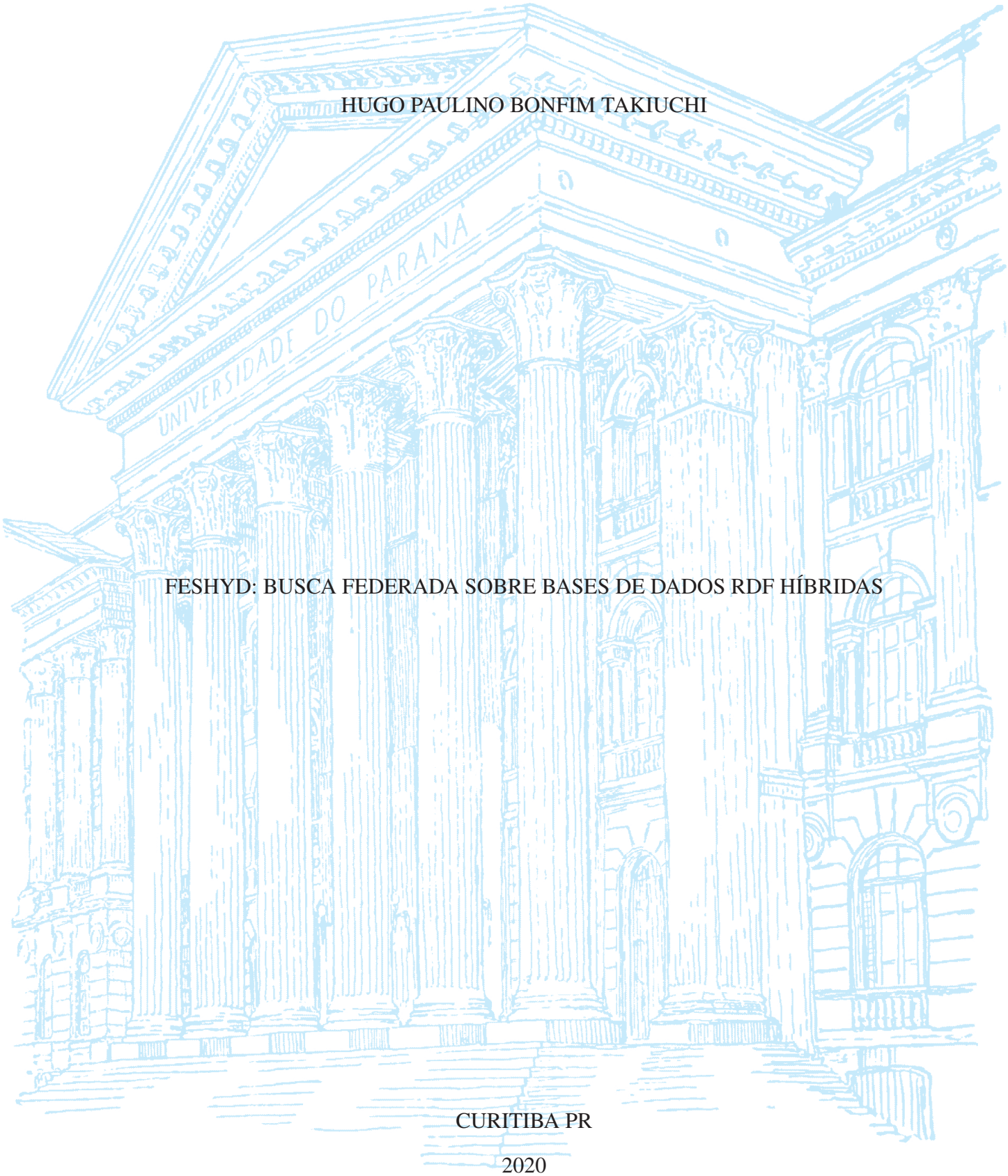
UNIVERSIDADE FEDERAL DO PARANÁ

HUGO PAULINO BONFIM TAKIUCHI



FESHYD: BUSCA FEDERADA SOBRE BASES DE DADOS RDF HÍBRIDAS

CURITIBA PR

2020

HUGO PAULINO BONFIM TAKIUCHI

FESHYD: BUSCA FEDERADA SOBRE BASES DE DADOS RDF HÍBRIDAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carmem Satie Hara.

Coorientador: Raqueline Ritter de Moura Penteado.

CURITIBA PR

2020

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

T136f Takiuchi, Hugo Paulino Bonfim
FESHYD [recurso eletrônico]: busca federada sobre bases de dados RDF híbridas / Hugo Paulino Bonfim Takiuchi. – Curitiba, 2020.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2020.

Orientadora: Carmem Satie Hara.
Coorientadora: Raqueline Ritter de Moura Penteado.

1. Banco de dados distribuído. 2. Web semântica. I. Universidade Federal do Paraná. II. Hara, Carmem Satie. III. Penteado, Raqueline Ritter de Moura. IV. Título.

CDD: 006

Bibliotecária: Vanusa Maciel CRB- 9/1928

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **HUGO PAULINO BONFIM TAKIUCHI** intitulada: **FESHYD: Busca Federada sobre Bases de Dados RDF Híbridas**, sob orientação da Profa. Dra. CARMEM SATIE HARA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 04 de Setembro de 2020.

Assinatura Eletrônica
07/09/2020 10:11:08.0
CARMEM SATIE HARA
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
05/09/2020 19:03:53.0
ROBERTO PEREIRA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
08/09/2020 14:10:11.0
RITA CRISTINA GALARRAGA BERARDI
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ)

A Nair, Mitsuru e Seiji

AGRADECIMENTOS

Agradeço a minha orientadora Carmem Hara e a minha coorientadora Raqueline Penteado por me auxiliarem durante todo o mestrado. Agradeço por todas as orientações, pela paciência e por acreditarem no meu trabalho. Obrigado por me apoiarem no meu crescimento profissional e no crescimento pessoal.

Agradeço a minha banca da defesa de mestrado por aceitarem o convite e pelas ótimas observações feitas na minha dissertação. A minha banca foi composta pelo professor Roberto Pereira e pela professora Rita Berardi. Também agradeço ao professor Eduardo Almeida e à professora Rita Berardi pelas considerações feitas na banca de qualificação.

Agradeço também a Capes pelo apoio com a concessão da bolsa de mestrado.

Um agradecimento aos meus familiares e amigos que me apoiaram durante o mestrado. Me ajudando no meu crescimento pessoal e profissional. Sem eles não conseguiria chegar tão longe e finalizar o mestrado.

Começando pela minha família, queria fazer um agradecimento muito especial ao meu pai Mitsuru e a minha mãe Nair pelo apoio incondicional. Sempre estiveram comigo mesmo morando longe, sempre me apoiaram de todas as formas possíveis. Mesmo quando sozinho sabia que eles estavam torcendo por mim. Então muito obrigado por toda a ajuda que me deram para chegar até aqui, sem eles nada disso seria possível. Sempre me ajudando a me tornar uma pessoa melhor e me fazendo continuar nos momentos mais complicados. Agradeço também ao meu irmão Seiji pelo apoio e amizade durante o mestrado.

Por fim, agradeço aos meus amigos. Um agradecimento especial a Acácia Terra pelo apoio durante o mestrado. Sua amizade me ajudou em momentos difíceis. Ela me ajudou muito no meu crescimento pessoal. Também me apoiou no meu crescimento profissional, sendo uma ótima colega de trabalho. Um agradecimento especial a Mariana Garcez que me apoiou em momentos difíceis durante o mestrado. Sua amizade me manteve equilibrado em momentos complicados. Também uma ótima colega de trabalho. Um agradecimento a minha psicóloga Rafaella Wotecoski, que me apoiou em muitos momentos com vários conselhos fundamentais para o meu desenvolvimento.

