. Pelo fato do Schism dedicar-se a dar suporte a cargas de trabalho do tipo OLTP, o número de transações distribuídas deve ser minimizado.

No modelo XML, os fragmentos verticais de um documento XML correspondem a subárvores contidas nele. A grande maioria dos trabalhos neste contexto [Ma and Schewe, 2003, Gertz and Bremer, 2003, Andrade et al., 2006, Bonifati et al., 2004] recomendam soluções similares às propostas definidas em [Navathe et al., 1984] ou em [Navathe and Ra, 1989]. Um grupo de trabalhos baseia-se na representação das afinidades entre elementos de documentos XML [Bordawekar and Shmueli, 2004, Kanne and Moerkotte, 2006, Bordawekar and Shmueli, 2008, Shnaiderman et al., 2008, Shnaiderman and Shmueli, 2009]. Diferentemente do grafo de afinidades completo proposto por [Navathe and Ra, 1989], estes trabalhos consideram uma representação simplificada que incluem as afinidades sobre conexões pai-filho dos próprios documentos juntamente com as conexões entre elementos

irmãos. A motivação desta estratégia é respeitar a estrutura dos documentos e reduzir a quantidade de conexões de afinidades para fins de avaliação. No entanto, algumas relações de afinidades são desconsideradas e são, consequentemente, perdidas na geração dos fragmentos. Outra desvantagem destas soluções e da ferramenta Schism está relacionada ao volume de dados a ser avaliado. Além de dificultar a representação, o volume de dados pode tornar o processo oneroso quando grandes BDs são considerados.

Observa-se que estas abordagens baseadas em heurísticas de afinidades são diretamente associadas a problemas de particionamento em grafos. Grande parte das soluções propostas para o particionamento vertical e híbrido de dados RDF são baseadas em métodos de particionamento em grafos. Em especial, o método proposto por [Jiewen Huang, 2011], estabelece o particionamento de grafos RDF e o processamento de consultas usando o *framework Hadoop*. O objetivo do particionamento é facilitar que as consultas possam ser totalmente paralelizadas sem que haja necessidade de troca de dados entre partições durante a execução. O processo de particionamento é dividido em 2 fases. Na primeira fase, os vértices do grafo RDF são particionados através do particionar de grafos METIS, dada um número de partições que se deseja gerar. No segundo passo, cada tripla (aresta) é avaliada com relação aos vértices presentes em uma partição. Este trabalho propõe a garantia de *n-hops* que seria a quantidade de saltos (arestas) a partir de cada um desses vértices que deve ser armazenada na partição. Isso em muitos casos envolve a replicação de dados (vértices) entre as partições. Essa análise é proposta tanto sobre o grafo direcionado quanto para não-direcionado. Vértices que possuem muitas arestas relacionadas são eliminadas nessas 2 fases e só incluídas no particionamento final onde são atribuídas a partição que apresenta o maios numero de vértices aos quais ele está relacionado.

Na seção 3.3 são apresentadas algumas heurísticas e algoritmos propostos para o problema geral de particionamento em grafos visto que eles podem ser diretamente aplicados ao contexto da fragmentação de BDs.

3.2 Alocação de Dados

Considere um conjunto de fragmento $F = \{f_1, ..., f_n\}$ e um sistema distribuído de servidores $S = \{s_1, ..., s_m\}$ sobre o qual um conjunto de transações $T = \{t1, ..., t_t\}$ é executado. O problema da alocação de dados envolve encontrar uma distribuição ótima de F sobre S [Ozsu and Valduriez, 2011]. A qualificação de uma distribuição ótima está relacionada à minimização de custos de armazenamento e acesso dos fragmentos em uma dada distribuição, bem como à maximização do desempenho das transações da carga de trabalho. O desempenho das transações está diretamente associado ao relacionamento entre fragmentos e ao custo de comunicação entre eles em uma determinada distribuição. Obter soluções ótimas não é computacionalmente viável para grandes dimensões do problema que envolvem grandes quantidades de fragmentos e nós [Ceri and Pelagatti, 1982]. Assim, o problema de alocação vem sendo considerado com um problema de otimização associado a soluções heurísticas.

Soluções de alocação são fortemente dependentes de restrições impostas pelo ambiente de processamento de uma carga de trabalho. Estas restrições podem estar associadas à capacidade individual de armazenamento e processamento dos servidores da rede, assim como dos canais de comunicação entre os servidores. Uma solução de alocação adequada deve considerar estas restrições juntamente com as necessidades de comunicação entre servidores para minimizar o tempo de resposta das transações e maximizar a vazão do sistema.

Soluções que utilizam o particionamento baseado em chaves como [Zilio, 1998], [Zilio et al., 2004], [Agrawal et al., 2004] e [Papadomanolakis and Ailamaki, 2004], alocam seus fragmentos agrupando-os por estratégias de *hash*, *range* ou *round-robin*. Este fato mostra a forte dependência da estratégia de alocação com a utilizada para fragmentar os dados. Os autores de [Bellatreche and Benkrid, 2009] apresentam algumas abordagens que tratam a fragmentação e a alocação de forma isolada, e demonstram sua ineficiência através de uma avaliação experimental em [Bellatreche et al., 2011]. Os autores defendem que não

deve haver interdependência entre os processos de fragmentação e alocação, uma vez que o esquema de fragmentação é a entrada para o processo de alocação e que ambos os processos visam otimizar o mesmo conjunto de consultas. Logo, o processo de alocação deve apenas adicionar a consideração das restrições impostas pelo ambiente de armazenamento e processamento.

Embora muitas abordagens de fragmentação listadas pela Tabela 3.1 não relacionam soluções de alocação, muitas consideram um fragmento como unidade de alocação [Pavlo et al., 2012, Nehme and Bruno, 2011, Figueiredo et al., 2010, Curino et al., 2010], enquanto outras recomendam heurísticas tradicionais de alocação [Ceri and Pelagatti, 1984, Gertz and Bremer, 2003, Ma and Schewe, 2003] baseadas essencialmente nas restrições de armazenamento dos servidores da rede. Os métodos de alocação propostos em [Abiteboul et al., 2008] e [Bonifati et al., 2004] são baseados nos métodos de distribuição de redes p2p estruturadas. Em [Bonifati et al., 2004] é escolhido um identificador único para cada fragmento para a posterior alocação dos mesmos através da aplicação de uma função de espalhamento da DHT subjacente. Similarmente, em [Abiteboul et al., 2008] palavras-chave que representam os fragmentos são utilizadas sobre uma rede p2p estruturada para endereçar o conteúdo.

Adicionalmente ao seu objetivo principal, soluções para alocação de fragmentos podem ser classificadas em redundantes ou não-redundantes, balanceadas ou não-balanceadas, estáticas ou dinâmicas e centralizadas ou descentralizadas [Hauglid et al., 2010]. Diversos trabalhos propõem a replicação de alguns fragmentos durante a alocação para melhoria do desempenho de transações e balanceamento de cargas de trabalho não-uniformes. Além disto, as soluções podem ser estáticas ou dinâmicas com relação a alterações na carga de trabalho ou na composição do ambiente de processamento. Esta característica está diretamente associada à classificação de métodos em centralizados ou descentralizados. Por exemplo, algumas soluções dinâmicas estão associadas aos processos de fragmentação e alocação incrementais que são desempenhadas individualmente pelos servidores da rede.

Diante dos trabalhos listados pela Tabela 3.1 e apresentados por este capítulo, constata-

se a ausência de soluções que tratam toda a extensão do problema de distribuição de dados para o contexto de sistemas de banco de dados transacionais na *nuvem*. Além de prover estratégias de alocação e técnicas de fragmentação horizontais e verticais apropriadas, soluções para atuar no contexto da *nuvem* devem dar suporte a processos de distribuição de dados dinâmicos sobre um modelo de dados flexível para uma variedade de tipos de aplicações. A proposta apresentada pelo Capítulo 4 desta tese visa solucionar estes problemas.

3.3 Particionamento de Grafos

Considere que um banco de dados e sua respectiva carga de trabalho possam ser representados por um grafo não-direcionado definido por $G = (V, E, c)$, onde (i) V é o conjunto de nós que representam itens de dados, (ii) E é o conjunto de arestas que relacionam pares de nós acessados em conjunto pela carga de trabalho, e (iii) $c : E \rightarrow \mathbb{N}$ define o custo de cada aresta que representa a quantidade de vezes que um par de nós é acessado em conjunto. A fragmentação em banco de dados pode ser definida como um problema de corte em grafos com a finalidade de gerar um esquema de fragmentação $F = \{f_1, f_2, ..., f_k\}$, tal que $f_i \subseteq V$, $\bigcup_{i=1}^{k}(f_i) = V$ e $f_i \cap f_j = \oslash$ para todo $i \neq j$.

Embora existam diferentes modelos associados ao problema de corte em grafos, esta seção dedica-se ao problema do corte mínimo. O custo de um corte em G definido por um esquema de fragmentação F é dado por $custo(F) = \sum_{v_1 \in f_i, v_2 \in f_j} c(v_1, v_2)$ para todo $i \neq j$. Intuitivamente, ao minimizar o custo do corte minimiza-se também o número de execuções de transações distribuídas, uma vez que itens de dados frequentemente acessados em conjunto serão mantidos em um mesmo fragmento. Neste capítulo são descritos modelos de corte mínimo para grafos, bem como algoritmos de aproximação para os problemas relacionados.

3.3.0.1 Modelos de Corte Mínimo

O problema base de corte mínimo de grafos é conhecido como *s-t Minimum Cut*, que visa particionar um grafo em dois subconjuntos que separam um nó origem ($s \in V$) e um nó destino ($t \in V$) de forma que o custo do corte seja mínimo. O algoritmo *Ford-Fulkerson* [G., 1954] é capaz de determinar o corte mínimo computando o fluxo máximo entre s e t em uma rede. Através do teorema *max-flow min-cut*, Ford and Fulkerson provaram que o valor do fluxo máximo é igual ao valor do corte mínimo [Papadimitriou and Steiglitz, 1998]. Este problema é uma generalização dos problemas de multi-corte, que por definição tratam-se de problemas NP-difíceis para $k \geq 3$ onde $k = |F|$.

Existem quatro classificações principais para problemas de multi-corte mínimo: *Minimum Multicut*, *k-Multicut*, *Multiway cut* e *k-cut*. O objetivo do problema *Minimum Multicut* segue o mesmo princípio do problema *s-t Minimum Cut*. Seja um grafo G e $P = \{(s_1, t_1), ..., (s_k, t_k)\}$ um conjunto específico de pares de nodos origem-destino, o problema é determinar um conjunto de arestas cuja remoção separa cada um dos pares de nodos em P, gerando um esquema de fragmentação F onde o valor de $custo(F)$ é minimizado. Um problema similar é o *Multiway cut*, cuja a finalidade é encontrar um corte mínimo que separa um conjunto de terminais $S = \{(s_1), ..., (s_k)\} \subseteq V$.

O problema *k-Multicut* pode ser reduzido ao problema de *Minimum Multicut*, porém neste caso o número de fragmentos gerados deve ser igual ou superior a k, ou seja, $|F| \geq k$. O problema que requer encontrar um conjunto de arestas cuja a remoção deixa k fragmentos é denominado *k-cut* ou *k-way*. O objetivo é encontrar o valor de k ($1 < k \leqslant |V|$) para o qual o valor de $custo(F)$ é mínimo.

Dentre os modelos de corte mínimo apresentados, *k-cut* é o que mais se aproxima ao problema de fragmentação em bancos de dados. Neste contexto, deseja-se obter um esquema de fragmentação em que $|F|$ é inicialmente desconhecido. No entanto, existem outras restrições que devem ser consideradas, como a capacidade de armazenamento de cada fragmento. Existem diversas especializações dos modelos de corte mínimo que consideram restrições ao peso imposto pelos nodos em um fragmento. Em [Galbiati, 2011],

o problema *s-t Minimum Cut* é estendido para considerar um limiar sobre o tamanho total ocupado por um dos fragmentos gerados. Para modelos de multi-corte, Abou-Rjeili e Karypis [Abou-Rjeili and Karypis, 2006] consideram um grafo G e uma função $w : V \rightarrow \mathbb{N}$ que define o peso para cada vértice do grafo e um fator de desbalanceamento ζ. O peso total de um fragmento $f_i \in F$ é dado por $peso(f_i) = \sum_{v_j \in f_i} w(v_j)$. O objetivo é encontrar um esquema de fragmentação F em que o valor de $custo(F)$ é minimizado, ao mesmo tempo em que $peso(f_i) \leq peso(f_j) + \zeta$ para todo $f_i \neq f_j$.

3.3.0.2 Técnicas de Particionamento

Para resolver o problema do corte mínimo, um grafo pode ser particionado por bisseções recursivas ou por uma estratégia de particionamento em múltiplos níveis. No paradigma de bisseções recursivas, um grafo é recursivamente dividido em duas partições até que o número de fragmentos desejado seja obtido. Em cada passo, são aplicadas heurísticas para refinar as bisseções e diminuir o custo do corte. No entanto, a execução destes refinamentos iterativos tendem a deteriorar o resultado, gerando soluções pouco aproximadas da solução ótima uma vez que não possuem uma visão global do problema [Karypis and Kumar, 1999, Aykanat et al., 2008]. Diversos trabalhos provam que o particionamento em múltiplos níveis produz melhores resultados que bisseções recursivas para diversos tipos de aplicação do problema [Karypis and Kumar, 1999, Çatalyürek and Aykanat, 1999, Abou-Rjeili and Karypis, 2006].

O particionamento em múltiplos níveis obtém uma sequência de aproximações sucessivas do grafo original a fim de reduzir o problema e, em seguida, refinar a solução para encontrar a solução mais aproximada. O processo compreende 3 etapas. A fase de *coarsening* executa uma sequência de aproximações (reduções) sucessivas do grafo. Em seguida, na fase de *particionamento inicial* o grafo reduzido é particionado. Na etapa final, a solução de particionamento encontrada é então projetada sequencialmente para níveis de granularidade mais fina do grafo, sendo que em cada nível um algoritmo de refinamento iterativo é executado para melhorar a qualidade do particionamento.

Na fase de *coarsening*, pares de vértices adjacentes são sequencialmente combinados por um método de *matching*. Dentre estes métodos destaca-se o método *randômico* e o *heavy edge*. No método randômico a fusão ocorre entre nodos adjacentes que se conectam aos mesmos nodos [Abou-Rjeili and Karypis, 2006]. Diferentemente, o método *heavy edge* seleciona as arestas de maior peso para executar a fusão dos nodos conectados por elas [Karypis and Kumar, 1998]. Na fase de *particionamento inicial*, algoritmos de bisseções podem ser executados. A seguir, são apresentados alguns destes algoritmos bem como algoritmos de refinamentos aplicados na última fase do particionamento em múltiplos níveis.