RESUMO

Na Web Semântica, os dados são disponibilizados no formato RDF e consultados por meio da linguagem SPARQL. A maioria dos processadores de consultas consideram apenas bases RDF federadas ou apenas bases proprietárias. Bases federadas consistem de um conjunto de repositórios autônomos, enquanto bases proprietárias permitem acesso irrestrito, tanto aos dados quanto ao processamento interno da consulta. Caso uma consulta envolva tanto dados de bases de terceiros autônomas bem como dados da base proprietária, existem duas alternativas para o seu processamento: (i) tratar a base proprietária como um componente da base federada; (ii) intervenção do usuário para integrar os dados de base proprietária e federada. Embora ambas permitam integração de dados da base, elas não exploram otimizações que são possíveis pelo fato de haver acesso irrestrito à base proprietária. Esta questão é tratada nesta dissertação, com a proposta de uma terceira alternativa, denominada de FeSHyD, que processa consultas SPARQL tanto sobre bases federadas quanto proprietárias distribuídas. O FeSHyD gera um plano de consultas otimizado, que é executado em paralelo por todos os servidores que compõem a base proprietária. Durante a geração do plano, a otimização envolve métodos para a seleção das fontes e para a ordenação dos blocos que compõem o plano de consulta, de forma que a base proprietária seja explorada antes de submeter subconsultas às bases de terceiros. Durante o processamento da consulta, os servidores da base proprietária submetem estas subconsultas à base federada diretamente, sem a existência de um ponto central de controle. O sistema foi implementado e os resultados experimentais mostram que ele reduz o tempo de processamento de consultas em até 45% comparado à alternativa de tratar a base proprietária como um componente de uma base federada.

Palavras-chave: busca federada, consulta SPARQL, bases de dados híbridas distribuídas, integração de sistemas distribuídos, seleção de fontes, ordenação das subconsultas.

ABSTRACT

In the Semantic Web, data is made available in RDF format and queried using the SPARQL language. Most query processors consider only federated RDF bases or only proprietary bases. Federated databases consist of a set of autonomous repositories, while proprietary databases allow unrestricted access, both to data and to query processing execution alternatives. If a query involves both data from autonomous third party databases as well as data from the proprietary database, there are two alternatives for processing it: (i) consider the proprietary base as a component of the federated database; (ii) rely on user intervention to integrate the proprietary and federated databases. Although both alternatives promote data integration, they do not explore optimizations that are possible by the fact that there is unrestricted access to the proprietary base. This issue is addressed in this dissertation, with the proposal of a third alternative, called FeSHyD, which processes SPARQL queries on both federated and distributed proprietary bases. FeSHyD generates an optimized query plan that is executed in parallel by all servers that compose the proprietary database. During the generation of the plan, the optimization involves methods for selecting external data sources, and for ordering the blocks that compose the query plan such that the proprietary base is explored before subqueries are submitted to external sources. During query processing, these subqueries are sent to third party databases directly by the servers, without relying on a central control point. The system was implemented and the experimental results show that it reduces query processing time by up to 45% compared to the alternative of considering the proprietary base as a component of a federated database.

Keywords: federated search, SPARQL query, distributed hybrid databases, distributed system integration, source selection, subquery ordering.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Parte dos dados publicados na Internet atualmente segue os padrões da Web Semântica[1], que é uma extensão da Web de documentos. Os objetivos da Web Semântica são: (i) produzir documentos interpretáveis por máquina bem como por humanos, visando a automação na análise dos recursos; (i) conseguir integrar dados dispersos na Web (Hendler, 2001). Essa integração leva em consideração o conceito de dados ligados. São criadas ligações entre informações de diferentes locais com a finalidade de facilitar a busca por informações relevantes. Os dados na Web Semântica estão publicados no formato RDF (*Resource Description Framework*) e acessados via a linguagem de consulta SPARQL *(SPARQL Protocol and RDF Query Language)*, ambos definidos pelo consórcio W3C.

Uma base de dados RDF consiste de um conjunto de triplas *(sujeito, predicado, objeto)*, que pode ser visualizado como um grafo direcionado, no qual sujeitos e objetos são representados por vértices e ligados através de arestas que representam predicados[2]. Um modo de armazenar e organizar os dados RDF é depositar todas as triplas em um único servidor de dados. Este modo é denominado como modelo de armazenamento centralizado. Atualmente algumas bases de dados apresentam um grande volume e comportam até 1 trilhão de triplas[3]. Com a finalidade de dar suporte a este grande volume de dados, alguns trabalhos propõem a utilização do armazenamento de dados RDF distribuído em diferentes servidores. Uma abordagem para acessar os dados distribuídos é conhecido como busca federada, na qual um moderador é responsável por coordenar o processamento e execução da consulta sobre bases de dados autônomas (Rakhmawati et al., 2013). As bases autônomas podem pertencer a terceiros. Neste caso, o moderador só tem acesso à base através de consultas SPARQL. Por outro lado, as bases proprietárias, que pertencem à própria aplicação, permitem otimizações durante a execução interna da consulta. Ou seja, bases de terceiros são acessadas como caixas-pretas, enquanto bases proprietárias podem ser acessadas como caixas-brancas[4].

Alguns sistemas são baseados em consultas federadas sobre bases de dados autônomas exclusivamente. oLinda (da Cunha e Lóscio, 2014) , SPLENDID (Görlitz e Staab, 2011), Lusail (Abdelaziz et al., 2017a), Ephedra (Nikolov et al., 2017) e FedX (Schwarte et al., 2011) são exemplos destes sistemas. Cada trabalho apresenta uma contribuição no contexto de bases RDF federadas. Por exemplo, para a execução de uma consulta, estes sistemas devem detectar as bases de dados envolvidas na execução da consulta. Neste sentido, oLinDa apresenta uma técnica de decomposição de consultas baseada em esquemas estruturais. SPLENDID utiliza metadados sobre as bases e consultas SPARQL na seleção de fontes. O sistema Lusail, por outro lado, propõe o uso de variáveis de junção entre padrões de triplas para determinar bases de dados relevantes. Já o Ephedra propõe considerar modelos de dados diferentes e interfaces de consultas variadas no processamento da busca. Por fim, o sistema FedX considera bases de dados mistas e opera sobre bases de terceiros distribuídas e centralizadas.

Diferentemente dos sistemas que utilizam exclusivamente bases de dados autônomas, SIHJoin (Ladwig e Tran, 2011) propõe o processamento de consultas federadas sobre bases de terceiros distribuídas e uma base proprietária centralizada. Neste sistema são propostas

[1] `https://www.w3.org/standards/semanticweb/`

[2] `https://www.w3.org/TR/rdf11-primer/`

[3] `https://www.w3.org/wiki/LargeTripleStores`

[4] Os termos caixa-branca e caixa-preta são utilizados com significado similar às técnicas de geração de casos de teste: que exploram a estrutura interna do programa e que utilizam apenas entrada e saída, respectivamente

otimizações sobre a execução da consulta na base centralizada. Dada a utilização do modelo centralizado, a quantidade de processamento e memória disponíveis é limitado se comparado ao modelo de armazenamento distribuído sobre diversos servidores.

Nessa dissertação é proposto o sistema FeSHyD *(**Fe**derated **S**earch on **Hy**brid **D**atabases)*, que viabiliza o processamento de consultas SPARQL sobre uma base de dados distribuída e híbrida. O termo híbrido refere-se à composição da base do sistema, com uma base proprietária distribuída, tratada como caixa-branca, e bases de terceiros, acessadas como caixas-pretas por meio de *endpoints* SPARQL. A abordagem de processamento proposta permite a comunicação direta dos servidores da base proprietária com *endpoints* SPARQL durante o processamento de consultas, explorando o processamento paralelo entre os servidores durante a execução de consultas. Nessa otimização, cada servidor é responsável por elaborar subconsultas SPARQL, submeter essas subconsultas aos *endpoints* remotos e juntar os resultados. O objetivo é diminuir a sobrecarga no processamento da consulta realizado pelo moderador do sistema e distribuí-la entre os servidores da base proprietária. Para realizar o processamento sobre uma base híbrida o sistema FeSHyD primeiro analisa a consulta SPARQL, requisitada pelo usuário, e a divide entre a base proprietária e as bases de terceiros. Para determinar as bases de dados relevantes à consulta o sistema explora metadados mantidos sobre a base de dados híbrida. Depois da execução da consulta pelos servidores da base proprietária, os resultados são unidos e enviados ao usuário. Vale destacar que a abordagem proposta oculta dos usuários os detalhes envolvidos na execução de consultas.

Um estudo experimental comparou a abordagem de processamento de consultas do FeSHyD com uma abordagem *baseline* que adota a base proprietária como caixa-preta, assim como os sistemas federados. Os resultados experimentais mostraram que o FeSHyD pode reduzir o tempo de processamento em até 45%.