Existem diversos algoritmos que fornecem soluções aproximadas para problemas de corte mínimo através de bisseções recursivas. A Tabela 3.2 apresenta uma lista dos principais algoritmos. Todos apresentam resultados aproximados, com exceção do trabalho de Goldschmidt e Hochbaum [Goldschmidt and Hochbaum, 1988] que constitui uma solução determinística para o problema de *k-Multicut*, isto é, quando k é fixo. Dentre as soluções para o problema de *k-cut*, destaca-se o algoritmo com maior aproximação definido em [Saran and Vazirani, 1995], cuja complexidade de execução é baseada no processamento de m fluxos máximos. Este trabalho estende o problema para considerar a restrição de balanceamento da quantidade de nós nos fragmentos gerados. Entretanto, trata-se de uma solução bastante primitiva e pouco aproximada para o problema de k-cut balanceado se comparada a uma solução equivalente proposta em [Guttmann-Beck and Hassin, 2000]. O algoritmo proposto para resolver o problema do *s-t Cut* em [Galbiati, 2011], considera pesos atribuídos aos nodos do grafo e um limiar B que determina o peso máximo para a partição s a ser gerada. A solução randômica apresentada pelos autores apresenta uma aproximação de $(1+\frac{\epsilon.B}{log|V|})$. Uma outra solução também é fornecida, porém neste caso a aproximação se dá sobre o custo do corte sobre o esquema de fragmentação F gerado e corresponde a $custo(F) \leqslant \frac{1}{1-\gamma} custo(F^*)$, onde $0 < \gamma < 1$ e F^* refere-se ao esquema de fragmentação ótimo.

O surgimento da classe de algoritmos de refinamento foi iniciada pelos algoritmos de

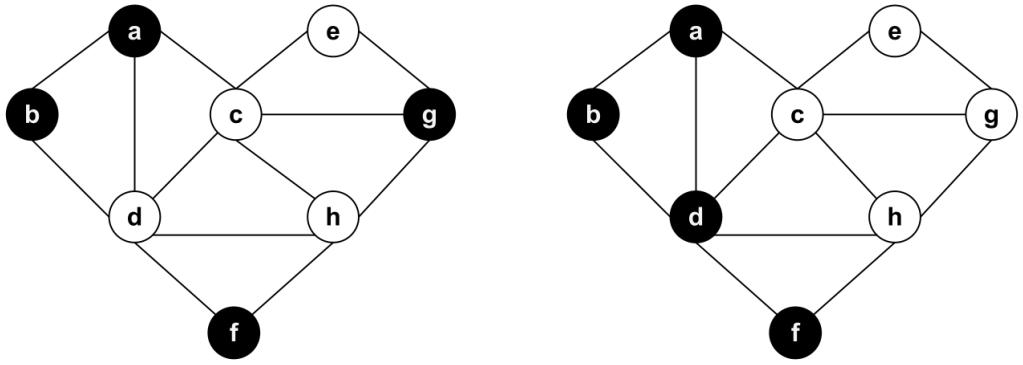
Algoritmo	Modelo	Aproximação	Complexidade Execução
[Goldschmidt and Hochbaum, 1988]	k-Multicut	determinístico	$\Theta(\|V\|^{k^2})$
[Saran and Vazirani, 1995]	k-cut	$(2-2/k)$	$\Theta(m\|V\|^2 + \|V\|^{3+\epsilon})$
[Saran and Vazirani, 1995]	k-cut balanceado	$\|V\|(\frac{k-1}{k})$	$\Theta(m\|V\|^2 + \|V\|^{3+\epsilon})$
[Guttmann-Beck and Hassin, 2000]	k-cut balanceado	$3k$	$\Theta(\|V\|^{k+1})$
[Galbiati, 2011]	s-t Cut *B threshold*	$(1 + \frac{\epsilon.B}{log\|V\|})$	$\Theta(\|V\|^{\|V\|-1})$

Tabela 3.2: Algoritmos de bisseções recursivas

Kernighan-Lin e Fiduccia-Mattheyses (KL e FM). Dado um grafo com um particionamento preliminar, o problema é melhorar a qualidade da partição enquanto mantem a restrição de balanceamento referente a quantidade de nós em cada fragmento. Portanto, dada uma bisseção de um grafo que separa os vértices nos conjuntos V' e V'', deseja-se encontrar dois sub-conjuntos de tamanho igual, R' e R'' a partir de V' e V'', respectivamente, de tal forma que substituindo-se nós entre V' e V'' diminui-se o custo do corte do grafo. Este processo pode ser efetuado repetidas vezes até que não seja mais possível diminuir o custo do corte. Os algoritmos KLFM trabalham sobre heurísticas para encontrar os sub-conjuntos R' e R'' próximos ao ótimo.

A Figura 3.3 exemplifica o processo efetuado pelos algoritmos de refinamento do tipo KLFM. O particionamento preliminar apresentado pela Figura 3.3(a) divide o grafo em dois conjuntos de vértices V' e V'', V' representado pelos nós com preenchimento e V'' pelos nós sem preenchimento. Assuma que o custo do corte é dado pela quantidade de arestas entre V' e V''. No caso do particionamento preliminar, o custo do corte é igual a 8 arestas. Um algoritmo do tipo KLFM efetua a troca dos nós d e g entre as partições e assim reduz o custo do corte para 4 arestas no grafo refinado da Figura 3.3(b).

O algoritmo KL [Kernighan and Lin, 1970] efetua trocas de todos os vértices entre as partições V' e V''. Durante cada passo, o algoritmo tenta encontrar um par de vértices $a \in V'$ e $b \in V''$ cuja a troca entre as partições pode reduzir o custo do corte. Para determinar a viabilidade da troca, alguns custos são previamente calculados. O custo externo de um nó $a \in V'$ é dado por $E(a) = \sum_{j \in V''} c(a, j)$, onde $c(a, j)$ corresponde ao custo do corte que separa a e j. O custo interno de um nó é dados por $I(a) = \sum_{j \in V'} c(a, j)$ e a diferença entre eles é dado por $D(a) = E(a) - I(a)$. Estas mesmas medidas são



(a) Particionamento Preliminar (b) Particionamento Refinado

Figura 3.3: Refinamento KLFM

igualmente calculadas para b. Por fim, o ganho em se trocar os vértices a e b é dado por $g(a,b) = D(a) + D(b) - 2.c(a,b)$. Neste caso, escolhe-se um par de vértices a e b cujo valor de $g(a,b)$ seja máximo. A troca é efetuada e o mesmo processo é repetido para os demais pares de V' e V'' até que todos os pares sejam movidos. Em cada troca, contabiliza-se o custo geral do corte do grafo para que no final o corte mínimo seja identificado. Na finalização deste passo, a composição do grafo correspondente ao corte mínimo é recuperada. Após finalizada uma execução do algoritmo KL, outra execução pode então ser iniciada no contexto da etapa de refinamento, em que a bisseção resultante é utilizada como entrada. O passo inicial do algoritmo possui complexidade de tempo $\Theta(n^2)$, onde n é a quantidade de nós. A busca pelos melhores ganhos na troca de pares custa $\Theta(n^2 log\ n)$, conforme demonstrado em [Kernighan and Lin, 1970].

O algoritmo *Fiduccia-Mattheyses* (FM) [Fiduccia and Mattheyses, 1982] é considerado um melhoramento do KL. Para casos gerais, o FM é capaz de reduzir o tempo de execução utilizando estruturas de dados mais apropriadas. O algoritmo apresenta tempo de execução de $\Theta(nlogn + |E|)$ e apresenta a mesma efetividade do algoritmo KL. No entanto, para casos específicos o FM pode gerar um corte maior e desempenho inferior que o apresentado pelo KL. Diversos algoritmos foram derivados das abordagens de KL e FM, dentre eles destaca-se uma combinação das abordagens conhecida como KLFM [Hauck and Borriello, 1995].

Existem diversas aplicações que se beneficiam de técnicas de particionamento mas que estão sujeitas a alterações de distribuição de carga. Para o contexto de BDs distribuídos, uma estratégia de particionamento do tipo *k-cut* balanceado é ideal para gerar um esquema de distribuição de dados com balanceamento da carga de trabalho entre os servidores da rede [Schloegel et al., 2000]. No entanto, variações na carga de trabalho poderão deixar o esquema de distribuição desbalanceado e reduzir a qualidade do corte inicial. Neste contexto, um procedimento de reparticionamento deve ser aplicado periodicamente para manter a qualidade do particionamento bem como o balanceamento da carga entre as partições.

Segundo Catalyurek U. et. al. [V. et al., 2009], abordagens dinâmicas baseadas em corte mínimo em grafos podem ser classificados em três categorias principais: *scratch-remap*, *incremental* e *reparticionamento*. Na abordagem *scratch-remap*, o grafo é particionado do início, isto é, sem considerar o particionamento inicial como referência. Em contrapartida, a estratégia *incremental* preza por prover o balanceamento de dados sobre o particionamento inicial. Nesta categoria, o balanceamento é considerado juntamente com o objetivo de minimizar ou o custo da migração de dados ou o custo do corte do grafo de dados. Por fim, na abordagem de *reparticionamento* ambos os objetivos são considerados ao mesmo tempo.

No contexto de banco de dados, grande parte das soluções que aderem as categorias *incremental* ou de *reparticionamento* são baseadas em replicação [Kossmann, 2000]. Uma solução recente proposta em [Yang et al., 2012] aplica replicações em fragmentos para acomodar o acesso não-uniforme a dados e gera fragmentos complementares para ajustar-se a alterações na carga de trabalho. Abordagens similares são apresentadas em outros trabalho recentes como [Pujol et al., 2010] e em [Brocheler et al., 2010]. Porém a geração de novos fragmentos e a replicação destes pode se tornar prejudicial para o desempenho do sistema, uma vez que ocorre um aumento significativo de volume do BD e de dados replicados. Existem algumas outras soluções que preocupam-se em monitorar e identificar a necessidade de reavaliar o esquema de particionamento de um BD, mas acabam por execu-

tar um processo *scratch-remap*. Um exemplo é a ferramenta PIXSAR [Shnaiderman et al., 2008] que atualiza periodicamente as afinidades sobre conexões de elementos pai-filho e irmãos de documentos XML de acordo com as transações que estão sendo executadas. Quando detecta-se que a qualidade do particionamento foi prejudicada, o algoritmo de particionamento proposto em [Bordawekar and Shmueli, 2008] é reaplicado sobre toda a coleção de documentos XML. Certamente, o custo de iniciar um novo processo de particionamento é impraticável em BDs que processam um grande volume de dados [Hauglid et al., 2010].

Como mencionado anteriormente, o problema da distribuição de dados pode ser endereçado com um problema de otimização de corte mínimo em grafos. Neste contexto, acredita-se que soluções de reparticionamento baseadas em refinamentos de esquemas de particionamento anteriores sejam mais adequadas uma vez que efetuam um processo incremental e assim reduzem a complexidade do problema [Aykanat et al., 2007] [Hendrickson et al., 1997] [Schloegel et al., 2000].

CAPÍTULO 4

ABORDAGEM PARA O PARTICIONAMENTO DE DADOS NA NUVEM BASEADA EM RELAÇÕES DE AFINIDADES

Este capítulo apresenta a solução proposta por esta tese para o particionamento de dados em repositórios em nuvem. A definição desta proposta está baseada no modelo RDF devido ao seu modelo genérico capaz de expressar outros modelos de dados como o XML. Dados RDF são definidos como um conjunto de triplas compostas de um sujeito, propriedade e objeto (s,p,o). Assume-se a existência dos conjuntos infinitos, disjuntos e pareados $\mathcal{U}$ e $\mathcal{L}$, onde $\mathcal{U}$ corresponde a URIs para representar recursos Web, e $\mathcal{L}$ correspondem a valores literais. Assim, uma tripla (s, p, o) $\in \mathcal{U} \times \mathcal{U} \times \{\mathcal{U} \cup \mathcal{L}\}$. RDF aplica um modelo de dados no qual triplas são relacionadas na forma de um grafo direcionado.

SPARQL é uma linguagem de consulta recomendada pela W3C sobre repositórios RDF. O núcleo da sintaxe SPARQL é baseada em um conjunto de padrões de triplas semelhante a triplas RDF, exceto que sujeitos, propriedades e objetos podem ser representadas por variáveis. SPARQL é uma linguagem de consulta cujo processamento é definido como um problema de casamento de subgrafos sobre um grafo RDF. Neste trabalho, *padrões de grafos* são definidos para representar o fragmento conjuntivo de consultas SPARQL. Antes de introduzir a definição de *padrão de grafo*, assume-se a existência de um conjunto $\mathcal{V}$ de variáveis, disjunto dos conjuntos $\mathcal{U}$ e $\mathcal{L}$. Variáveis em $\mathcal{V}$ são representadas pelo uso de pontos de interrogação (?).

Definition 4.0.1 *(Padrão de Grafo): Um padrão de grafo é definido por* $G = (V, E, r)$*, onde: (1)* $V \subseteq \{\mathcal{V} \cup \mathcal{U} \cup \mathcal{L}\}$*; (2)* $E \subseteq V \times \mathcal{U} \times V$*, sendo que para cada aresta (*$\hat{s}$*,*$\hat{p}$*,*$\hat{o}$*)*$\in E$*,* $\hat{s}$ *é a origem da aresta,* $\hat{p}$ *é o rótulo da propriedade,* $\hat{o}$ *é o alvo da aresta; e (3)* r *é uma função parcial que atribui expressões de filtro para nós variáveis em* G*. Uma expressão de filtro é expressa pela forma* $?x\ \theta\ c$*, onde* $?x \in \mathcal{V}$*,* $c \in \{\mathcal{U} \cup \mathcal{L}\}$ *and* $\theta \in \{=, >, \leqslant, <, \geqslant\}$*.*

A partir deste ponto, usa-se $V(G)$ e $E(G)$ para denotar o conjunto de nós e arestas de um padrão de grafo, respectivamente.