1.1 MOTIVAÇÃO

Em um cenário no qual uma consulta envolve dados armazenados tanto em uma base de propriedade do próprio usuário que submete a consulta como em bases de terceiros, existem duas alternativas que podem ser adotadas para o processamento da consulta: *(i)* utilizar um sistema de consultas federado que considera todas as bases como caixas-pretas; *(ii)* ter uma intervenção do usuário para obter resultados dos *endpoints* e integrá-los com o sistema RDF proprietário. A primeira alternativa é preferível, comparada à segunda, porque não passa para o usuário a responsabilidade de fazer as ligações entre as bases proprietárias e de terceiros. No entanto, tratar uma base proprietária como uma caixa-preta pode deixar de explorar possíveis otimizações, uma vez que o proprietário pode ter acesso irrestrito aos seus dados e à estratégia de processamento interno às consultas. Ou seja, podem existir otimizações que só são possíveis se a base proprietária for tratada como uma caixa-branca. O sistema proposto FeSHyD explora esta terceira alternativa. Com isto, uma empresa que possui os dados armazenados em uma base proprietária distribuída, consegue tanto estender sua base com dados pertencentes a bases de terceiros quanto aproveitar possíveis otimizações na base proprietária. Uma otimização é decentralizar o processamento da consulta no moderador e distribuí-la entre os servidores da base proprietária.

1.2 CONTRIBUIÇÕES GERAIS

A contribuição desta dissertação é uma abordagem de processamento de consultas SPARQL distribuídas sobre uma base de dados híbrida, composta por uma base de dados

distribuída proprietária acessada como caixa-branca e por bases remotas acessadas como caixas pretas. Nesta abordagem é proposta uma otimização no processamento interno da base proprietária, com o objetivo de otimizar o desempenho da consulta.

1.3 CONTRIBUIÇÕES ESPECÍFICAS

As contribuições específicas da dissertação são listadas abaixo.

- Uma otimização na base proprietária distribuída que utiliza os servidores que a compõem na geração e submissão de subconsultas para bases externas autônomas. Tal ação é realizada em paralelo pelos servidores.
- Geração de um plano de consulta em que é determinada a sequência de processamento e as informações necessárias para dividir a consulta entre a base proprietária e os *endpoints* SPARQL remotos.
- Implementação do sistema FeSHyD, que recebe a consulta SPARQL e faz seu processamento sobre a base híbrida.
- Uma análise experimental para determinar o desempenho do sistema FeSHyD, em que são analisados o tempo de execução da consulta e o efeito da variação no número de servidores que compõem a base proprietária.

.

1.4 ESTRUTURA DA DISSERTAÇÃO

No Capítulo 2 são apresentados os conceitos básicos. No capítulo 3 são descritos alguns sistemas relacionados. No Capítulo 4 é apresentada a abordagem de processamento proposta. No Capítulo 5 são apresentados detalhes da implementação e os experimentos realizados. No Capítulo 6 é apresentada a conclusão.

2 CONCEITOS BÁSICOS

Este capítulo apresenta a fundamentação teórica necessária para propor o sistema FeSHyD. Na Seção 2.1 é apresentado o modelo de dados RDF e na Seção 2.2 são descritos conceitos relacionados à linguagem de consulta SPARQL. Na seção 2.3 são descritos os critérios para classificar um sistema com relação ao seu tipo de armazenamento de dados e de processamento de consultas. Nas seções 2.4 e 2.5 são descritas as etapas principais do processamento de consultas sobre bases de dados distribuídas e de busca federada sobre *endpoints* SPARQL.

2.1 RDF

O modelo de dados *Resource Description Framework* (RDF) foi proposto para expressar informações sobre diferentes recursos[1]. Um recurso pode representar conceitos concretos e abstratos como pessoas, objetos e ações. Uma relação simples entre dois recursos é representada por meio de uma tripla no formato `(sujeito,predicado,objeto)`. Esta tripla indica que o recurso contido no sujeito possui uma relação (predicado) com o recurso do objeto. A relação é direcionada do sujeito para o objeto (e.g., o sujeito `Aluno` executa a ação `Estudar` no objeto `Universidade`). Um recurso pode ser sujeito em uma tripla e objeto de outra tripla. Esta característica possibilita visualizar o conjunto de todas as triplas como um grafo direcionado.

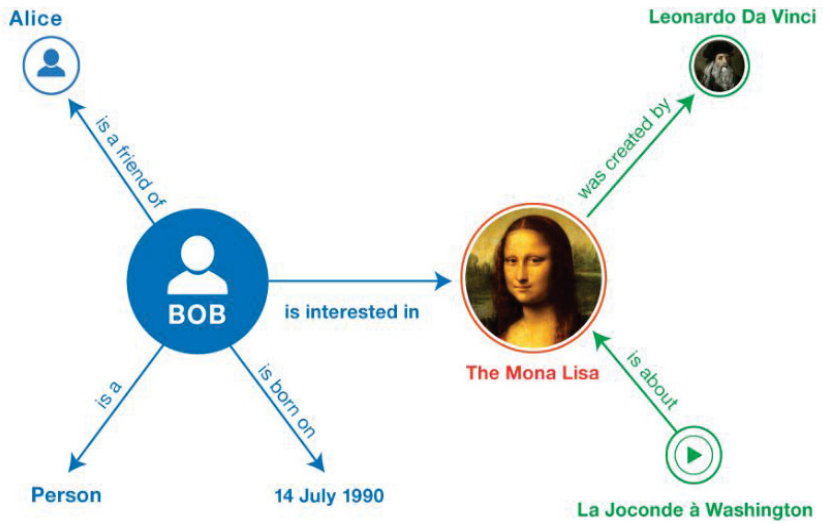


Figura 2.1: Exemplo de um grafo de dados RDF[1]

Na Figura 2.1 o recurso representando uma pessoa (chamada `BOB`) possui quatro relações (predicados) com outros recursos. Em todas as relações `BOB` aparece como sujeito da tripla. Um outro recurso, que descreve a pintura `The Mona Lisa`, aparece como sujeito da tripla contendo o predicado `Was created by` e o objeto `Leonardo Da Vinci`. Porém, o recurso em questão aparece como objeto de duas outras triplas. Na representação em forma de grafo os sujeitos e objetos são associados a nodos e os predicados às arestas direcionadas.

Os recursos podem ser classificados em três tipos: IRI (*International Resource Identifier*), literal ou um nodo branco (*blank node*). Uma IRI identifica um recurso, que contém informações sobre esse recurso. Uma IRI pode apresentar o formato de uma *Uniform Resource Locator* (URL)

[1] `https://www.w3.org/TR/rdf11-primer/`

e ser acessada via internet. Como exemplo, a URL `http://dbpedia.org/resource/Leonardo_da_Vinci` indica o conteúdo do recurso `Leonardo Da Vinci` da Figura 2.1. As IRIs também podem ser representadas por uma abreviação[2] A IRI acima citada pode ser reduzida e descrita como `dbr:Leonardo_da_Vinci`, na qual o prefixo `dbr` é a forma abreviada de parte da URL original. Em uma tripla RDF, as IRIs podem aparecer em qualquer uma das três posições. Um recurso também pode ser do tipo literal, quando é formado por uma sequência de caracteres (*string*). Tais recursos são apenas possíveis como objeto de uma tripla. Alguns literais apresentam uma identificação do tipo de dados como data e valores numéricos e do idioma na qual a *string* foi escrita. Nodos brancos indicam que um elemento da tripla é um recurso, porém sem a necessidade de identificar ou nomear eles. Tais recursos podem ser tanto sujeitos como objetos.

O grafo formado por triplas armazenadas em uma base de dados é chamado de **grafo de dados**. Existe a possibilidade de dois recursos localizados em bases distintas estarem ligados por um predicado, criando uma ponte entre dois grafos de dados (Rakhmawati et al., 2013). Este predicado unindo dois grafos de diferentes servidores recebe o nome de **aresta de ligação**. Uma dificuldade originada desta ligação é a heterogeneidade entre as bases, na qual cada uma apresenta sua própria definição para um mesmo recurso. Um predicado comumente utilizado para ligar dois recursos em diferentes bases é o `owl:sameAs`, indicando que ambos representam o mesmo recurso. Grafos conectados entre diferentes bases formam um único grafo de dados.