Consultas SPARQL são construídas a partir de padrões de grafos em conjunto com os operadores AND, OPTIONAL e UNION. Assume-se que o operador FILTER é representado pela função r no padrão de grafo. Neste trabalho é considerada a classe de expressões SPARQL *bem-formadas* originalmente identificada por [Pérez et al., 2009]. Consultas *bem-formadas* foram introduzidas como um fragmento de consultas que utilizam o operador OPTIONAL com uma complexidade adequada para avaliação da consulta. Padrões de consultas SPARQL bem-formadas são recursivamente definidas como segue:

Definition 4.0.2 *(Consulta SPARQL): Qualquer padrão de grafo definido como G é uma consulta SPARQL. Se g_1 e g_2 são consultas SPARQL, então expressões (g_1 AND g_2), (g_1 UNION g_2) and (g_1 OPTIONAL g_2) são consultas SPARQL. Para qualquer sub-consulta $g' = (g_1$ OPTIONAL $g_2)$ de uma consulta g, se uma variável $?x$ ocorre em g_2 e fora de g', então $?x$ também ocorre em g_1.*

A arquitetura considerada para o processamento de consultas SPARQL é apresentada pela Figura 4.1. Um usuário submete sua consulta para um *Coordenador* que gera um plano de consulta e entrega o plano para todos os servidores que mantêm os dados requisitados. Cada servidor executa o plano da consulta em paralelo sob a coordenação de um *Engine* de execução que efetua o casamento da consulta sobre os sub-grafos das partições. Ao final, os resultados provenientes de todos os servidores são unidos e enviados como resposta ao cliente. Durante a execução da consulta, servidores podem necessitar se comunicar entre si para troca de resultados intermediários. Estas rodadas de comunicação entre servidores podem tornar-se em um gargalo de desempenho e, consequentemente, elevar a latência das consultas. O objetivo da solução de particionamento é reduzir, sempre que possível, estas rodadas de comunicação através do agrupamento adequado de dados requiridos por uma consulta. Intuitivamente, o tempo de resposta em consultas diminui

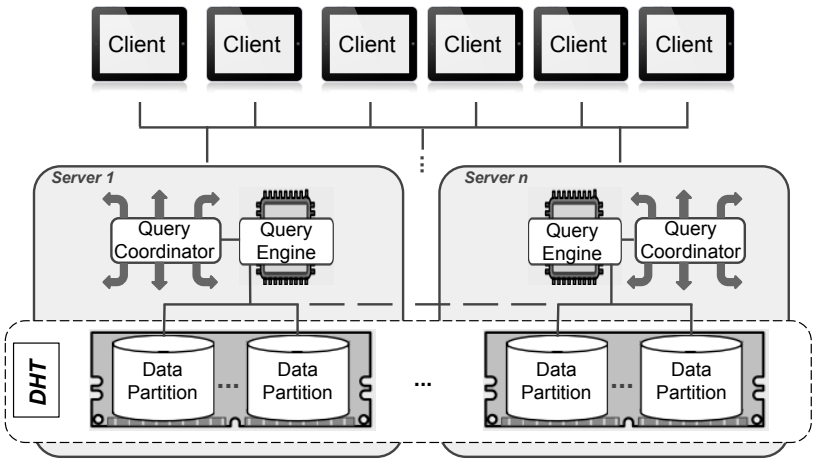


Figura 4.1: Arquitetura de Processamento de Consultas SPARQL

quando a troca de mensagens entre servidores é evitada em tempo de consulta conforme demostrado por [Fan, 2012].

Embora o foco deste trabalho não é prover um *engine* distribuído de processamento para consultar dados RDF, foi desenvolvido um *engine* para avaliar a recuperação de dados efetuado por uma consulta sobre repositórios de dados particionados. Em resumo, uma consulta SPARQL é decomposta em um conjunto de padrões de triplas para conduzir uma sequência de casamentos no grafo de forma que os padrões de triplas mais seletivas tenham a preferência no processamento. Neste processo de exploração ocorre a poda de casamentos desnecessários baseada na sequência dos casamentos de triplas e seus resultados intermediários associados. Em oposição a esta abordagem, abordagens relacionadas têm por prática comum processar padrões de triplas separadamente para posteriormente combinar resultados parciais através de operações de junções, o que inevitavelmente resulta em uma grande quantidade de resultados intermediários.

Um conjunto de consultas SPARQL Q é representado por padrões de grafos definidos por G. A sequência de padrões de triplas de uma consulta $q \in Q$ que representa seu plano de consulta é submetida a todos os servidores que mantêm dados requisitados pelo primeiro padrão. Intuitivamente, isto leva a uma execução de consulta paralela baseado no intervalo de casamentos de cada servidor com relação ao primeiro padrão. Assume-se que uma consulta $q \in Q$ pode ser decomposta em um conjunto de sub-consultas $\{q_1, ..., q_n\}$

onde $E(q_i) = E(q)$, $V(q_i) = V(q)$ e n representa a quantidade de servidores que detêm os dados requeridos pelo primeiro padrão de q tal que $n \geq 1$. Com o objetivo de evitar a troca de mensagens entre servidores durante a execução da consulta, este trabalho prevê o particionamento dos dados de forma que o processamento de consultas possa ser efetuado em paralelo sempre que possível. Isto é, gerar um particionamento $\mathcal{P} = \{P_1, ..., P_m\}$ para um grafo RDF representado por D sobre m servidores, onde a quantidade de partições necessárias para recuperar dados de cada sub-consulta $q_i \in Q$ é minimizada. Para este fim, considera-se que a qualidade de um particionamento é dada pela medida $\hat{P}$ em relação a uma sub-query q_i como segue:

Definition 4.0.3 *Dado um particionamento $\mathcal{P}$ de um grafo RDF D, deixe $B(q_i)$ ser um conjunto de sub-grafos de D que representa os casamentos de uma sub-consulta $q_i \in Q$. A medida $\hat{P}$ de $\mathcal{P}$ com relação a q_i é definida por:*

$$\hat{P}(q_i, \mathcal{P}) = 1 + \Big|\{(D', P') \in B(q_i) \times \mathcal{P} | E(D') \cap E(P_i) \neq \oslash\}\Big| - \Big|B(q_i)\Big| \tag{4.1}$$

Uma extensão da medida $\hat{P}$ é considerada para definir o objetivo do particionamento da presente abordagem. Antes de apresentar propriamente o objetivo, é importante destacar que a solução proposta favorece as consultas mais frequentes de uma carga de trabalho, onde a função f define a frequência esperada de uma consulta q em um período de tempo. O problema de particionamento proposto é formalmente definido para encontrar um particionamento $\mathcal{P}$ que minimiza a seguinte equação:

$$min \sum_{q_i \in Q}^{\mathcal{P}} f(q_i).\hat{P}(q_i, \mathcal{P}) \tag{4.2}$$

O problema de computar um particionamento adequado para um repositório RDF é proposto através de uma solução baseada em informações da carga de trabalho. A carga de trabalho é caracterizada para identificar nós acessados em conjunto por um conjunto de consultas em Q. O caminho percorrido por consultas e suas frequências respectivas são considerados para quantificar a afinidade entre pares de nós. Esta medida de afinidade

é a base da solução de particionamento proposta. O problema de particionamento é decomposto em dois sub-problemas, denominados *fragmentação* e *agrupamento*.

A *fragmentação* tem por objetivo gerar fragmentos os quais a estrutura corresponde a estrutura das consultas mais frequentes da carga de trabalho. Neste nível são produzidos fragmentos de acordo com a afinidade entre itens de dados. Na arquitetura proposta, um fragmento é uma unidade de armazenamento assim como uma unidade de transferência de dados entre servidores. Observa-se que o desempenho do processamento de consultas distribuídas não é apenas afetado pela quantidade de mensagens intercambiadas, mas também pelo tamanho destas mensagens. Um tamanho adequado de mensagens motivou a adoção de um limiar de armazenamento para determinar o tamanho máximo dos fragmentos. O tamanho deste limiar depende do ambiente e configurações da rede, e devem ser experimentalmente determinadas. Na fase de *agrupamento*, a medida de afinidade é estendida aos fragmentos gerados pela fase anterior para agrupar fragmentos relacionados. Em resumo, a *fragmentação* cria unidades de armazenamento e o *agrupamento* procura alocar fragmentos relacionados em um mesmo servidor. Valores de afinidade são considerados em ambas as etapas como uma medida para suportar a equação objetivo definida em 4.2.

4.1 Caracterização da Carga de Trabalho

Nesta seção é apresentado um método para representar informações da carga de trabalho. O fundamento deste método é identificar e medir relações de afinidade entre nós do grafo RDF. A notação de afinidade detém um papel fundamental na abordagem de particionamento a ser proposta.

Denomina-se uma *Estrutura RDF* uma composição da estrutura de um grafo RDF e o tamanho esperado para suas instâncias. Embora RDF possa definir um modelo livre de esquema, em geral um grafo RDF representa ambos esquema e instâncias. Usualmente, repositórios RDF definem a propriedade `type` para conectar entidades a suas respectivas

classes. Na Figura 4.2 uma *Estrutura RDF* é representada pela forma tracejada e denota os componentes das classes e os relacionamentos entre elas. Uma *Estrutura RDF* é um grafo cíclico não direcionado e definido como uma 6-tupla $S = (C, L, l, A, s, o)$, onde (1) C é um conjunto de nós rotulados que representam classes RDF; (2) L é um conjunto de nós rotulados que se referem a propriedades entre classes e valores literais no grafo RDF; (3) l atribui um tipo de dado para cada nó em L; (4) A é um conjunto de arestas não-direcionadas $(n_1, n_2) \in \{C \cup L\} \times \{C \cup L\}$ representando associações entre nós; (5) s é uma função que atribui o tamanho esperado para as instâncias de nós em $\{C \cup L\}$; e (6) o fornece a cardinalidade esperada das associações entre dois nós; isto é, uma função que mapeia um par em $C \times (C \cup L)$ para um inteiro que define para cada nó $n_1 \in C$ o número esperado de ocorrências das associações para um nó $n_2 \in (C \cup L)$.

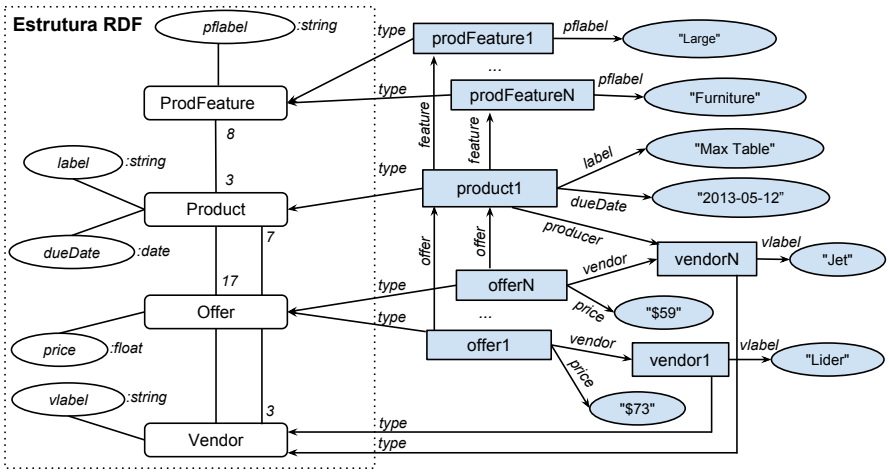


Figura 4.2: Grafo RDF

A Figura 4.2 mostra uma *Estrutura RDF* onde o tamanho de um nó consiste do tamanho esperado dos seus valores. Para fins de simplificação, nos exemplos é considerado que para qualquer nó n, $s(n) = 1$. Adicionalmente, no exemplo, $o(Product, ProdFeature) = 8$ porque o número médio de ocorrências de *ProdFeature* associado a uma instância de *Product* é 8. De forma análoga, uma instância de *ProductFeature* é relacionada a três instâncias de *Product* em média. Isto é, $o(ProdFeature, Product) = 3$. Além disto, existem relacionamentos multi-valorados entre $(Product, Offer)$ e $(Vendor, Offer)$. Assume-se que para as associações restantes entre quaisquer nós n_1 e n_2 no exemplo, $o(n_1, n_2) = 1$.

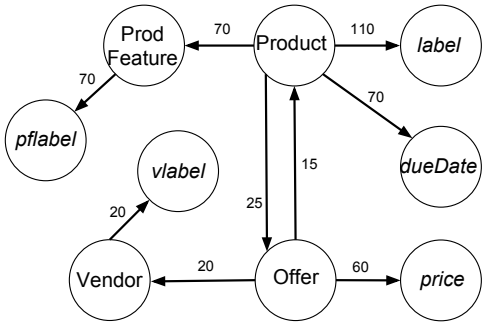


Figura 4.3: Grafo de Afinidade

Dada uma representação de uma *Estrutura RDF*, a carga de trabalho é caracterizada com base no conjunto de caminhos percorridos em cada uma das consultas. Formalmente, Q é um conjunto de consultas SPARQL representadas por padrões de grafos definidos por G, onde uma função f define a frequência esperada de consultas em Q.

Dada uma carga de trabalho sobre uma *Estrutura RDF*, a afinidade de dois nós n_i e n_j é dada pela frequência na qual elas são acessadas em conjunto por qualquer consulta da carga de trabalho. Deste modo, uma função de afinidade $aff(n_i, n_j)$ recebe como parâmetro um conjunto de consultas Q e computa a soma das frequências das consultas que involvem ambos n_i e n_j através de uma direção específica, isto é, n_i é o nó origem e n_j é o nó destino em um padrão de grafo. Formalmente, $Q_{ij} = \{q \in Q \mid (n_i, p_{ij}, n_j) \in E(q)\}$, e $aff(n_i, n_j) = \sum f(q)$, $q \in Q_{ij}$. A função de afinidade pode ser usada para rotular arestas em um grafo direcionado envolvendo todos os nós de uma *Estrutura RDF*, conforme mostra a Figura 4.3. A este grafo é dado o nome de *Grafo de Afinidades* o qual é definido como uma tupla $\mathcal{A} = (N, E, \texttt{aff})$, onde N é um conjunto de nós em uma *Estrutura RDF* e E é um conjunto de arestas que relacionam dois nós n_i e n_j por um valor de afinidade ($\texttt{aff}(n_i, n_j)$).

A solução de particionamento proposta é baseada em dois sub-problemas. O primeiro consiste da fragmentação de dados, isto é, como efetuar o corte de um grafo RDF de forma a manter dados fortemente relacionados por relações de afinidade em uma mesma unidade de armazenamento. O segundo envolve agrupar e alocar fragmentos relacionados em um mesmo servidor.

4.2 Fragmentação RDF

O desempenho de consultas distribuídas não é apenas afetado pela quantidade de mensagens trocadas entre servidores, mas também pelo tamanho de suas mensagens. Um tamanho adequado para estas mensagens motivou a adoção de um limiar de armazenamento como a base do método de particionamento proposto. Denota-se por Γ o limiar de armazenamento assumido para uma dada carga de trabalho. Intuitivamente, o objetivo é particionar nós de uma *Estrutura RDF*, de forma que as partições contenham a maior quantidade de nós relacionados possível sem extrapolar o limiar de armazenamento assumido. Na sequência é introduzido o problema de fragmentação RDF bem como o método proposto para resolvê-lo.

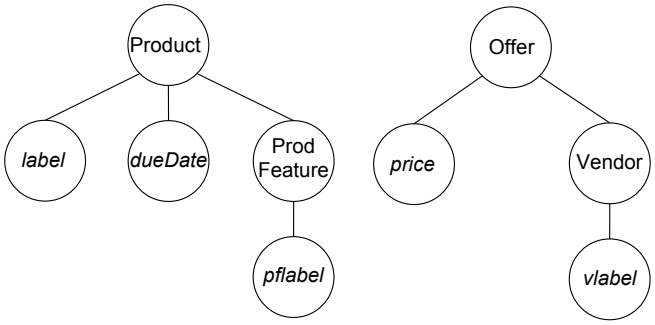
4.2.1 O Problema da Fragmentação RDF

Dada uma *Estrutura RDF* $S = (C, L, l, A, s, o)$ e um grafo de afinidade $\mathcal{A} = (N, E, aff)$, pretende-se obter um *template* de fragmentação $T = \{t_1, ..., t_m\}$, $m \geqslant 1$, de forma que t_i é um sub-grafo de S, $\bigcup_{i=1}^{m}(t_i) = (N, E')$, onde $E' \subseteq E$. A Figura 4.4 apresenta um exemplo de um *template* de fragmentação para uma *Estrutura RDF* da Figura 4.2.

Dado que a fragmentação é baseada em um limiar de armazenamento, é necessário determinar o tamanho esperado para um *template* de fragmento $t_i \in T$. O tamanho de t_i é dado pela soma do número esperado de ocorrências dos nós multiplicados pelos seus respectivos tamanhos. A composição em árvore para *templates* de fragmento requer medir a ocorrência do nó na sua estrutura de aninhamento. A função $occ(n)$ mapeia cada nó em um *template* t_1 para seu número esperado de ocorrências em uma instância de t_i. A função é recursivamente definida como segue: $occ(n) = 1$ se n é o nó raiz de t_i, e $occ(n) = occ(p) \times o(n)$ onde p é um nó pai de n em t_i. O tamanho de t_i é finalmente definido por $size(t_i) = \sum_{n \in t_i}(occ(n) \times s(n))$.

Com o objetivo de formalmente definir o problema de fragmentação, é introduzida a noção de um conjunto de nós fortemente correlacionados *scs* a um nó específico de um

grafo de afinidade, definido como segue: $scs(n) = \{n' | aff(n, n') \geq aff(n', n'')$ para cada nó n'' diretamente conectado a $n'\}$. Intuitivamente, *scs* determina quais nós detêm as mais altas afinidades com n que com outros nós do grafo. Denota-se por scs^+ o fechamento transitivo da relação *scs*.



(a) t_1 (b) t_2

Figura 4.4: *Templates* de fragmentação

O problema de fragmentação constitui em encontrar T de forma que as seguintes condições sejam satisfeitas: (1) $size(t_i) \leq \Gamma$ para cada $t_i \in T$; e (2) se n_1 e n_2 são nós no mesmo fragmento, então $n_2 \in scs^+(n_1)$. A primeira condição define que todos os fragmentos em T devem se ajustar a Γ e a última estabelece a geração de fragmentos pelo agrupamento de nós relacionados por valores de afinidades mais altas do que com nós em outros fragmentos.

Como exemplo, considere $\Gamma = 20$ e o grafo de afinidade da Figura 4.3. O *template* de fragmentação da Figura 4.4 satisfaz ambas as condições porque (1) o tamanho dos *templates* não extrapola o limiar de armazenamento, isto é $size(t_1) = 19$ e $size(t_2) = 4$; e (2) a afinidade entre qualquer nó em t_1 com qualquer outro nó em t_2 é menor que a afinidade entre qualquer par de nó no mesmo fragmentos, por exemplo, $aff(Offer, Product) < aff(Offer, Vendor)$.

4.2.2 O Algoritmo *affFrag*

Um algoritmo de fragmentação é proposto com base em *Estruturas RDF* e cargas de trabalho. O Algoritmo *affFrag* recebe como entrada uma *Estrutura RDF* S com infor-

mações do tamanho dos nós e o número de suas ocorrências, um grafo de afinidades $\mathcal{A}$ e um limiar de armazenamento Γ. O algoritmo produz *templates* de fragmentos baseado na análise de conjuntos fortemente relacionados dos nós se seus respectivos tamanhos estão de acordo com Γ.

Algoritmo affFrag

Entrada: *Estrutura RDF* $S = (C, L, l, A, s, o)$, Grafo de Afinidades $\mathcal{A} = (N, E, aff)$ e Γ
SaÃda: T *template* de fragmentação
$T \leftarrow \{\}$;
$allNodes \leftarrow N$;
$allEdges \leftarrow E$;
repeat
  $(n_1, n_b) \leftarrow$aresta em *allEdges* com a mais alta afinidade;
  $tNodes \leftarrow \{n_1\}$;
  $tEdges \leftarrow \{\}$;
  $tSize \leftarrow s(n_1)$;
  $Occ(n_1) \leftarrow 1$;
  $border \leftarrow \{(n_1, n_b) | n_b \in allNodes\}$;
  $allNodes \leftarrow allNodes - \{n_1\}$;
  Enquanto $tSize < \Gamma$ *and* $border! = \{\}$ **faça**
    $(n_1, n_b) \leftarrow$ **extrair** aresta de *border* com a mais alta afinidade, onde $n_1 \in tNodes$ e $n_b \notin tNodes$;
    $n_b Edges \leftarrow \{(n_b, n) \in allEdges | n \in allNodes\}$;
    Se *para toda aresta* $e \in n_b Edges$*:* $aff(e) \leq aff(n_1, n_b)$ **então**
      $Occ(n_b) \leftarrow Occ(n_1) \times o(n_b)$;
      Se $s(n_b) \times n_b Occ + tSize \leq \Gamma$ **então**
        $tNodes \leftarrow tNodes \cup \{n_b\}$;
        $tEdges \leftarrow tEdges \cup \{(n_1, n_b)\}$;
        $boder \leftarrow border \cup n_b Edges$;
        $allNodes \leftarrow allNodes - \{n_b\}$;
        $tSize \leftarrow tSize + s(n_b) \times n_b Occ$;
      fim
    fim
  fim
  $T \leftarrow T \cup \{(tNodes, tEdges)\}$;
  $allEdges \leftarrow allEdges - tEdges$;
until $allNodes == \{\}$;
output T;

Variáveis auxiliares são utilizadas para computar os *scs*'s: *allNodes* para manter os nós em $\mathcal{A}$ que não foram atribuídos a um fragmento até o momento; e *allEdges* para manter as arestas em $\mathcal{A}$ que são percorridas para compor um conjunto fortemente relacionado. Nós e arestas no fragmento corrente que está sendo computado são inseridos nas variáveis *tNodes* e *tEdges*, respectivamente, enquanto seu tamanho é mantido em *tSize*. A variável *border* consiste de arestas que correspondem ao fechamento transitivo de *scs*.

O algoritmo processa as arestas em $\mathcal{A}$ em ordem descendente de afinidade. Dada uma arestas (n_1, n_b), o objetivo inicial é computar $scs(n_1)$. O nó n_1 é definido como a raiz do *template* de fragmento que está sendo formado por razão de n_1 constituir o nó origem da aresta com a mais alta afinidade. Um novo fragmento é gerado pelo processamento de arestas (n_1, n_b) em *border* como segue: n_b é somente inserido no fragmento corrente se ele estiver relacionado com algum nó já inserido no fragmento corrente e a afinidade for mais alta do que qualquer outra medida de afinidade com nós fora do fragmento (Linhas 14-15). De acordo com a Linha 13, nós candidatos são processados em ordem decrescente de afinidade para formar o *template* de fragmento corrente com nós relacionados. Ao final, todos os nós terão sido atribuídos a algum dos *templates* de fragmento. Porém, antes de inserir novos nós em $tNodes$ verifica-se se sua inclusão não excederá o limiar Γ dado o tamanho e ocorrência do nó a ser incluído (Linhas 16-17).

Como um exemplo, considere o grafo de afinidades da Figura 4.3 e $\Gamma = 20$ como parâmetros de entrada do algoritmo *affFrag*. A primeira aresta a ser processada é aquela com a mais alta afinidade envolvendo os nós *Product* e *label*. *Product* é inserido no *template* t_1 como o nó raiz. O tamanho de t_1 assume inicialmente 1, dada a asserção que todos os nós possuem tamanho 1. Uma vez que o limiar de armazenamento não foi atingido, nós continuam a ser inseridos em t_1 dentre aqueles conectados a *Product* que estão mantidos em *border*. Dentre estes, o nó com mais alta afinidade é *label*. Tal nó é inserido em t_1, uma vez que ele não está conectado a qualquer outro nó com afinidade mais alta e sua inserção não excede Γ. O mesmo comportamento se dá na inserção dos nós *dueDate*, *ProdFeature* e *pflabel* em t_1. Neste ponto, *tSize = 19* dado a ocorrência simples de *dueData* e *label* com a múltipla ocorrência de *ProdFeature* e *pflabel*. As próximas arestas a considerar em *border* relacionam *Product* a *Offer* e *price*. *Offer* não deveria ser inserida no fragmento porque sua afinidade é mais alta com nós que não estão no *template* corrente. Assim, o primeiro *template* é criado com os nós *Product*, *label*, *dueDate*, *ProdFeature* e *pflabel*. Um processo similar cria o segundo *template* com *Offer*, *price*, *Vendor* e *vlabel*. O *template* de fragmentação final gerado é apresentado pela Figura 4.4.

O *template* de fragmentação define como particionar instâncias de uma *Estrutura RDF*, isto é, um grafo RDF. Assim, um fragmento é gerado para cada instância do nó raiz de acordo com o *template* $t_i \in T$. No exemplo, t_1 deve gerar fragmentos para cada instância de *product*.

4.3 Agrupamento de Fragmentos

Dada a abordagem proposta para tratar o problema de fragmentação, o problema de alocação é tratado na sequência. Isto é, dado que um fragmento é a unidade de comunicação entre servidores, o problema da alocação envolve determinar quais fragmentos deveriam ser alocados em um mesmo servidor. O problema da alocação é tratado em dois passos. Primeiramente é determinado quais fragmentos de *templates* diferentes devem ser co-alocados. O conjunto destes fragmentos é referenciado por *grupos*. Em seguida, a alocação dos grupos nos servidores é determinada por qualificadores de consultas.

Para agrupar *templates* de fragmentação, observa-se que embora o algoritmo *affFrag* corta o grafo de afinidades baseado nas relações de alta afinidade, nós atribuídos a *templates* diferentes podem ainda manter uma forte relação de afinidade. Isto ocorre porque o processo de fragmentação foi projetado para satisfazer a um limiar de armazenamento. Para evitar a comunicação entre servidores quando uma consulta envolve fragmentos de diferentes *templates*, procura-se co-alocar tais fragmentos em uma mesmo servidor sempre que possível. Dado que é possível existir diversas relações de afinidade entre *templates* de fragmento, da-se preferência por agrupar aqueles que detêm as mais altas afinidades. Considere que um *template* de fragmentação $T = \{t_1, ..., t_m\}$ é definido baseado em um grafo de afinidades $\mathcal{A} = (N, E, \texttt{aff})$ e que $E_T \subseteq E$ é o conjunto de arestas que conectam um nó em um fragmento t_i ao nó raiz de um fragmento t_j, tal que $i \neq j$. Um *template* de agrupamento é definido como $G = \{g_1, ..., g_n\}, n \leq m$, tal que G é uma floresta de *templates* de fragmentos relacionados. De maneira análoga ao algoritmo *affFrag*, grupos em G são construídos considerando arestas de E_T em ordem descendente de valores de

afinidade. Embora não se define um limiar para o tamanho dos grupos, ele é limitado pela a capacidade de armazenamento de um servidor. Como um exemplo, a Figura 4.5 apresenta um *template* de agrupamento que relaciona os fragmentos t_1 e t_2 através da aresta (*Product*, *Offer*).

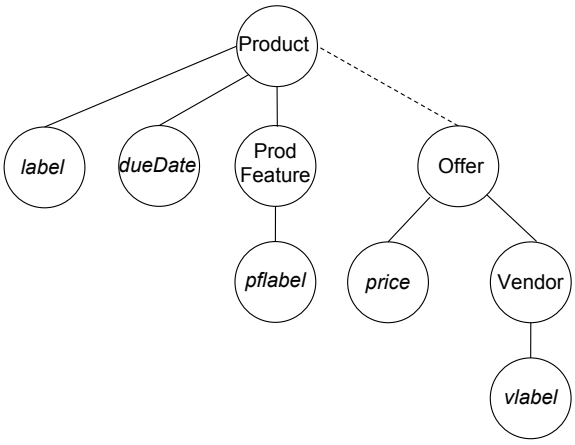


Figura 4.5: *Template* de Agrupamento

No segundo passo, o problema da alocação dos grupos nos servidores é tratado. Da mesma maneira que o método aplicado no projeto físico de banco de dados tradicional, nesta abordagem são considerados qualificadores de consultas para co-alocar *templates* de agrupamento baseado nos valores dos nós. Para este fim, são considerados apenas valores de nós mono-valorados dentro do grupo. Intuitivamente, não seria possível assumir um critério de co-alocação baseado em nós multi-valorados uma vez que a ordem para os valores destes nós dentro de uma instância de grupo podem não coincidir com a ordem em outra instância. O objetivo do processo de co-alocação é construir uma lista de nós qualificadores $L_{g_i} = \langle l_1, ...l_n \rangle$ tal que para quaisquer qualificadores l_i, l_j, um grande número de consultas pode se beneficiar da alocação de l_i se $l_i < l_j$. Uma vez mais é reutilizada a ideia da técnica de agrupamento aplicada a banco de dados tradicionais baseada em chaves compostas. Formalmente, seja P o conjunto de nós qualificadores de uma carga de trabalho Q, para cada grupo g_i os nós que compõe L_{g_i} são definidos pelo conjunto $P \cap \{n \mid \text{ nó } n \in g_i, occ(n) = 1\}$. Nós neste conjunto são alocados em ordem descendente da frequência das consultas nas quais os qualificadores são aplicados, dado por $freq(l_i)$. Isto

é, $freq(l_i) = \sum f(q)$, $q \in Q, l_i \in qual(q)$ e para cada l_i, l_j, se $i < j$ então $freq(l_i) > freq(l_j)$. Como um exemplo, considere os dados da carga de trabalho e os fragmentos gerados a partir dos *templates* da Figura 4.4, os fragmentos devem ser agrupados pela ordenação dos valores de *dueDate*.

CAPÍTULO 5

CLUSTERRDF

Um sistema baseado na arquitetura da Figura 4.1 foi desenvolvido para construir e avaliar o método de particionamento proposto. Nesta seção são apresentadas as decisões de projeto efetuados durante a construção deste sistema denominado *ClusterRDF*.

5.1 Armazenamento de Dados

O repositório chave-valor Scalaris [Schütt et al., 2008] foi utilizado pela camada de armazenamento. Scalaris é um repositório em memória que implementa uma rede virtual. Sua composição envolve uma rede de sobreposição estruturada que confere desempenho de roteamento em escala logarítma que é usada para armazenar e recuperar pares chave-valor distribuídos entre um conjunto de servidores. A justificativa para o uso do Scalaris é dada por duas razões principais. Primeiramente, trata-se de um sistema capaz de conferir disponibilidade e escalabilidade para sistemas de armazenamento em nuvem. A segunda razão se refere a noção de localidade de dados fornecida pelo sistema através da alocação de dados que mantem a ordem lexicográfica de chaves. A habilidade de suportar a localidade de dados é essencial para permitir o agrupamento de fragmentos relacionados como proposto pelo presente trabalho.

Para armazenar fragmentos RDF no Scalaris, foi considerado seu modelo genérico e simples baseado em pares chave-valor. Seja F um conjunto de fragmentos tal que $fr \in F$ é um sub-grafo RDF conformado a um *template* $t \in T$, um repositório de dados é populado como segue: um par chave-valor é gerado para cada fragmento $fr \in F$, tal que em cada par a chave contem um identificador único e o valor contem o fragmento RDF propriamente dito.