É possível classificar e dar significado aos recursos por meio de semântica. A linguagem *RDF Schema* (RDFS) define recursos como propriedades, classes e instâncias. O conjunto das definições forma um vocabulário. Um exemplo é a propriedade `rdfs:type`, indicando que um recurso é instância (ou tipo) de uma classe (outro recurso). A partir do RDFS, é possível produzir novas triplas (ou novo conhecimento) usando regras de inferência sobre o grafo. Uma regra de inferência analisa as classificações criadas sobre os recursos e identifica novas classificações e ligações ocultas. O conjunto formado pelo vocabulário, interconexões semânticas e regras de inferências é chamado de ontologia (Hendler, 2001). Por meio da ontologia é possível visualizar como os dados estão organizados na base. Um **esquema local** descreve como as relações entre os tipos de recursos são feitas em uma base de dados. Um **esquema global** proporciona uma visão integrada dos esquemas estruturais locais. Ambos os esquemas são considerados como **grafos de estruturas** da base de dados.

Ontologias possibilitam a atribuição de significado semântico aos recursos. Este processo de aplicar semântica a recursos é parte intrínseca da **Web Semântica**[3]. A Web tradicional é formada por documentos sem identificação ou significado. A ausência da semântica sobre os documentos dificulta o processo da leitura e reconhecimento automático dos recursos por máquinas. Um dos objetivos da Web Semântica é produzir documentos legíveis ás máquinas, visando a automação na análise dos recursos (Hendler, 2001). Outro objetivo da Web Semântica é conseguir integrar dados dispersos em vários locais. Esse integração leva em consideração o conceito de **dados ligados** (*Linked Data*). Ligações entre recursos de diferentes locais são criadas com a finalidade de facilitar a busca por informações relevantes[4]. Para tal, os dados devem ser identificados por IRIs acessíveis remotamente, estarem modelados em RDF e possuir links para recursos de outras bases.

[2] `https://www.w3.org/TR/rdf11-concepts/`
[3] `https://www.w3.org/standards/semanticweb/`
[4] `https://www.w3.org/DesignIssues/LinkedData.html`

2.2 SPARQL

SPARQL[5] (*SPARQL Protocol and RDF Query Language*) é a linguagem padrão recomendada pelo consórcio W3C para consultas em bases RDF. Uma busca SPARQL é formada por **padrões de triplas**, que reunidos formam um **grafo padrão básico** (*Basic Graph Pattern*). Os padrões de triplas são semelhantes às triplas de dados, porém nas três posições (sujeito, predicado e objeto) pode haver variáveis. O grafo padrão também é identificado como **grafo da consulta**.

Devido a possibilidade de um mesmo recurso ser sujeito ou objeto de uma tripla, um grafo padrão possui diferentes formatos. A Figura 2.2 ilustra os tipos de grafos que uma consulta SPARQL pode assumir. Na Figura 2.2 (a) é apresentado o padrão estrela, que consiste em um único sujeito com um ou mais objetos. No padrão de cadeia apresentado no item (b) o objeto de uma tripla se torna o sujeito da próxima. No item (c) o padrão de árvore pode conter vários padrões estrela, no qual os objetos de uma tripla se tornam os sujeitos da próxima estrela. O padrão ciclo no item (d) é uma variação do padrão cadeia, no qual o objeto da última tripla é relacionado com o sujeito da primeira tripla. O item (e) é chamado de padrão complexo devido a possibilidade de ocorrência dos outros padrões em um mesmo grafo da consulta.

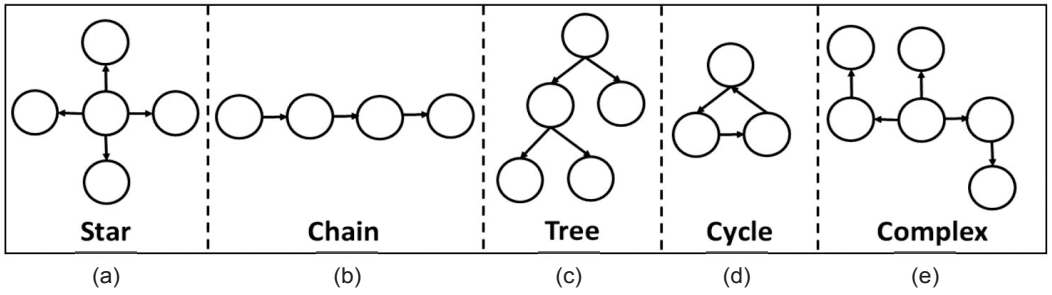


Figura 2.2: Formatos do grafo de consulta. Estrela(a). Cadeia(b). Árvore(c). Ciclo(d). Complexo(e) (Wylot et al., 2018)

A execução de uma consulta SPARQL é baseada no casamento do grafo da consulta com a base de dados RDF. As variáveis do grafo de consulta são substituídas por valores do grafo de dados durante a sua execução da consulta. A substituição de variáveis da consulta por valores da base de dados recebe o nome de **mapeamento** (da Cunha e Lóscio, 2014). Caso o grafo de consulta gerado a partir destas substituições for igual a um subgrafo da base de dado então um resultado foi encontrado (casamento de padrões). Logo, uma variável não mapeada (*unbound*) passa a estar mapeada (*bound*) quando um dado da base é associado a essa variável.

Na Figura 2.3 é apresentado um exemplo de consulta SPARQL sobre uma base de dados RDF. Na Figura 2.3(a) uma base de dados que armazena triplas RDF é exemplificada por meio de um grafo direcionado. Nesta base é possível identificar três padrões estrelas (cada uma relacionada a uma pessoa). As IRIs estão no formato abreviado e os literais estão no formato *string* (entre aspas duplas) ou numérico. Os retângulos representam os recursos indicados por IRIs ou nodo branco. Tais recursos se relacionam com outros retângulos ou com literais. A propriedade que liga dois recursos também é identificada por uma IRI. No exemplo, o recurso representado pela IRI `ex:Bob` possui a propriedade `rdf:type` com outra IRI `foaf:Person`. A propriedade em questão indica que o sujeito é uma instância da classe identificada no objeto.

A Figura 2.3(b) ilustra uma consulta SPARQL conjuntiva, ou seja, consultas que adotam a cláusula SELECT, a junção entre padrões de triplas (.), e a cláusula FILTER. Nesta dissertação, há a restrição que variáveis são usadas somente em sujeitos e objetos de padrões de triplas da

[5] `https://www.w3.org/TR/rdf-sparql-query/`

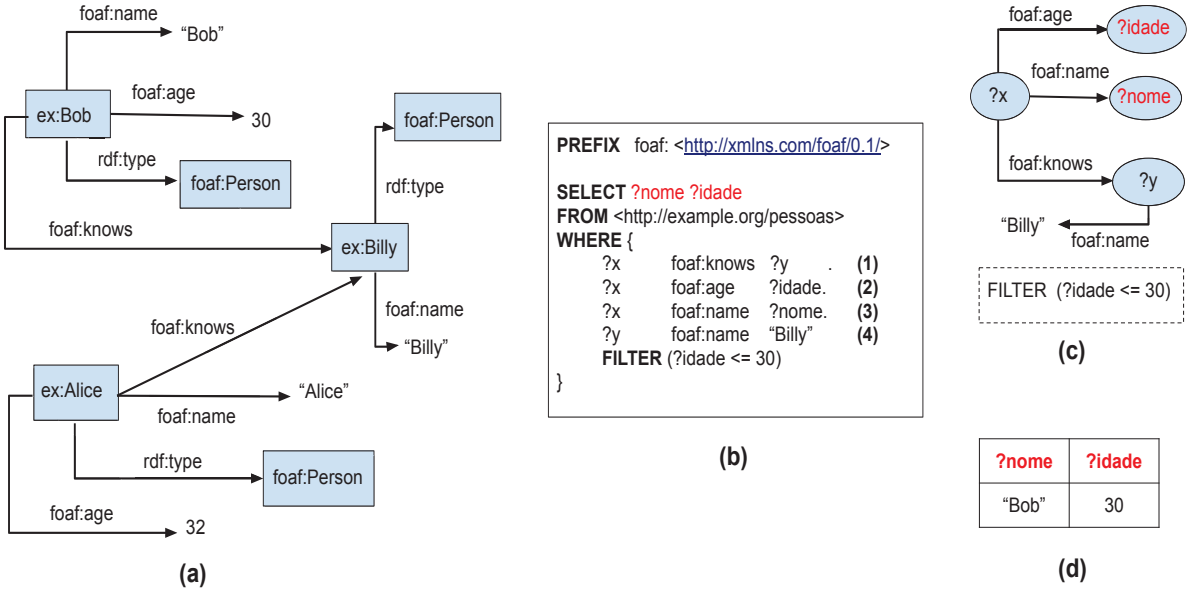


Figura 2.3: Grafo de dados(a); Consulta SPARQL(b); Grafo da consulta(c); Mapeamentos encontrados(d)