Fragmentos são co-alocados pela atribuição de um mesmo prefixo para as chaves destes

fragmentos. Note que se o repositório aloca dados em ordem lexicográfica, pares chave-valor que detêm um mesmo prefixo de chave são potencialmente armazenados em um mesmo servidor, ou então em servidores próximos. Para explorar este particionamento por intervalo de chaves do repositório subjacente, as chaves de fragmentos foram definidas de forma a se adequar a abordagem de alocação proposta. Assim, todos os fragmentos de um mesmo *template* de agrupamento devem possuir um prefixo comum para suas chaves. Para endereçar unicamente os dados, é introduzida uma função *id* que mapeia fragmentos ou valores de nós qualificadores para um identificador único. Dado um fragmento fr e seu *template* t, g denota o *template* de agrupamento para o qual t é atribuído, onde $L_g = \langle l_1, ..., l_n \rangle$ é a lista de valores para os qualificadores de g. A chave para fr é prefixada por identificadores dos qualificadores de g seguido pelo identificador do fragmento pai de fr em g, representado por fr_{parent}. O símbolo "/" é utilizado como operador de concatenação de *strings*. Uma expressão m para compor *strings* como pares chave-valor pode ser dada conforme segue:

$$\texttt{m = \{ id(}li_1\texttt{)/.../id(}li_n\texttt{)/id(}f_{parent}\texttt{)/id(f), f\}}$$

Como um exemplo, considera os *templates* de fragmentação e agrupamento das figuras 4.4 e 4.5, respectivamente. Exemplos de mapeamentos de fragmentos para pares chave-valor pode ser definido como segue:

$$\begin{aligned}&\texttt{m = \{ (id(``13-5-12")/id(}fr_{Product1}\texttt{), } fr_{Product1}\texttt{),}\\ &\qquad\texttt{(id(``13-5-12")/id(}fr_{Product1}\texttt{)/id(}fr_{Offer1}\texttt{), } fr_{Offer1}\texttt{),}\\ &\qquad\texttt{(id(``13-5-12")/id(}fr_{Product1}\texttt{)/id(}fr_{Offer2}\texttt{), } fr_{Offer2}\texttt{)\}}\end{aligned}$$

5.2 Recuperação de Dados

Embora o foco deste trabalho não está em prover um *engine* de processamento distribuído para consultas sobre dados RDF, um *engine* de recuperação de dados paralelo e distribuído foi desenvolvido para suportar avaliações experimentais utilizando diferentes soluções de particionamento. Especificamente, o objetivo é avaliar abordagens de parti-

cionamento em termos do custo de comunicação de mensagens trocadas entre servidores para a execução de consultas. Conforme demostrado por [Cong et al., 2007], o custo de comunicação na avaliação de consultas distribuídas é determinada pelo particionamento de um banco de dados a o tamanho do resultado da consulta. Assim, o *engine* para processamento de consultas desenvolvido é capaz de simular a execução de uma consulta pela recuperação de dados definida em termos de requisições enviadas para servidores locais ou distribuídos.

Conforme apresentado anteriormente, o problema do processamento de consultas SPARQL pode ser definido como um problema de casamento de sub-grafos. Ao invés de junções custosas, é aplicada a exploração em grafos como método para processamento de consultas. Primeiramente, uma consulta é dada através de um padrão de grafo G. A execução se inicia pela busca dos casamentos de um nó v de G. A partir de cada casamento, o grafo é explorado para encontrar os casamentos dos filhos de v, e assim prossegue até que todos os nós em G tenham sido explorados. A ordem de exploração é importante para reduzir a quantidade de resultados intermediários. Assim, assume-se que um padrão de grafo de consulta é ajustado a um padrão de árvore de *template* de fragmento, de forma que o aninhamento estabeleça que os padrões de triplas mais seletivas sejam colocados próximo a raiz do padrão de árvore para que sejam exploradas por primeiro. A exploração se dá paralelamente sobre servidores distribuídos. No passo final, os casamentos de todos os servidores são coletados para produzir o resultado final.

Os componentes da arquitetura idealizada pela Figura 4.1 são implementados por componentes correspondentes no sistema desenvolvido, como mostra a Figura 5.1. Uma consulta é submetida a um servidor específico. O *Coordinator* gera o plano de recuperação de dados e entrega-o, em paralelo, a todos os servidores que mantem os dados requisitados. Cada servidor executa o plano através da requisição de dados a uma DHT sob a coordenação de um nó *Scalaris*. Uma observação importante é que o *Scalaris* provê uma funcionalidade de empacotamento de conjunto de requisições para um determinado servidor em uma única mensagem para minimizar o custo de troca de mensagens. Quando

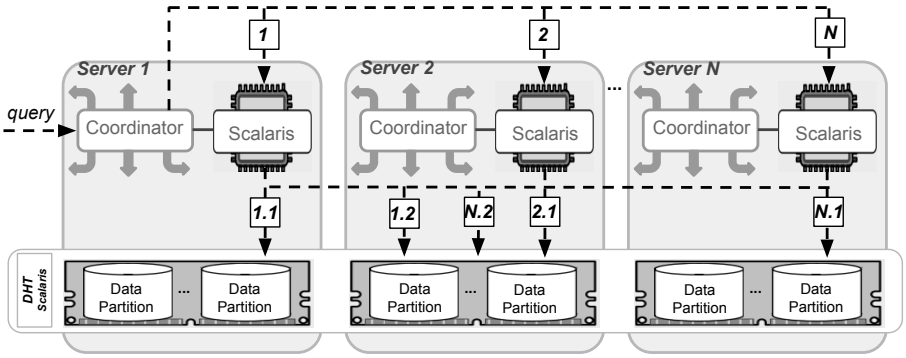


Figura 5.1: Arquitetura para o Processamento no ClusterRDF

todos os casamentos são obtidos, cada servidor envia os resultados parciais ao *Coordinator* que iniciou a consulta.

Para executar a exploração de grafos, o *Coordinator* gera o plano de recuperação completo antes de iniciar o processamento. O plano é definido em termos de um conjunto de chaves para endereçar pares chave-valor que representam fragmentos que contem dados requisitados pela consulta. Este conjunto de chaves é obtido pela emulação da exploração de grafos no padrão de grafo G. Na prática, este conjunto de requisições a chaves é decomposto em um número de sub-conjuntos correspondendo ao número de *threads* paralelas, cada qual executada por um servidor. Em resumo, o conjunto de requisições é submetido a um servidor que deve recuperar dados locais e, se necessário, dados distribuídos.

A Figura 5.1 mostra os passos de um processamento paralelo para recuperação de dados de uma consulta. Como um exemplo, considere que uma consulta é submetida para o *Coordinator* do *Server1* sobre um repositório de dados distribuído e particionado de acordo com os *templates* da Figura 4.4. Assuma que a consulta pode ser decomposta em sub-consultas definidas por intervalo de valores de *dueDate*. Assim, o *Coordinator* gera um conjunto de requisições de chave para cada intervalo e submete-o aos servidores que mantem tais chaves. Neste caso, *threads* de 1 a N foram iniciadas como mostrado pelas linhas tracejadas da Figura 5.1 e que estão rotuladas com os identificadores das *threads*. o *Server1* recupera dados locais através da requisição 1.1 e ainda necessita acessar o *Server2* através da requisição 1.2. Um processamento distribuído é também executados pela *thread*

N. Nestes casos, pode-se assumir que o processamento distribuído é necessário porque algumas informações de *ProdFeature* foram distribuídas entre os servidores. Isto não acontece na segunda *thread*, onde todos os dados requisitados se encontram armazenados no *Server2*.

A comparação da abordagem de particionamento proposta com abordagens relacionadas é realizada em termos do número de servidores acessados por *threads* individuais. Embora estas abordagens comparadas foram construídas sobre uma arquitetura de processamento genérica, é importante observar que os resultados reportados são válidos para determinar o custo da troca de mensagens em *engines* de processamento paralelos e distribuídos. Isto ocorre porque o modelo de processamento de consulta não tem efeito na necessidade de recuperação de todos os dados do conjunto de resultados, além disto não são considerados outros custos envolvidos na execução da consulta, como o tempo de CPU.

CAPÍTULO 6

AVALIAÇÃO EXPERIMENTAL

Um estudo experimental foi realizado para determinar o efeito da abordagem de particionamento proposta na recuperação de dados de consultas efetuadas sobre repositórios RDF distribuídos. O desempenho fornecido pela abordagem proposta por *ClusterRDF* foi comparado com as abordagens relacionadas introduzidas por Huang et al [Jiewen Huang, 2011] e Trinity.RDF [Zeng et al., 2013] usando o benchmark SPARQL chamado Berlin (BSBM).

Huang et al. aplica o particionador METIS [met, 2013] sobre um grafo RDF, em seguida replica nós para sobrepor dados entre partições de acordo com uma garantia *n-hop*. Esta abordagens é identificada neste estudo experimental como *METIS-2hops* porque uma versão deste método foi implementada utilizando uma garantia de *2-hops*. Embora Trinity.RDF é focada em prover um *engine* de consulta para RDF, este sistema considera um particionamento em *hash* de nós RDF e uma distribuição de lei de potência do grau de nós para efetuar o agrupamento de dados. Detalhes adicionais de implantação destes sistemas são fornecidos a seguir.

6.1 Abordagens Comparadas

METIS-2hops foi construída originalmente sobre o sistema RDF-3X usando o *framework* de processamento *Hadoop*. Nos experimentos, foi considerado apenas o método de particionamento e consultas executadas sobre a arquitetura de processamento introduzida na seção anterior. Observa-se que a arquitetura proposta aplica o método de exploração de grafos ao invés da exploração baseada em junções aplicada pelo RDF-3X. Por este motivo os resultados reportados neste estudo não são equivalentes aos reportados em [Jiewen Huang, 2011] e em [Zeng et al., 2013], dado que em ambos os casos o *METIS-2hops* é

implantado sobre o sistema RDF-3X. Além do método de exploração em grafo, a arquitetura considerada elimina o custo envolvido no escalonamento de processos que seriam necessários ao utilizar a plataforma *Hadoop*.

Outra decisão importante relacionada a implantação do *METIS-2hops* foi tomada com relação ao modelo de armazenamento, uma vez que esta abordagem foi originalmente projetada para *triple-stores*. Com o objetivo de prover uma comparação justa com *ClusterRDF*, decidiu-se por aplicar o mesmo limiar de armazenamento para os fragmentos de *METIS-2hops*, neste caso 12kb. Para isto, inicialmente foram geradas partições usando o *METIS* e, em seguida, alocou-se as triplas de acordo com a garantia não-direcionada de *2-hops*. No caso do tamanho da partição exceder o limiar, dividiu-se os arquivos em tamanhos adequados aos limiar e agrupou-se os arquivos resultantes através da atribuição de um mesmo prefixo para as chaves DHT. Observe que este procedimento de alocação é similar ao aplicado por *ClusterRDF* no sistemas descrito pelo capítulo anterior.

O particionador METIS foi aplicado utilizando a otimização de vértices de alto-grau conforme introduzido por [Jiewen Huang, 2011]. Vértices de alto-grau correspondem a nós RDF que estão conectados a muitos outros nós. Neste caso, um nó é dito de alto-grau se seu grau excede em três unidades o desvio padrão sobre o grau médio de vértices em um grafo RDF. Esta otimização visa evitar problemas no particionamento uma vez que grafos bem conectados são difíceis de particionar. Assim, vértices de alto-grau devem ser ignorados durante o particionamento no METIS e a alocação de *2-hops* das triplas. Após o particionamento e alocação, os vértices ignorados são adicionados as partições que contem o maior número de nós originalmente adjacentes a estes vértices.

O mesmo método para identificar nós de alto-grau é utilizado em *Trinity.RDF*. Entretanto, nós de alto-grau em *Trinity.RDF* não são ignorados como em *METIS-2hops*. Neste caso, um nós de alto-grau é co-alocado com nós da sua lista de adjacência. *Trinity.RDF* armazena dados como um conjunto de nós em um repositório chave-valor. Cada par mantem um identificador de um nó RDF e um conjunto de identificadores para os nós na sua lista de adjacência. Um servidor de índices é constituido para mapear *ids* de nós para

seus valores correspondentes. O mesmo modelo de armazenamento foi implementado para avaliar o *Trinity.RDF* neste estudo.

Os sistemas utilizados para executar *ClusterRDF*, *METIS-2hops* e *Trinity.RDF* foram desenvolvidos em Java e construídos sobre um *cluster* de instâncias EC2 da Amazon, cada qual com 15 GB de memória, 4 vCPU 8 GHz Intel Xeon e 420 GB de disco. Cada instância executou um nó *Scalaris* conectado a um sistema de DHT. Nós *Scalaris* compõem a camada de armazenamento e são responsáveis pela recuperação de dados através de requisições *put-get-remove* da *interface* DHT para obter os fragmentos requisitados. A comunicação entre o *Scalaris* e seu sistema de DHT subjacente é realizado por chamadas remotas de procedimento sobre o protocolo HTTP, enquanto a comunicação entre nós *Scalaris* usam um protocolo nativo em *Erlang*. O sistema foi configurado para manter 4 réplicas de cada dado armazenado para garantir a disponibilidade do sistema.

6.2 Configurações do Experimento

O Benchmark *Berlin SPARQL Benchmark* (BSBM) [Bizer and Schultz, 2009] foi aplicado neste estudo e foi construído sobre um caso de uso de *e-commerce*, onde o esquema modela os relacionamentos entre produtos, suas características, produtores, vendedores e revisões de produto. O BSBM provê uma carga de trabalho com 12 consultas e um gerador de dados que suporta a criação de bases de tamanho arbitrário baseado no número de produtos como fator de escala. Dentre as 12 consultas, 11 delas foram utilizadas por este estudo e correspondem ao conjunto de consultas que satisfazem a definição de padrões de grafos bem-formados. Diferentemente de outros *benchmarks* para RDF, o BSBM fornece um conjunto de informações adequado para a aplicação da caracterização da carga de trabalho, tais como frequências de consultas, tamanho de nós e cardinalidades sobre uma estrutura RDF.

Para um tamanho específico de banco de dados e carga de trabalho fornecida pelo BSBM, foram gerados repositórios particionados de acordo com as abordagens de *Clus-*

terRDF, *METIS-2hops* e *Trinity.RDF*. Para cada abordagem, um banco de dados foi populado em um repositório distribuído para fins da avaliação da recuperação de dados das consultas. A Tabela 6.1 sumariza as estatísticas dos bancos de dados utilizados neste estudo. O tamanho dos bancos se referem ao tamanho dos arquivos no formato *N-triple* gerado para BSBM. Como esperado, o *ClusterRDF* e *Metis-2hop* requerem um maior espaço de armazenamento em termos de triplas replicadas. Porém, *Metis-2hop* produz o dobro de triplas se comparado a *ClusterRDF*.

6.3 Resultados do Experimento

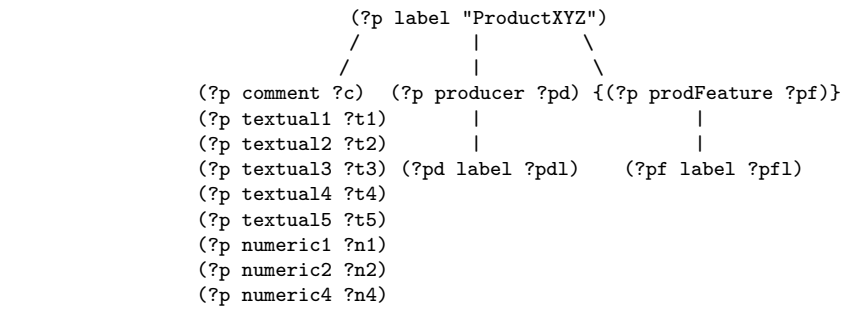
O objetivo dos experimentos reportados nesta seção é determinar o efeito do método de particionamento proposto sobre o desempenho do sistema, e compará-lo com as abordagens *Metis-2hops* e *Trinity.RDF*. A comparação está baseada no tempo de resposta para recuperar dados de uma consulta a partir de um repositório de dados. Os efeitos do particionamento são demonstrados através da análise de 4 consultas selecionadas. Os respectivos padrões de grafo das consultas são apresentados pela Figura 6.1.

6.3.1 Desempenho da Recuperação de Dados

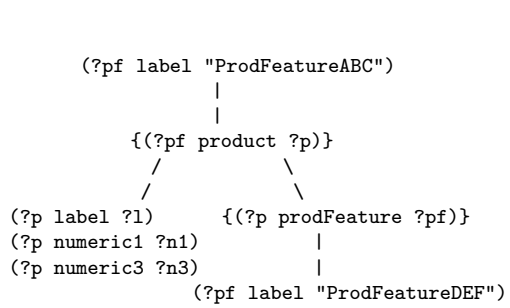
As abordagens de particionamento foram inicialmente comparadas em um *cluster* de 8 servidores com o *dataset* BSBM_5. Os resultados são apresentados pela Tabela 6.2. Os tempos obtidos em milissegundos correspondem a média de 10 execuções de cada consulta e representam o custo de recuperação de dados da consulta sobre um repositório

				Overhead de Triplas	
Dataset	#Produtos	#Triplas	Tamanho	*ClusterRDF*	*Metis-2hops*
BSBM_1	100	40405	10.2MB	14141	27071
BSBM_2	200	75620	19.2MB	22686	44615
BSBM_3	500	191650	48.9MB	67329	120739
BSBM_4	1000	375163	96MB	105045	213842
BSBM_5	10000	3567636	922.3GB	891909	1748141
BSBM_6	100000	35300350	68.6GB	7766077	15532154
BSBM_7	300000	100399052	27GB	20079810	40159620

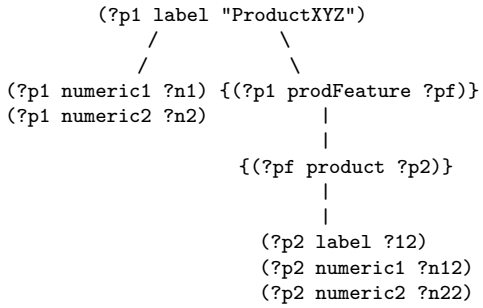
Tabela 6.1: Estatísticas de *datasets* utilizados na avaliação



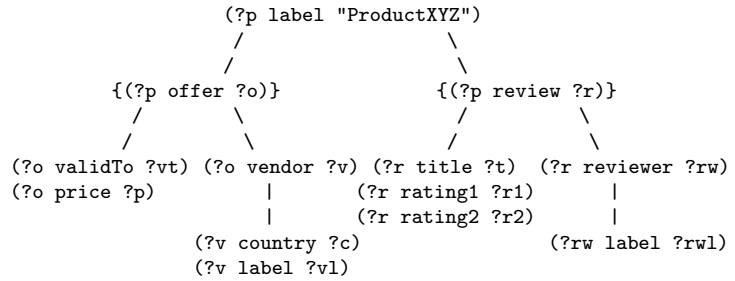
(a) Q2



(b) Q3



(c) Q5



(d) Q7

Figura 6.1: Padrões de Grafo para Consultas do Experimento

distribuído. De acordo com o sistema de processamento proposto, cada servidor inicia uma *thread* e executa um número arbitrário de requisições locais e remotas para recuperação dos dados. Nesta composição paralela de recuperação, a *thread* que executa o maior número de requisições entre servidores determina os tempos de resposta reportados. Foram coletados tanto o número máximo de requisições executadas por um único servidor, quanto o número total de requisições. Observe que o total de requisições tem uma correspondência direta ao tamanho do resultado da consulta. Isto é, uma métrica referente ao total de fragmentos (unidades de armazenamento) recuperados.

Tabela 6.2: Resultados para consultas BSBM - 8 servidores e *dataset* BSBM_5

Tempo de resposta para consultas BSBM (ms)											
Método	Q1(4%)	Q2(24%)	Q3(4%)	Q4(4%)	Q5(4%)	Q6(4%)	Q7(16%)	Q8(8%)	Q9(16%)	Q10(8%)	Q12(4%)
ClusterRDF	37,27	8,24	38,55	62,08	49,80	19,51	28,77	27,31	9,19	29,22	27,09
Metis-2hops	70,94	59,73	94,92	74,39	26,19	52,47	50,85	37,53	8,65	50,03	48,00
Trinity.RDF	67,52	79,04	133,41	161,25	87,80	209,47	132,22	38,93	15,76	91,00	76.12

#Requisições a Servidores Distintos											
ClusterRDF	4	1	4	6	5	1	3	3	1	3	3
Metis-2hops	7	6	8	7	3	3	5	4	1	5	5
Trinity.RDF	6	7	6	7	5	4	8	4	2	7	7

#Total de Requisições											
ClusterRDF	19	1	19	21	21	1250	35	22	1	23	3
Metis-2hops	9	15	10	10	6	957	50	31	2	45	8
Trinity.RDF	78	71	804	741	515	2847	156	30	2	91	23

6.3.1.1 Requisições entre Servidores

Como esperado, existe uma correspondência direta entre o número de requisições a servidores distintos e o tempo de resposta. Isto é, um alto número de requisições distribuídas representa um alto custo para recuperar dados distribuídos entre servidores distintos. Observe que a execução de $Q1$ em *ClusterRDF* requer o acesso a 4 servidores distintos, que executa em 37,27 ms. A execução da mesma consulta em *Metis-2hops* e *Trinity.RDF* quase dobra o número de requisições e tem o mesmo efeito no tempo de resposta (70,94 ms e 67,52, respectivamente).

Intuitivamente, o número de requisições entre servidores necessárias para recuperar dados de consultas mede a eficácia dos métodos de particionamento. A diferença entre os resultados para os métodos avaliados pode ser explicado pela cobertura que cada método proporciona, em termos de padrões de consulta. Pode-se verificar que o *Metis-2hops* assegura uma cobertura de *2-hops* a partir da raiz de qualquer árvore padrão. No entanto, uma garantia 2-hops não é suficiente para cobrir todo o padrão da maioria das consultas na carga de trabalho BSBM. Como exemplo, considere o padrão de grafo de $Q7$ na Figura 6.1(d). Neste caso, uma garantia *2-hops* não é suficiente para recuperar os dados de *vendor* e *reviewer*.

Observa-se que *Metis-2hops* não garante uma cobertura de *2-hops* para alguns casos para os quais é aplicada a otimização para vértices de alto-grau. Considere um caso particular para a consulta $Q2$ na Figura 6.1(a), onde o produto p a ser consultado é

identificado como um vértice de alto-grau. Observe que, se $?p$ é removido do grafo RDF, também se remove as associações de nós literais relacionados com $?p$. Como consequência, esses nós podem ser alocados entre partições arbitrárias. Embora $?p$ é adicionado à partição que contém o maior número de nodos relacionados, no passo final, não é possível evitar a distribuição completa dos nós literais. Isso explica o número elevado de requisições entre servidores para $Q2$ para *Metis-2hops*.

Trinity.RDF fornece uma cobertura simples na maioria dos casos devido à sua unidade de armazenamento de granularidade fina com base em nós RDF. Para os nós de alto-grau, *Trinity.RDF* tenta criar agrupamentos através da co-alocação de nós de alto-grau com seus nós adjacentes. A menos que os nós a serem consultados sejam nós de alto-grau, não há relação entre os nós destes agrupamentos e os padrões de consulta. Isso explica porque o *Trinity.RDF* apresenta os piores resultados dentre as três soluções.

ClusterRDF fornece uma cobertura completa para as consultas $Q2$, $Q6$ e $Q9$, uma vez que as requisições são emitidas para apenas um servidor. Para as demais consultas, *ClusterRDF* não consegue evitar requisições distribuídas. No entanto, ele reduz o número de servidores acessados quando comparado com as outras duas alternativas. Os resultados apresentados na Tabela 6.2 mostram que *ClusterRDF* supera *Metis-2hops* e *Trinity.RDF* para a maioria das consultas, com exceção de *Q5* e *Q9* em *Metis-2hops*. Isto porque *ClusterRDF* atribui dados para os agrupamentos de acordo com o padrão de acesso das consultas mais frequentes da carga de trabalho. Observe que, o *mix* de consulta BSBM é dado em termos de percentuais ao longo dos rótulos da consulta na Tabela 6.2. Dado que *ClusterRDF* favorece a cobertura das consultas mais frequentes, o desempenho de consultas menos frequentes, como *Q5* pode ser comprometida.

6.3.1.2 Total de requisições

O tamanho dos resultados das consultas é reportado pela quantidade de requisições totais na Tabela 6.2. Esta medida representa a quantidade total de fragmentos (unidades de armazenamento) recuperada. Lembre-se que o *Scalaris* fornece uma funcionalidade para

empacotar um conjunto de requisições para o mesmo servidor em uma única mensagem para minimizar o custo da troca de mensagens. Observa-se que o custo desses pacotes de mensagem pode ser ignorado quando a quantidade de requisições é de até 10 requisições por servidor. Este não é o caso de *Q6* que exige 1250 fragmentos de *ClusterRDF* alocados em apenas um servidor. Em *Trinity.RDF*, esse valor é ainda maior para todas as consultas. A razão para isso é o modelo de armazenamento de granularidade fina aplicado por *Trinity.RDF*.

O número total de requisições também está relacionado com a quantidade de dados irrelevantes nos fragmentos a ser recuperados. Observe que *ClusterRDF* requer um número menor de requisições a servidores que *Metis-2hops* em $Q6$. No entanto, *ClusterRDF* requer um número maior de requisições a fragmentos. Isto pode ser explicado pelo fato de que os dados requeridos estão no mesmo agrupamento/servidor, mas provavelmente não no mesmo fragmento.

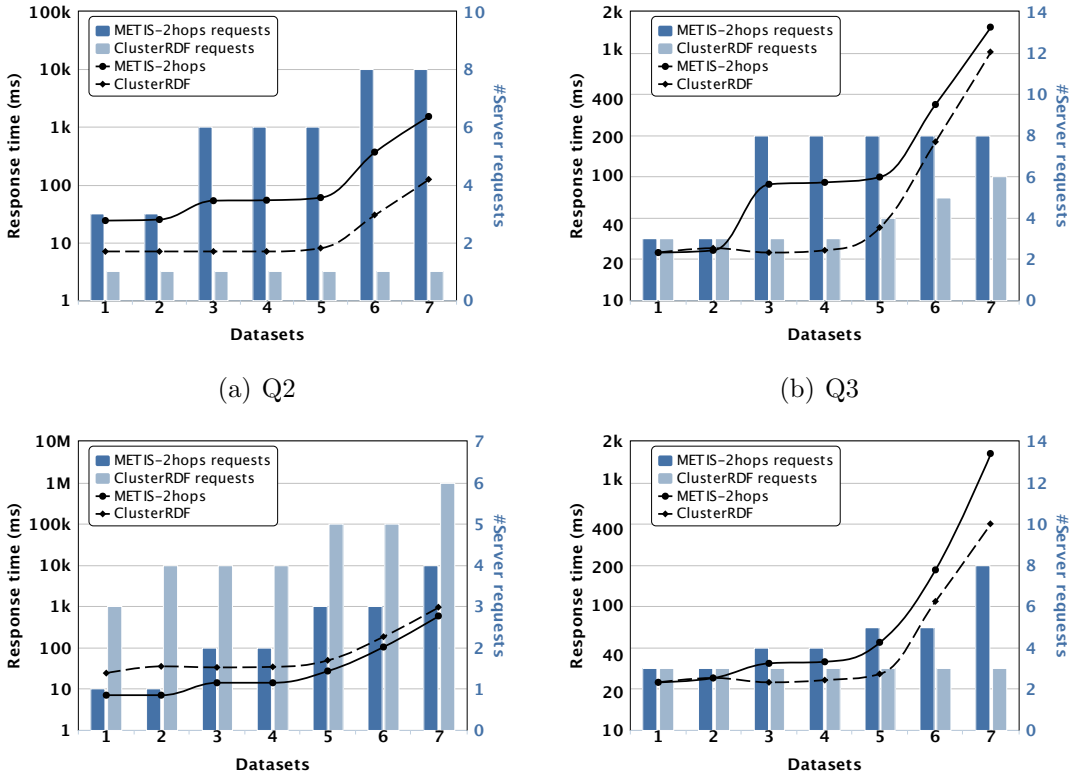
Além das requisições entre servidores e requisições no total, o tempo de resposta é também afetado pela estrutura da consulta, o tamanho do conjunto de dados e o número de servidores de armazenamento no sistema distribuído. Os efeitos destas variantes são discutidos a seguir em relação as abordagens *ClusterRDF* e *Metis-2hops*

6.3.2 Escalabilidade

Para avaliar a escalabilidade de *ClusterRDF* e *Metis-2hops*, foram realizados experimentos variando-se o tamanho dos *datasets* e a quantidade de servidores.