consulta. Na primeira parte da consulta são declaradas as abreviações com o auxílio da cláusula PREFIX, exemplificada na consulta da Figura 2.3(b). Após a declaração das abreviações, a cláusula SELECT descreve uma lista de **variáveis de projeção**. Tais variáveis são as variáveis retornadas pela consulta. Variáveis em SPARQL começam com os caracteres `?` ou `$`, seguido do identificador da variável. No exemplo, as variáveis de projeção são `?nome` e `?idade`. A consulta segue com a cláusula FROM, que tem como finalidade especificar qual grafo de dados da base será utilizado como grafo de dados padrão para o processamento da consulta. No exemplo o grafo de dados da Figura 2.3(a) é indicado como o grafo padrão através da IRI `http://example.org/pessoas` declarada na cláusula FROM. Na próxima parte da consulta é descrito o grafo da consulta e as triplas que o compõem, utilizando a cláusula WHERE. Os padrões de triplas são agrupados entre chaves. Cada padrão de tripla é terminado em ponto, o qual significa uma junção entre padrões de triplas (Penteado, 2017). Para a junção ser executada é necessário que duas triplas apresentem uma variável em comum. Esta variável é chamada de **variável de junção**. A agregação de todas as triplas resulta no grafo da consulta. No exemplo, o grafo da consulta contém quatro padrões de tripla. As três primeiras triplas apresentam a variável de junção `?x` como sujeito da tripla.

O conjunto de resultados de uma consulta SPARQL pode sofrer restrições declaradas por meio da cláusula FILTER. Dentro do filtro pode haver uma ou mais restrições combinadas por operadores lógicos. Um filtro pode realizar comparações entre *strings* (com auxílio da função `regex`) e comparações algébricas. No exemplo é possível observar uma cláusula FILTER após a descrição do grafo da consulta, restringindo a idade para ser menor ou igual a 30.

Na Figura 2.3(c) é possível visualizar o grafo da consulta referente a Figura 2.3(b). O grafo padrão produzido a partir da variável de junção `?x` está no formato estrela. A última tripla do grafo da consulta apresenta a variável de junção `?y` posicionada como sujeito da tripla. Esta tripla realiza a junção com a variável comum (`?y`) localizada no primeiro padrão de tripla. Diferente do padrão estrela encontrado no grafo da consulta, a junção da variável `?y` ocorre entre sujeito de um padrão de tripla com o objeto contido em outra tripla. Por fim, o conjunto dos resultados finais encontrados na execução da consulta da Figura 2.3(b) na base da Figura 2.3(a) é demonstrado na Figura 2.3(d).

Existem outras cláusulas que podem modificar como o grafo da consulta deve ser avaliado sobre uma base de dados. Quando é realizado um casamento de padrões entre o grafo

da consulta e o grafo de dados, o grafo inteiro da consulta é encontrado na base ou nenhuma resposta é obtida (linhas em branco na tabela resposta). Através da cláusula OPTIONAL o grafo de consulta declara que parte dos padrões de triplas é opcional no conjunto de resultados. A ideia é similar à cláusula relacional de junção externa. Outro cláusula denominada UNION descreve dois padrões de triplas, nos quais ambos são analisados sobre a base de dados. Esta cláusula possibilita que um ou ambos os padrões de triplas casem com o grafo de dados. Logo, a cláusula UNION cria padrões de triplas alternativos. A cláusula SERVICE[6] da versão SPARQL 1.1 pode ser utilizado para requisitar parte da consulta SPARQL sobre bases de dados externas. Para essa cláusula a base de dados externa tem que ser especificada de alguma forma antes da execução da consulta, pois ela é adicionada na consulta SPARQL, dentro da cláusula WHERE. Um exemplo é apresentado na Figura 2.4. Nessa consulta as triplas 1, 2 e 3 são consultadas na base local, designada pela cláusula FROM, e a tripla 4 é buscada na base externa representada pela URL `<http://people.example.org/sparql>`.