6.3.2.1 Escalabilidade de dados

Os métodos foram testados em um *cluster* de 8 servidores e 7 *datasets* (BSBM_1 a BSBM_7) de tamanhos crescentes. Os resultados são mostrados na Figura 6.2 (escala logarítmica). Em geral, os resultados destas consultas aumentam à medida que o tamanho do conjunto de dados aumenta. O aumento do tamanho do conjunto de dados leva a um



(a) Q2 (b) Q3

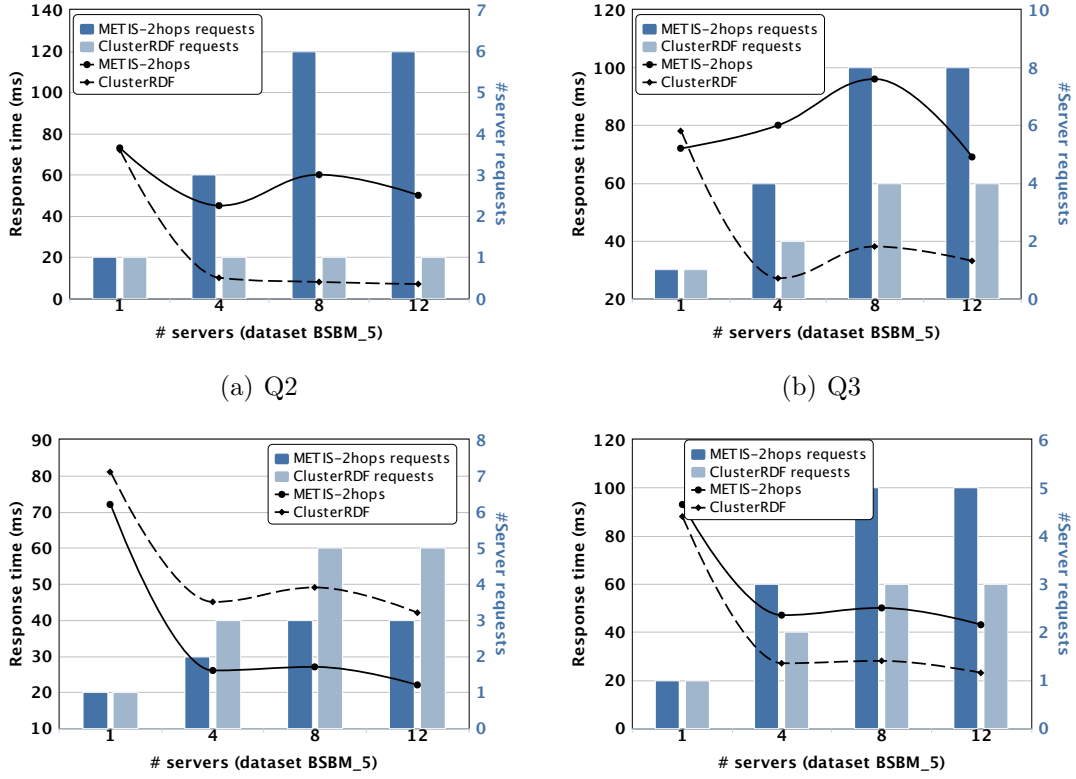
(c) Q5 (d) Q7

Figura 6.2: Escalabilidade de dados

maior número de requisições a servidores na maioria dos casos . Isto pode ser explicado por um grau mais elevado dos nós RDF que obriga a equilibrar a carga entre os servidores. No entanto, isso só acontece quando todo o conjunto de itens de dados de consulta não está agrupado. *ClusterRDF* mantém o mesmo número de requisições a servidores nas consultas *Q2* e *Q7* para todos os tamanhos. Por exemplo, em *Q2*, todos os dados necessários estão agrupados, e o aumento do conjunto de dados não é suficiente para sobrecarregar a capacidade do servidor. Lembre-se que uma outra razão que explica o aumento de requisições de servidores para $Q2$, $Q3$ e $Q7$ em *Metis-2hops* está relacionada com o problema da otimização nos vértices de alto-grau.

Metis-2hops supera *ClusterRDF* em $Q5$. Esta consulta representa um padrão de consulta recursiva entre *ProdFeature* e *Product*, que não é suportado por *ClusterRDF*. Apesar de uma garantia de *2-hops* poder representar tal associação, não é suficiente para cobrir o padrão completo de $Q5$. Outra consulta recursiva entre *ProdFeature* e *Product* é ap-

resentado em *Q*3. Neste caso, um resultado semelhante é relatado por *ClusterRDF* e *Metis-2hops* nos *datasets BSBM_1* e *BSBM_2*. Em *ClusterRDF*, os nós de produtos e suas características relacionadas são agrupadas por um qualificador composto sobre valores literais de *numeric1* e *numeric3*. Isso evita as requisições extra a servidores para explorar a associação recursiva. Em *BSBM_3*, *Metis-2hops* produz um número maior de requisições por causa da otimização para os nós de alto-grau.



(a) Q2

(b) Q3

(c) Q5

(d) Q7

Figura 6.3: Escalabilidade de servidores

6.3.2.2 Escalabilidade de servidores

Na variação do número de servidores optou-se por utilizar o *dataset* BSB_5. Os resultados são mostrados na Figura 6.3. Em geral, o aumento do número de servidores proporciona os benefícios do processamento paralelo e reduz a carga dos servidores. No entanto, este aumento também pode levar a uma maior distribuição de dados entre os

servidores quando os itens de dados de consulta não estão definidos para serem agrupados. Acredita-se que o elevado número de requisições que estão sendo executadas por cada *thread* em paralelo eleva a competição por recursos e afeta o desempenho do sistema. O pior resultado relacionado a este efeito é observado na $Q3$ em um *cluster* de 8 de servidor para *METIS-2hops*, onde cada *thread* necessita acessar todos os servidores. Note que o efeito do processamento paralelo somente consegue reduzir o tempo de resposta quando a capacidade do sistema é aumentada para 12 servidores fazendo com que o número de requisições a servidores permaneça estável.

CAPÍTULO 7

CONCLUSÃO

Esta tese apresentou uma abordagem para o particionamento de dados baseada em informações da carga de trabalho e restrições de transferência e armazenamento de um sistema distribuído. O problema do particionamento de dados foi dividido em dois subproblemas: fragmentação e alocação. Na fragmentação, a solução proposta analisa as informações da carga de trabalho disponíveis para identificar itens de dados que devem ser mantidos em conjunto e os atribui a um mesmo fragmento de acordo com um limiar de armazenamento. Na alocação, são consideradas as afinidades que relacionam dados em fragmentos distintos para definir como os fragmentos devem ser agrupados. A solução está focada em prover um particionamento adequado que cobre o padrão de acesso das consultas mais frequentes de uma carga de trabalho. Como resultado, é possível maximizar a execução paralela de consultas capazes de executar localmente e minimizar o número de consultas envolvendo dados distribuídos por diversos servidores. Este trabalho contribui para o contexto de bancos de dados largamente distribuídos, nos quais os custos de comunicação devem ser reduzidos para prover um serviço escalável. Os estudos experimentais realizados indicam que a solução proposta para o particionamento de dados é capaz de melhorar o desempenho de consultas se comparado a trabalhos relacionados.

Os trabalhos futuros incluem uma série de problemas considerados adjacentes a proposta desta tese, e que também estão relacionados a escalabilidade de sistemas de banco de dados em nuvem. Como atividade imediata, pretende-se estender este trabalho para dar suporte a cargas de trabalho dinâmicas. Além disto, o estudo de estratégias de replicação devem ser investigadas, tanto para melhoria de performance sobre uma base particionada, quanto para suportar diferentes cargas de trabalho. Outros problemas incluem a gestão de meta-dados e a otimização de consultas.

PUBLICAÇÕES REALIZADAS NO DOUTORADO

A lista a seguir registra os artigos publicados e os artigo submetidos durante o período deste doutorado e referentes ao tema desta tese.

1. R. R. M. Penteado, R. Schroeder, D. Hoss, J. Nande, R. M. Maeda, W. O. Couto and C. S. Hara. Um Estudo sobre Bancos de Dados em Grafos Nativos. *Escola Regional de Banco de Dados (ERBD)*, Abr. 2014.

2. R. Schroeder e C. S. Hara. Clustering RDF Data by Affinity Relations. *International Conference on Management of Data (SIGMOD)*, submetido em Dez. 2013.

3. R. Schroeder, R. R. M. Penteado and C. S. Hara. Partitioning RDF Exploiting Workload Information. *International Conference on World Wide Web Companion (WWW)*, pp. 213-214, Mai. 2013.

4. R. Schroeder and C. S. Hara. Towards Full-fledged XML Fragmentation for Transactional Distributed Databases. *Workshop de Teses e Dissertações em Banco de Dados*, Out. 2012.

5. R. Schroeder and C. S. Hara. Affinity-based XML Fragmentation. *International Workshop on the Web and Databases (WebDB: co-located with SIGMOD)*, pp. 61-66, Mai. 2012

6. D. E. M. Arnaut, R. Schroeder and C. S. Hara. Phoenix: A Relational Storage Component for the Cloud. *International Conference on Cloud Computing (IEEE Cloud)*, pp. 684-691, Jul. 2011

REFERÊNCIAS

[bam, 2012] (2012). Bamboodb. Available at: http://www.bamboodb.com/.

[hyp, 2012] (2012). Hypertable. Available at: http://www.hypertable.org/.

[mon, 2012] (2012). Mongodb. Available at: http://www.mongodb.org/.

[red, 2012] (2012). Redis. Available at: http://redis.io/.

[met, 2013] (2013). METIS. Available at: http://glaros.dtc.umn.edu/gkhome/views/metis.

[Abadi et al., 2009] 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.

[Abiteboul et al., 2008] Abiteboul, S., Manolescu, I., Polyzotis, N., Preda, N., and Sun, C. (2008). Xml processing in dht networks. In *Proceedings of the IEEE 24th ICDE*, pages 606–615.

[Abou-Rjeili and Karypis, 2006] Abou-Rjeili, A. and Karypis, G. (2006). Multilevel algorithms for partitioning power-law graphs. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 124–124, Washington, DC, USA. IEEE Computer Society.

[Agrawal et al., 2013] Agrawal, D., Das, S., and Abbadi, E. A. (2013). *Data Management in the Cloud: Challenges and Opportunities*.

[Agrawal et al., 2004] Agrawal, S., Narasayya, V., and Yang, B. (2004). Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of ACM SIGMOD*, pages 359–370.

[Amazon, 2012] Amazon (2012). Amazon webservices. Available at: http://aws.amazon.com/.

[Andrade et al., 2006] Andrade, A., Ruberg, G., Baião, F. A., Braganholo, V. P., and Mattoso, M. (2006). Efficiently processing xml queries over fragmented repositories with partix. In *EDBT Workshops*, pages 150–163.

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

[Apache, 2012a] Apache (2012a). Couchdb. Available at: http://couchdb.apache.org/.

[Apache, 2012b] Apache (2012b). Hbase. Available at: http://hadoop.apache.org/hbase/.

[Arnaut et al., 2011] Arnaut, D. E. M., Schroeder, R., and Hara, C. S. (2011). Phoenix - A Relational Storage Component for the Cloud. In *4th IEEE CLOUD*, pages 684 – 691.

[Aslett, 2011] Aslett, M. (2011). How will the database incumbents respond to nosql and newsql? The 451 Group.

[AWS, 2012] AWS, A. (2012). Simpledb. Available at: http://aws.amazon.com/simpledb/.

[Aykanat et al., 2007] Aykanat, C., Cambazoglu, B. B., Findik, F., and Kurc, T. (2007). Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids. *J. Parallel Distrib. Comput.*, 67(1):77–99.

[Aykanat et al., 2008] Aykanat, C., Cambazoglu, B. B., and Uçar, B. (2008). Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. *J. Parallel Distrib. Comput.*, 68(5):609–625.

[Baker et al., 2011] Baker, J., Bond, C., Corbett, J., Furman, J. J., Khorlin, A., Larson, J., Leon, J.-M., Li, Y., Lloyd, A., and Yushprakh, V. (2011). Megastore: Providing scalable, highly available storage for interactive services. In *CIDR*, pages 223–234.

[Bellatreche and Benkrid, 2009] Bellatreche, L. and Benkrid, S. (2009). A joint design approach of partitioning and allocation in parallel data warehouses. In *Proceedings*

of the 11th International Conference on Data Warehousing and Knowledge Discovery, DaWaK '09, pages 99–110, Berlin, Heidelberg. Springer-Verlag.

[Bellatreche et al., 2011] Bellatreche, L., Benkrid, S., Ghazal, A., Crolotte, A., and Cuzzocrea, A. (2011). Verification of partitioning and allocation techniques on teradata dbms. In *Proceedings of the 11th international conference on Algorithms and architectures for parallel processing - Volume Part I*, ICA3PP'11, pages 158–169, Berlin, Heidelberg. Springer-Verlag.

[Bellatreche and Boukhalfa, 2005] Bellatreche, L. and Boukhalfa, K. (2005). An evolutionary approach to schema partitioning selection in a data warehouse. In *Proceedings of the 7th international conference on Data Warehousing and Knowledge Discovery*, DaWaK'05, pages 115–125, Berlin, Heidelberg. Springer-Verlag.

[Bizer and Schultz, 2009] Bizer, C. and Schultz, A. (2009). The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems*, 5(2):1–24.

[Bonifati and Cuzzocrea, 2007] Bonifati, A. and Cuzzocrea, A. (2007). Efficient fragmentation of large xml documents. In *Proceedings of the international conference on Database and Expert Systems Applications*, pages 539–550. Springer-Verlag.

[Bonifati et al., 2004] Bonifati, A., Matrangolo, U., Cuzzocrea, A., and Jain, M. (2004). Xpath lookup queries in p2p networks. In *Proceedings of the 6th annual ACM international workshop on Web information and data management*, WIDM '04, pages 48–55, New York, NY, USA. ACM.

[Bordawekar and Shmueli, 2004] Bordawekar, R. and Shmueli, O. (2004). Flexible workload-aware clustering of xml documents. In *Database and XML Technologies*, Lecture Notes in Computer Science, pages 346–357. Springer Berlin / Heidelberg.

[Bordawekar and Shmueli, 2008] Bordawekar, R. and Shmueli, O. (2008). An algorithm for partitioning trees augmented with sibling edges. *Inf. Process. Lett.*, 108(3):136–142.

[Brantner et al., 2008] Brantner, M., Florescu, D., Graf, D., Kossmann, D., and Kraska, T. (2008). Building a database on s3. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 251–264, New York, NY, USA. ACM.

[Brewer, 2000] Brewer, E. A. (2000). Towards robust distributed systems (abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, pages 7–, New York, NY, USA. ACM.

[Brocheler et al., 2010] Brocheler, M., Pugliese, A., and Subrahmanian, V. S. (2010). Cosi: Cloud oriented subgraph identification in massive social networks. In *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '10, pages 248–255, Washington, DC, USA. IEEE Computer Society.

[Bunch and et al, 2010] Bunch, C. and et al (2010). An Evaluation of Distributed Datastores Using the AppScale Cloud Platform. In *IEEE Cloud10: International Conference on Cloud Computing*, pages 305–312.

[Buyya et al., 2008] Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, HPCC '08, pages 5–13, Washington, DC, USA. IEEE Computer Society.

[Cattell, 2011] Cattell, R. (2011). Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27.

[Ceri and Pelagatti, 1982] Ceri, S. and Pelagatti, G. (1982). Allocation of operations in distributed database access. *IEEE Trans. Computers*, 31(2):119–129.

[Ceri and Pelagatti, 1984] Ceri, S. and Pelagatti, G. (1984). *Distributed databases principles and systems*. McGraw-Hill, Inc., New York, NY, USA.

[Ceri et al., 1987] Ceri, S., Pernici, B., and Wiederhold, G. (1987). Distributed database design methodologies. *Proceedings of the IEEE*, 75(5):533 – 546.

[Chang and et al, 2006] Chang, F. and et al (2006). Bigtable: a distributed storage system for structured data. In *Symposium on Operating Systems Design and Implementation*, page 15. USENIX Association.

[Chong et al., 2005] Chong, E. I., Das, S., Eadon, G., and Srinivasan, J. (2005). An efficient sql-based rdf querying scheme. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1216–1227. VLDB Endowment.

[Cong et al., 2007] Cong, G., Fan, W., and Kementsietsidis, A. (2007). Distributed query evaluation with performance guarantees. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, page 509, New York, New York, USA. ACM Press.

[Cooper and et al, 2008] Cooper, B. F. and et al (2008). PNUTS: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288.

[Curino and et al, 2011] Curino, C. and et al (2011). Relational Cloud: A Database-as-a-Service for the Cloud. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, pages 235–240.

[Curino et al., 2010] Curino, C., Jones, E., Zhang, Y., and Madden, S. (2010). Schism: a workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3:48–57.

[Cuzzocrea et al., 2009] Cuzzocrea, A., Darmont, J., and Mahboubi, H. (2009). Fragmenting very large xml data warehouses via kmeans clustering algorithm. *Int. J. Bus. Intell. Data Min.*, 4:301–328.

[Das et al., 2010] Das, S., Agrawal, D., and El Abbadi, A. (2010). G-store: a scalable data store for transactional multi key access in the cloud. In *Proceedings of the 1st*

ACM symposium on Cloud computing, SoCC '10, pages 163–174, New York, NY, USA. ACM.

[DeCandia et al., 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220.

[Egger, 2009] Egger, D. (2009). SQL in the Cloud. Master's thesis, Swiss Federal Institute of Technology Zurich (ETH).

[Fan, 2012] Fan, W. (2012). Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 8–21, New York, NY, USA. ACM.

[Fiduccia and Mattheyses, 1982] Fiduccia, C. M. and Mattheyses, R. M. (1982). A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA. IEEE Press.

[Figueiredo et al., 2010] Figueiredo, G., Braganholo, V. P., and Mattoso, M. (2010). Processing queries over distributed xml databases. *Journal of Information and Data Management*, 3(1):455–470.

[Franz Inc, 2013] Franz Inc (2013). AllegroGraph 4.11. Available at: http://www.franz.com/agraph/allegrograph/.

[G., 1954] G. (1954). Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404.

[Galárraga et al., 2014] Galárraga, L., Hose, K., and Schenkel, R. (2014). Partout: A distributed engine for efficient rdf processing. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, WWW Companion '14, pages 267–268, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Galbiati, 2011] Galbiati, G. (2011). Approximating minimum cut with bounded size. In *Proceedings of the 5th international conference on Network optimization*, INOC'11, pages 210–215, Berlin, Heidelberg. Springer-Verlag.

[Gertz and Bremer, 2003] Gertz, M. and Bremer, J. M. (2003). Distributed xml repositories: Top-down design and transparent query processing. Technical report, Departement of Computer Science,University of California, Davis, CA, USA.

[Goldschmidt and Hochbaum, 1988] Goldschmidt, O. and Hochbaum, D. S. (1988). Polynomial algorithm for the k-cut problem. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 444–451, Washington, DC, USA. IEEE Computer Society.

[Google, 2012] Google (2012). Amazon webservices. Available at: http://cloud.google.com/appengine/.

[Guttmann-Beck and Hassin, 2000] Guttmann-Beck, N. and Hassin, R. (2000). Approximation algorithms for minimum k-cut. *Algorithmica*, 27(2):198–207.

[Hauck and Borriello, 1995] Hauck, S. and Borriello, G. (1995). An evaluation of bipartitioning techniques. In *Proceedings of the 16th Conference on Advanced Research in VLSI (ARVLSI'95)*, ARVLSI '95, pages 383–, Washington, DC, USA. IEEE Computer Society.

[Hauglid et al., 2010] Hauglid, J. O., Ryeng, N. H., and Nørvåg, K. (2010). Dyfram: dynamic fragmentation and replica management in distributed database systems. *Distrib. Parallel Databases*, 28(2-3):157–185.

[Hendrickson et al., 1997] Hendrickson, B., Leland, R. W., and Driessche, R. V. (1997). Skewed graph partitioning. In *PPSC*.

[Hose and Schenkel, 2013] Hose, K. and Schenkel, R. (2013). Warp: Workload-aware replication and partitioning for rdf. In *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on*, pages 1–6.

[Jiewen Huang, 2011] Jiewen Huang, D. J. A. (2011). Scalable SPARQL Querying of Large RDF Graphs. *PVLDB*, 4(11):1123–1134.

[Kanne and Moerkotte, 2006] Kanne, C.-C. and Moerkotte, G. (2006). A linear time algorithm for optimal tree sibling partitioning and approximation algorithms in natix. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 91–102. VLDB Endowment.

[Karypis and Kumar, 1998] Karypis, G. and Kumar, V. (1998). Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1):96–129.

[Karypis and Kumar, 1999] Karypis, G. and Kumar, V. (1999). Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 343–348, New York, NY, USA. ACM.

[Kernighan and Lin, 1970] Kernighan, B. W. and Lin, S. (1970). An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1):291–307.

[Khan and Hoque, 2010] Khan, S. I. and Hoque, A. S. M. L. (2010). A New Technique for Database Fragmentation in Distributed Systems. *International Journal of Computer Applications*, 5(9):20–24.

[Kling et al., 2010] Kling, P., Özsu, M. T., and Daudjee, K. (2010). Distributed xml query processing: Fragmentation, localization and pruning. Technical report, University of Waterloo.

[Klophaus, 2010] Klophaus, R. (2010). Riak core: building distributed applications without shared state. In *ACM SIGPLAN Commercial Users of Functional Programming*, CUFP '10, pages 14:1–14:1, New York, NY, USA. ACM.

[Kossmann, 2000] Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469.

[Kossmann et al., 2010] Kossmann, D., Kraska, T., and Loesing, S. (2010). An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the SIGMOD '10*, pages 579–590.

[Lakshman and Malik, 2009] Lakshman, A. and Malik, P. (2009). Cassandra: structured storage system on a p2p network. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, PODC '09, pages 5–5, New York, NY, USA. ACM.

[Lamport, 1998] Lamport, L. (1998). The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169.

[Laney, 2001] Laney, D. (2001). 3D Data Management: Controlling Data Volume, Velocity and Variety. Technical report, Gartner Inc.

[Laney, 2012] Laney, D. (2012). The Importance of 'Big Data': A Definition. Technical report, Gartner Inc.

[Ma and Schewe, 2003] Ma, H. and Schewe, K.-D. (2003). Fragmentation of xml documents. In *SBBD*, pages 200–214.

[McCormick et al., 1972] McCormick, W., Schweitzer, P., and White, T. (1972). Problem decomposition and data reorganization by a clustering technique. *Oper. Res.*, 20:993–1009.

[Mulay and Kumar, 2012] Mulay, K. and Kumar, P. S. (2012). Spovc: A scalable rdf store using horizontal partitioning and column oriented dbms. In *Proceedings of the 4th International Workshop on Semantic Web Information Management*, SWIM '12, pages 8:1–8:8, New York, NY, USA. ACM.

[Navathe et al., 1984] Navathe, S., Ceri, S., Wiederhold, G., and Dou, J. (1984). Vertical partitioning of algorithms for database design. *ACM Trans. Database Syst.*, 9:680–710.

[Navathe and Ra, 1989] Navathe, S. and Ra, M. (1989). Vertical partitioning for database design: a graphical algorithm. *ACM SIGMOD International Conference on Management of Data*, 18:440–450.

[Nehme and Bruno, 2011] Nehme, R. and Bruno, N. (2011). Automated partitioning design in parallel database systems. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 1137–1148, New York, NY, USA. ACM.

[Noaman and Barker, 1999] Noaman, A. Y. and Barker, K. (1999). A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse. In *Proceedings of the eighth international conference on Information and knowledge management*, CIKM '99, pages 154–161, New York, NY, USA. ACM.

[Ozsu et al., 2013] Ozsu, M. T., Daudjee, K., and Hartig, O. (2013). chameleon-db: a Workload-Aware Robust RDF Data Management System. Technical Report iv, University of Waterloo.

[Ozsu and Valduriez, 2011] Ozsu, T. and Valduriez, P. (2011). *Principles of Distributed Database Systems*. Third edition. Prentice-Hall.

[Papadimitriou and Steiglitz, 1998] Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover.

[Papadomanolakis and Ailamaki, 2004] Papadomanolakis, S. and Ailamaki, A. (2004). Autopart: Automating schema design for large scientific databases using data partitioning. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, SSDBM '04, pages 383–, Washington, DC, USA. IEEE Computer Society.

[Paparizos et al., 2004] Paparizos, S., Wu, Y., Lakshmanan, L. V. S., and Jagadish, H. V. (2004). Tree logical classes for efficient evaluation of xquery. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 71–82, New York, NY, USA. ACM.

[Pavlo et al., 2012] Pavlo, A., Curino, C., and Zdonik, S. B. (2012). Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *SIGMOD'12*, pages 61–72.

[Pérez et al., 2009] Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45.

[Pujol et al., 2010] Pujol, J. M., Erramilli, V., Siganos, G., Yang, X., Laoutaris, N., Chhabra, P., and Rodriguez, P. (2010). The little engine(s) that could: scaling online social networks. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 375–386, New York, NY, USA. ACM.

[Rao et al., 2011] Rao, J., Shekita, E. J., and Tata, S. (2011). Using paxos to build a scalable, consistent, and highly available datastore. *Proc. VLDB Endow.*, 4(4):243–254.

[Ribas et al., 2011] Ribas, E. A., Uba, R., Reinaldo, A. P., de Campos Jr., A., Arnaut, D., and Hara, C. (2011). Layering a dbms on a dht-based storage engine. *Journal of Information and Data Management*, 2(1):59–66.

[Rowstron and Druschel, 2001] Rowstron, A. I. T. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK. Springer-Verlag.

[Sacca and Wiederhold, 1985] Sacca, D. and Wiederhold, G. (1985). Database partitioning in a cluster of processors. *ACM Trans. Database Syst.*, 10(1):29–56.

[Saran and Vazirani, 1995] Saran, H. and Vazirani, V. V. (1995). Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108.

[Schloegel et al., 2000] Schloegel, K., Karypis, G., and Kumar, V. (2000). A unified algorithm for load-balancing adaptive scientific simulations. In *Proceedings of the 2000*

ACM/IEEE conference on Supercomputing (CDROM), Supercomputing '00, Washington, DC, USA. IEEE Computer Society.

[Schütt et al., 2008] Schütt, T., Schintke, F., and Reinefeld, A. (2008). Scalaris: reliable transactional p2p key/value store. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, ERLANG '08, pages 41–48, New York, NY, USA. ACM.

[Sengupta, 2008] Sengupta, S. (2008). Sql data services: A lap around. In *Microsoft Professional Developers Conference*.

[Shnaiderman and Shmueli, 2009] Shnaiderman, L. and Shmueli, O. (2009). ipixsar: incremental clustering of indexed xml data. In *Proceedings of the 2009 EDBT/ICDT Workshops*, EDBT/ICDT '09, pages 74–84, New York, NY, USA. ACM.

[Shnaiderman et al., 2008] Shnaiderman, L., Shmueli, O., and Bordawekar, R. (2008). Pixsar: incremental reclustering of augmented xml trees. In *Proceedings of the 10th ACM workshop on Web information and data management*, WIDM '08, pages 17–24, New York, NY, USA. ACM.

[Son and Kim, 2004] Son, J. H. and Kim, M. H. (2004). An adaptable vertical partitioning method in distributed systems. *J. Syst. Softw.*, 73(3):551–561.

[Stoica et al., 2003] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., and Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17 – 32.

[Tatarowicz et al., 2012] Tatarowicz, A., Curino, C., Jones, E., and Madden, S. (2012). Lookup tables: Fine-grained partitioning for distributed databases. In *Inter. Conference on Data Engineering*.

[V. et al., 2009] V., C. U., G., B. E., D., D. K., D., B., T., H. R., and A, R. L. (2009). A repartitioning hypergraph model for dynamic load balancing. *J. Parallel Distrib. Comput*, 69(8):711–724.

[Valduriez and Pacitti, 2004] Valduriez, P. and Pacitti, E. (2004). Data management in large-scale p2p systems. In *6th International Conference VECPAR*, pages 104–118.

[Voldemort, 2012] Voldemort, P. (2012). Project voldemort: A distributed database. Available at: http://project-voldemort.com/.

[Wehrle et al., 2005] Wehrle, P., Miquel, M., and Tchounikine, A. (2005). A model for distributing and querying a data warehouse on a computing grid. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems - Volume 01*, ICPADS '05, pages 203–209, Washington, DC, USA. IEEE Computer Society.

[Weiss and Zgorski, 2012] Weiss, R. and Zgorski, L. (2012). Obama administration unveils big data initiative: Announces $200 million in new r&d investments. Available at: http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_press_release_final_2.pdf.

[Wilkinson et al., 2003] Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D., and Database, J. (2003). Efficient rdf storage and retrieval in jena2. In *EXPLOITING HYPERLINKS 349*, pages 35–43.

[Wu et al., 2011] Wu, E., Curino, C., and Madden, S. (2011). No bits left behind. In *5th Biennial Conference on Innovative Data Systems Research*.

[Yang and Wu, 2013] Yang, M. and Wu, G. (2013). A workload-based partitioning scheme for parallel rdf data processing. In Li, J., Qi, G., Zhao, D., Nejdl, W., and Zheng, H.-T., editors, *Semantic Web and Web Science*, Springer Proceedings in Complexity, pages 311–324. Springer New York.

[Yang et al., 2012] Yang, S., Yan, X., Zong, B., and Khan, A. (2012). Towards effective partition management for large graphs. In *SIGMOD'12*, pages 517–528, New York, NY, USA. ACM.

[Yang et al., 2013] Yang, T., Chen, J., Wang, X., Chen, Y., and Du, X. (2013). Efficient sparql query evaluation via automatic data partitioning. In Meng, W., Feng, L.,

Bressan, S., Winiwarter, W., and Song, W., editors, *Database Systems for Advanced Applications*, volume 7826 of *Lecture Notes in Computer Science*, pages 244–258. Springer Berlin Heidelberg.

[Zeng et al., 2013] Zeng, K., Yang, J., Wang, H., Shao, B., and Wang, Z. (2013). A distributed graph engine for web scale rdf data. *Proc. VLDB Endow.*, 6(4):265–276.

[Zilio, 1998] Zilio, D. C. (1998). *Physical Database Design Decision Algorithms and Concurrent Reorganization for Parallel Database Systems.* PhD thesis, University of Toronto.

[Zilio et al., 2004] Zilio, D. C., Rao, J., Lightstone, S., Lohman, G., Storm, A., Garcia-Arellano, C., and Fadden, S. (2004). Db2 design advisor: integrated automatic physical database design. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, pages 1087–1097. VLDB Endowment.

[Zuse Institute Berlin, 2012] Zuse Institute Berlin (2012). Scalaris - distributed transactional key-value store. Available at http://code.google.com/p/scalaris/.

[Çatalyürek and Aykanat, 1999] Çatalyürek, Ü. and Aykanat, C. (1999). Patoh: partitioning tool for hypergraphs. Technical report, Technical Report, Department of Computer Engineering, Bilkent University.