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?nome ?idade
FROM <http://example.org/pessoas.rdf>
WHERE {
      ?x      foaf:knows  ?y      .   (1)
      ?x      foaf:age    ?idade. (2)
      ?x      foaf:name   ?nome.  (3)
      SERVICE <http://people.example.org/sparql> {
      ?y      foaf:name   "Billy". } (4)
}
```

Figura 2.4: Exemplo de consulta SPARQL utilizando a cláusula SERVICE

Uma consulta pode conter a cláusula ASK ao invés da cláusula SELECT. Este tipo de consulta retorna um valor booleano ao invés de uma tabela. O valor booleano indica se o grafo da consulta analisado produz algum resultado quando casado com o grafo de dados da base. No endereço `https://www.w3.org/TR/rdf-sparql-query/` são descritas várias outras cláusulas.

2.3 SISTEMAS DE CONSULTAS SOBRE DADOS RDF

Dada uma consulta SPARQL, um sistema de consulta deve retornar seu resultado explorando uma base de dados RDF, conforme mostra a Figura 2.5. Um ***endpoint*** SPARQL[7] é um recurso na Web, identificado por uma URL, capaz de receber requisições, na forma de uma consulta SPARQL, e retorna seu resultado usando uma base de dados local (Rakhmawati et al., 2013). O moderador do sistema recebe a requisição da consulta, elabora o plano de consulta e requisita que *endpoints* participem da execução do plano a fim de gerar o resultado da consulta. Em um sistema distribuído, os dados explorados na execução da consulta podem estar alocados na mesma máquina que o moderador ou em máquinas remotas. Os tipos de armazenamento e de processamento podem ser usados para a classificação de sistemas de consulta.

2.3.1 Armazenamento dos dados

Bases de dados podem pertencer a diferentes organizações. Cada organização estabelece os níveis de acesso aos dados para os diferentes tipos de usuários. Estas bases são classificadas como **bases de dados de terceiros**. A maioria dos sistemas descritos no Capítulo 3 requisita a

[6] `https://www.w3.org/TR/sparql11-federated-query/`
[7] `https://www.w3.org/TR/sparql11-federated-query/`

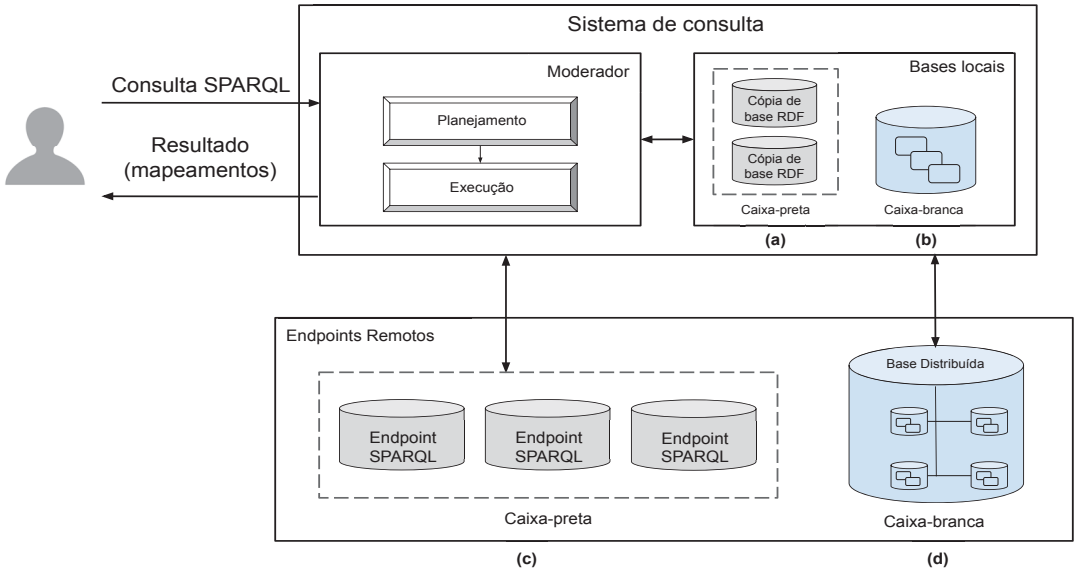


Figura 2.5: Imagem ilustrando base de dados locais caixa-preta(a) e caixa-branca(b) e base de dados remotas caixa-preta(c) e caixa-branca(d)

consulta a este tipo de base. Quando a base de dados e o sistema de consulta pertencem à mesma organização, a base é classificada como **proprietária**.

Com relação a forma de armazenamento e organização dos dados, uma base pode ser classificada como centralizada ou distribuída. Tanto bases proprietárias como pertencentes a terceiros podem apresentar ambos os tipos de armazenamento. Uma base **centralizada** armazena os dados em um ou mais repositórios contidos em apenas uma máquina. O processamento da consulta também é realizado sobre esta máquina. Logo, o processamento não ocasiona uma latência na comunicação entre a base e o sistema da consulta (Wylot et al., 2018). Em contrapartida, a quantidade de processamento e memória disponíveis está limitado apenas a esta máquina. É possível considerar que bases centralizadas contenham tanto dados proprietários quanto de terceiros. Uma base classificada como centralizada e com dados provindos de terceiros armazena uma cópia da base externa em repositórios locais. Este tipo de armazenamento centralizado pode conter cópia dos dados de diversas bases externas. O acesso aos dados destas cópias é através de consultas SPARQL. Um exemplo de base centralizadas formada por bases de terceiros e uma base de proprietárias é apresentado nas Figura 2.5(a) e Figura 2.5(b), respectivamente.

O armazenamento de dados RDF em bases **distribuídas** é composto por uma rede interna de máquinas conectadas (*cluster*). A rede interna é formada por máquinas proprietárias, ou seja, a rede não contém máquinas pertencentes a outras organizações. Este modelo de armazenamento por meio de uma rede local é apresentado na Figura 2.5(d). Cada máquina contém um conjunto de dados RDF armazenado de maneira independente (Wylot et al., 2018; Özsu, 2016). O processamento da consulta considera que os dados podem estar alocados em uma ou diversas máquinas, ou não existir na base distribuída. Uma abordagem para distribuir os dados RDF entre as bases de dados é particionar o grafo de dados, no qual cada base contém uma parte dos dados.

Também é considerada uma base distribuída uma rede global formada por bases externas pertencentes a terceiros, ligados remotamente (e.g., ligação entre bases por meio de predicados). Esta base distribuída em uma rede global é apresentada na Figura 2.5(c) O conjunto de todos os grafos de estrutura das diversas bases forma uma visão integrada, descrevendo a base distribuída. Os proprietários de outras bases de dados utilizam seus próprios métodos para criar os grafos

de dados, divulgar e fazer ligações com outras bases externas da rede. Em (Rakhmawati et al., 2013) é exemplificada uma busca distribuída envolvendo duas bases localizadas em diferentes servidores. Este exemplo de consulta aborda a área farmacêutica, no qual foram consultadas as bases distintas `Kegg` e `Drugbank` buscando informações sobre remédios e componentes. O modelo distribuído possui maior poder de processamento e armazenamento, se comparado com o modelo centralizado. Isso se deve à quantidade de máquinas disponíveis para a consulta. O conjunto de bases distribuídas possui mais memória disponível se comparado a uma única máquina. Também é possível explorar o paralelismo na execução da consulta entre os servidores da rede. A desvantagem está no custo e tempo para realizar a comunicação dos dados entre as bases.

2.3.2 Modelo de processamento

Um dos critérios para classificar o processamento de uma consulta SPARQL é a localidade do moderador com relação a base de dados. Esta localidade pode alterar o custo de comunicação para a recuperação dos dados (Wylot et al., 2018). Um sistema apresenta processamento **local** quando os dados e o moderador estão localizados no mesmo servidor. As Figura 2.5(a) e Figura 2.5(b) apresentam bases locais contidas no mesmo servidor que o moderador. Em contrapartida, quando os dados estão armazenados separadamente do moderador o processamento é classificado como **remoto**. As bases remotas são apresentadas nas Figura 2.5(c) e Figura 2.5(d).

O modelo de processamento da consulta também varia de acordo com as limitações, tanto no acesso aos dados contidos na base quanto na escolha do método de execução. Quanto ao nível de acesso, o sistema de consulta pode estar restrito a submeter consultas à base. Este tipo de acesso é nomeado de **caixa-preta**. Este nome foi atribuído devido a seu comportamento limitado, no qual a estratégia de execução de consulta é desconhecida ou inacessível, e informações como o tipo de armazenamento podem não estar disponíveis. As bases de dados acessadas como caixas-pretas apenas recebem a consulta SPARQL e retornam o resultado. Alguns sistemas disponibilizam informações estatísticas sobre os dados (metadados) (Görlitz e Staab, 2011) e outros descrevem como os dados estão organizados por meio de ontologias (da Cunha e Lóscio, 2014). Mesmo com tais informações sobre o conteúdo, a análise interna da execução da consulta não é visível. Logo, implementar uma otimização no processamento da consulta sobre uma base com este tipo de acesso não é possível.

Um modelo de acesso menos restritivo sobre uma base de dados permite que um sistema de busca possa acessar os dados e modificar como o processamento da consulta sobre esta base é realizado. A base de dados com acesso menos restrito recebe o nome de **caixa-branca**. Esta base possui algumas das características de uma base acessada como caixa-preta, que recebe uma requisição de consulta e retorna o resultado. Também é possível existir metadados e o esquema estrutural da base. Porém uma base de dados acessada como caixa-branca permitem algumas otimizações. Um possível aprimoramento é a criação de índices.

Existem diversos métodos de processamento de uma consulta SPARQL sobre uma base distribuída composta por *endpoints* acessados como caixas-pretas. **Links transversais** é um tipo de processamento distribuído que descobre as bases de dados de terceiros em tempo de execução. Logo, as bases não são conhecidas pelo sistema antes do seu processamento. A descoberta de novas bases acontece percorrendo IRIs encontradas durante a execução da consulta (Rakhmawati et al., 2013; Ladwig e Tran, 2011). Como descrito na subseção 2.1, uma URL é uma IRI. Logo, o percurso de IRI é considerado uma exploração de URLs. O sistema toma como ponto de partida um padrão de tripla do grafo da consulta e uma IRI inicial. Durante o percurso são avaliadas

quais as bases que respondem os padrões de triplas da busca. As IRIs encontradas são utilizadas para buscar novas IRIs.

Outro modelo de processamento sobre bases de dados RDF distribuídas, acessadas como caixas-pretas, é conhecido como **busca federada**. O conjunto destas bases distribuídas pertencentes a terceiros é denominado federação. Em (Rakhmawati et al., 2013; Saleem et al., 2016; Oguz et al., 2015) são comparados alguns sistemas de busca federada sobre bases RDF. O moderador pode prover uma visão integrada das bases RDF distribuídas para o usuário (da Cunha e Lóscio, 2014). Deste modo, o esquema global possibilita ao usuário criar uma consulta sem associar o dado a sua localidade. O moderador analisa uma consulta SPARQL e a decompõe em subconsultas baseando o particionamento nos endpoints relevantes para a resolução da consulta. Após a decomposição, as subconsultas são submetidas aos `endpoints` remotos. A requisição de uma subconsulta resulta em um conjunto de mapeamentos os quais são usados para determinar os valores das variáveis de outra subconsulta. Os resultados obtidos pelas requisições das subconsultas são reunidos em um conjunto de resultados. Logo, estes resultados produzidos nas subconsultas são considerados resultados intermediários. Ao final, este conjunto é retornado ao usuário do sistema. Ao contrário do método de links transversais, que encontra as bases em tempo de execução, o conhecimento prévio dos `endpoints` contidos na federação é necessário para o processamento da consulta.

A Figura 2.6 resume os conceitos apresentados nesta seção. Na Figura 2.6(a) são ilustrados os tipos de armazenamento e na Figura 2.6(b) os tipos de processamento da consulta. A partir do tipo de armazenamento é possível deduzir regras de inferências sobre o tipo de processamento. Estas inferências são apresentadas na Figura 2.6(c).

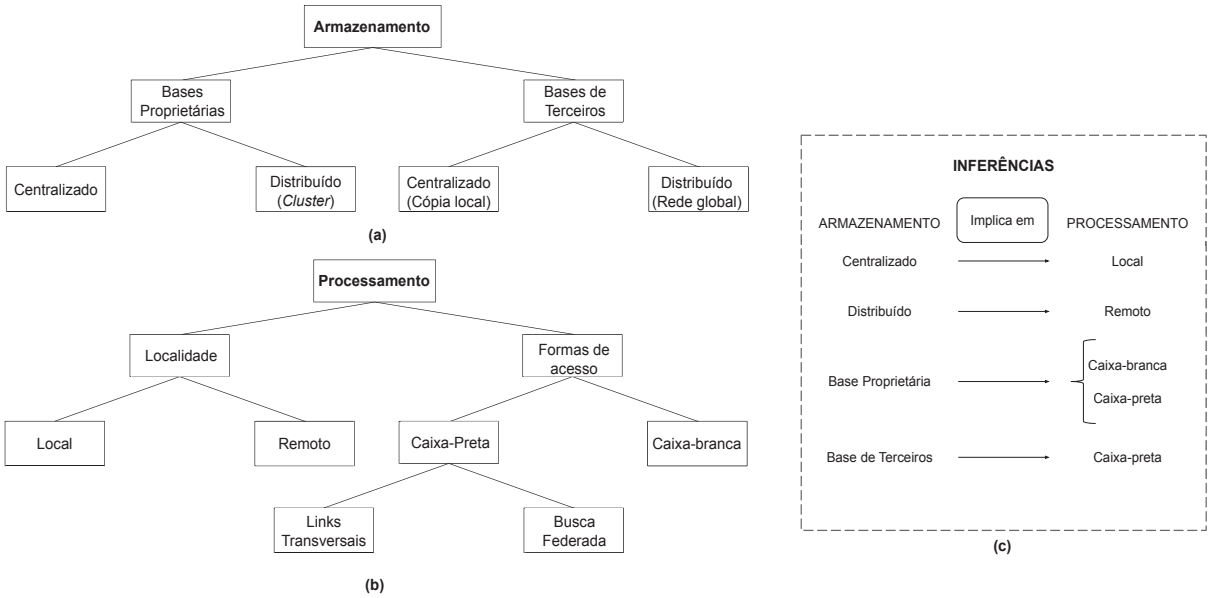


Figura 2.6: Tipo de armazenamento(a) e Processamento da consulta(b). Regras de inferência(c)

O modelo de busca federada sobre bases de dados RDF por meio de SPARQL é a base do sistema proposto nesta dissertação. Assim, na Seção 2.4 é descrito com mais detalhes o processo de uma busca distribuída comum entre os diferentes modelos de processamento e na Seção 2.5 são descritas as etapas específicas de uma busca federada.

2.4 ETAPAS DA BUSCA DISTRIBUÍDA SOBRE DADOS RDF

Nesta seção são descritas as etapas elementares das abordagens de processamento de dados RDF distribuídos, conforme (Wylot et al., 2018; Özsu, 2016). De uma forma geral, o

processamento de uma busca distribuída é dividido na fase de planejamento e fase de execução (Rakhmawati et al., 2013). A **fase de planejamento** compreende as seguintes etapas: análise da consulta SPARQL, decomposição da consulta original em subconsultas e ordenação dos padrões de triplas. Ao fim da fase de planejamento são criados diversos planos de consulta. A **fase de execução** da consulta processa os possíveis planos de consultas. Nesta fase é realizada a etapa de escolha do método de junção dos resultados intermediários utilizado na execução do plano de consulta. Com a finalização da execução do plano de consulta, o resultado final é reunido e enviado para o usuário. No resto desta seção são descritos os módulos que compõem um sistema de consulta distribuído.

2.4.1 Análise da Consulta

Como descrito na Seção 2.1 uma consulta SPARQL pode ser vista como um grafo direcionado chamado de grafo da consulta. Entretanto, existem diferentes tipo de representações internas. O primeiro módulo elementar de uma busca distribuída é responsável por receber a consulta SPARQL do usuário e criar uma **representação interna da consulta** (Rakhmawati et al., 2013). Este módulo recebe o nome de analisador da consulta. A partir das representações internas da consulta SPARQL do usuário é possível decompor a consulta e implantar otimizações.

No processo de análise da consulta SPARQL, ela é disponibilizada pelo usuário no formato de uma *string* de caracteres[8]. O sistema analisa a *string* SPARQL de entrada e gera uma **árvore sintática abstrata** utilizando os elementos da consulta SPARQL (e.g., termos RDF, padrões de triplas). Neste processo as IRIs abreviadas são estendidas para criar as IRIs completas (tornando uma IRI acessível como uma URI). A partir da árvore sintática o analisador produz um formato de consulta SPARQL abstrata utilizando representações internas da linguagem para representar os elementos. Esta nova consulta é composta pelas bases de dados descritas na árvore, o tipo de consulta SPARQL (e.g., SELECT, ASK) e uma expressão algébrica SPARQL derivada do grafo da consulta. Esta expressão apresenta uma descrição textual da consulta original, utilizando diferentes terminologias para os elementos da consulta. Para cada símbolo na consulta SPARQL abstrata é definido adequadamente um operador para a avaliação sobre a base de dados. Durante todo este processo de análise da consulta é possível modificar a execução da consulta com a finalidade e efetuar otimizações (Nikolov et al., 2017).

A partir da árvore abstrata podem ser derivados outros tipos de representação interna da consulta. Os elementos desta árvore possibilitam a criação de um grafo direcionado para representar internamente a consulta.

2.4.2 Criação do plano de consulta

Nesta etapa se inicia a criação do plano de consulta. A elaboração deste plano tem início na decomposição do grafo de consulta original em subconsultas. Cada subconsulta produzida é composta por um ou mais padrões de triplas. Dentre as diferentes formas de agrupamento de triplas, o método de grupos exclusivos é proposto para o processamento de uma busca federada (Rakhmawati et al., 2013). A ideia principal é agrupar as triplas que possuem apenas uma base de dados relevante `S`, e essa base é igual para todo o grupo (e.g., servidores em um *cluster* ou *endpoints SPARQL*). Portanto, ao invés de requisitar uma única tripla para uma base de dados `S`, é enviado o grupo exclusivo de triplas. O objetivo é diminuir as requisições aos *endpoints*.

Após a criação das subconsultas, elas podem ser reordenadas para criar diferentes planos de consulta (Rakhmawati et al., 2013). Um plano de consulta otimizado tende a gerar menos

[8] `https://www.w3.org/TR/rdf-sparql-query/#sparqlDefinition`

resultados intermediários, melhorando assim o desempenho do processamento da consulta. Esta ordem pode ser baseada em funções que determinam o custo de execução de cada plano de consulta. Tais funções podem ser baseadas em heurísticas ou metadados da base. Além de ordenar as subconsultas, outra modo de reduzir a quantidade de resultados é ordenar os padrões de triplas contidos nas subconsultas. Um modo de ordenar os padrões de triplas é utilizar seletividade das triplas. Padrões de triplas seletivos geram menor quantidade de resultados intermediários, que são utilizados no mapeamento de outros padrões de triplas (Stocker et al., 2008). Em outras palavras, padrões de triplas mais seletivos são mais restritivos na busca por resultados na base. Uma possível otimização é posicionar triplas mais seletivas no início do plano de consulta. O motivo é que os resultados intermediários são usados para mapear as variáveis da próxima subconsulta, descrita no plano de consulta. Logo, um padrão de tripla que produz menos mapeamentos gera menos subconsultas e, consequentemente, menos requisições à base, diminuindo assim o tempo de execução.

Um modo de estimar a seletividade dos padrões de tripla é aplicar a heurística de contagem de variáveis (Stocker et al., 2008). Este método não requer informações pré-processadas para determinar a quantidade de resultados intermediários. Nela, a seletividade de um padrão de tripla leva em consideração a quantidade das variáveis. Um padrão de tripla é mais seletivo se tiver menos variáveis. Uma variável pode retornar um ou mais resultados. Logo, a quantidade de resultados intermediários encontrados é menor se houverem menos variáveis no padrão de tripla. Em contrapartida, alguns dos sistemas de consulta utilizam informações estatísticas (e.g., catálogo de dados) para determinar uma ordem otimizada para os padrões de tripla. Um padrão de tripla é mais seletivo se tiver menos variáveis.

2.4.3 Execução da consulta

Com a finalização da fase de planejamento um plano de consulta é elaborado. A primeira etapa da fase de execução é determinar a estratégia de junção dos padrões de triplas descritos no plano. A escolha do algoritmo de junção impacta no desempenho do sistema de busca. Esse algoritmo descreve como os resultados intermediários de uma subconsulta são utilizados para mapear as variáveis da próxima subconsulta. Alguns aspectos como quantidade de resultados intermediários e custo de comunicação determinam a escolha do algoritmo de junção. Os processadores de consulta podem utilizar mais de uma estratégia de junção.

Um exemplo de algoritmo de junção é o *Nested Loop Join* (NLJ). Neste algoritmo os resultados intermediários produzidos por uma subconsulta são usados para determinar os valores das variáveis na próxima subconsulta através dos mapeamentos encontrados pela subconsulta anterior(Schwarte et al., 2011). Uma otimização do NJL é o *Bind Join*. Esta estratégia transmite os resultados intermediários como filtros (No SPARQL através da cláusula FILTER) para a próxima subconsulta (Rakhmawati et al., 2013). O custo de comunicação com a base neste algoritmo é menor se comparado com NLJ porém é necessário o término da subconsulta anterior. Uma outra estratégia é o *Hash Join*. Esta estratégia envia os dois argumentos da junção em paralelo para os *endpoints* (Görlitz e Staab, 2011). Os resultados são reunidos e armazenados localmente em uma ou mais tabelas hash.

2.5 BUSCA FEDERADA

Em uma busca federada os dados de interesse estão armazenados em *endpoints* SPARQL pertencentes a terceiros. É necessário um processo de seleção de fontes durante a fase de planejamento. Outros aspectos são relevantes na geração do plano de consulta, tais como o modelo de dados utilizado para armazenar os dados e a interface de comunicação com o *endpoint*.

2.5.1 Seleção de fontes

A seleção de fontes em uma busca federada determina quais as bases de dados, contidas na federação, são relevantes na resolução de uma consulta SPARQL (Rakhmawati et al., 2013). Escolher `endpoints` que não agregam no conjunto de resultados finais pode ocasionar no aumento do custo de comunicação e no tempo de execução da consulta. O mediador do sistema federado é responsável pela correta execução desta etapa. Nesta subseção são descritos alguns métodos de seleção de fontes para consultas distribuídas SPARQL.

- **Consultas SPARQL ASK:** esta abordagem utiliza consultas do tipo ASK da linguagem SPARQL (Schwarte et al., 2011). A consulta ASK resulta em um valor booleano dizendo se a base de dados em questão responde ao padrão de tripla consultado. Este método não exige metadados sobre as bases para a seleção. Com isso a integração de novas bases de dados na federação se torna mais simples. Uma desvantagem é o aumento de acessos às bases de dados na etapa de planejamento da consulta, devido a necessidade de utilizar consultas ASK SPARQL para verificar quais partes da consulta podem ser respondidas por cada base (Saleem et al., 2016).

- **Catálogo de dados VoID:** o descritor VoID (*Vocabulary of Interlinked Dataset*)[9] é disponibilizado pelo *endpoint* da base dados e contém informações estatísticas e metadados sobre a base. Neste método de seleção os metadados são a base da criação e requisição da consulta federada aos *endpoints* (Görlitz e Staab, 2011). Exemplos de informações neste catálogo de dados incluem a quantidade total de triplas, os predicados e instâncias de classes e suas respectivas cardinalidades. Estas informações podem ser usadas para estimar ou determinar a quantidade de resultados intermediários dos padrões de tripla da consulta. O vocabulário, exemplos de recursos e informações sobre os métodos de acesso à base (e.g., *endpoint*) também estão contidos neste catálogo. Na Figura 2.7 é apresentado um exemplo de descritor VoID resumido descrevendo a base `ChEBI`. Este exemplo apresenta informações como a quantidade de triplas e propriedades contidas na base.

- **Indexação de dados:** a indexação de dados auxilia na otimização da busca por padrões de tripla (Ladwig e Tran, 2011). Neste processo informações como triplas de dados RDF são indexadas e armazenadas localmente, criando um conjunto de metadados sobre as bases. O processo de indexação pode ser efetuado antes ou durante a execução da consulta (Rakhmawati et al., 2013). A desvantagem deste método é o espaço de armazenamento necessário para indexar as triplas de dados.

- **Esquemas estruturais da base:** este tipo de seleção aborda o conceito de casamento de padrões entre estruturas para identificar os *endpoints* relevantes para uma consulta. O método aborda a comparação do grafo da consulta, dos esquemas locais e do esquema global da federação. Neste processo são identificados os elementos equivalentes entre os diferentes grafos de estrutura. Uma possível vantagem é o armazenamento dos esquemas estruturais junto ao moderador da consulta, diminuindo o custo de comunicação. Entretanto uma desvantagem pode ser, em caso de modificação dos dados na base, a necessidade da requisição dos esquemas estruturais atualizados.

- **Variáveis globais:** Este modelo de seleção das fontes é baseado na localização dos resultados, onde são buscadas as variáveis de junção entre dois *endpoints* (Abdelaziz

[9] `https://www.w3.org/TR/void/`

et al., 2017a). Variáveis de junção entre diferentes bases são nomeadas de variáveis globais. Tais variáveis não podem ser respondidas sobre um único *endpoint*, pois necessitam de duas bases de dados distintas para produzir um resultado completo. Logo, uma subconsulta requisitada a um *endpoint* não contém variáveis globais. A partir deste conceito, é possível localizar a base de dados relevante para um conjunto de triplas. Uma variável global pode aparecer como objeto de uma tripla e sujeito de outra ou como objeto (ou sujeito) em ambos os padrões de triplas. Neste método de seleção não são considerados metadados e informações estatísticas para encontrar as bases relevantes. Uma possível desvantagem é o aumento de requisições aos *endpoints*.

```
:ChEBI a void:Dataset ;

# general information
dcterms:title "ChEBI" ;
dcterms:description "Chemical Entities" ;
foaf:homepage
    <http://chebi.bio2rdf.org/>
void:sparqlEndpoint
    <http://chebi.bio2rdf.org/sparql> ;

# simple data statistics:
void:triples          "7325744" ;
void:entities          "50477" ;
void:properties          "28" ;
void:distinctSubjects "50477" ;
void:distinctObjects "772138" ;
```

```
# entity count per concept
void:classPartition [
    void:class chebi:Compound ;
    void:entities "50477" .
] ;

# triple count per predicate
void:propertyPartition [
    void:property bio:formula ;
    void:triples "39555" ;
] , [
    void:property bio:image ;
    void:triples "34055" ;
] , [
    ...
] .
```

Figura 2.7: Exemplo de um catálogo de dados VoID descrevendo a base de dados ChEBI (Görlitz e Staab, 2011)

2.5.1.1 Interface da consulta

Alguns sistemas distribuídos disponibilizam diferentes formas de acessos às bases. A maioria considera a linguagem de consulta SPARQL sobre dados RDF. Porém é possível usar outros tipos de consulta como a busca por palavras-chaves e serviços Web (Nikolov et al., 2017). Algumas bases de dados apresentam busca por palavras-chaves embutidas em consultas SPARQL. Os serviços Web são incorporados à linguagem SPARQL com o auxílio dos descritores de serviço. Neste processo o moderador verifica as informações dispostas no descritor com a finalidade de requisitar a busca e incorporar os dados do serviço no processamento da consulta SPARQL.

2.5.1.2 Armazenamento interno

Embora o acesso a bases de dados RDF seja realizado através da linguagem de consulta SPARQL, alguns sistemas consideram que outros modelos de dados podem ser acessados com a mesma linguagem. É possível acessar dados em uma base relacional, na qual os dados são traduzidos como RDF e acessados com SPARQL (Nikolov et al., 2017). Estes sistemas com modelos de dados heterogêneos adaptam os planos de consulta com o objetivo de integrar corretamente todas as bases de dados. No plano de consulta é especificada a metodologia para adequar esta diversidade de modelo de dados do sistema distribuído.

2.6 DISCUSSÃO

Neste capítulo foram apresentados conceitos e classificações relacionados ao armazenamento de bases RDF e processamento de consultas SPARQL. Estes conceitos serão utilizados no próximo capítulo para apresentar e comparar trabalhos correlatos.

3 TRABALHOS RELACIONADOS

Atualmente há diversas bases de dados modeladas a partir do conceito de Web semântica e dados ligados. Diversos sistemas foram desenvolvidos com a finalidade de realizar buscas sobre estas bases de dados. Tais sistemas podem apresentar diferentes métodos de armazenamento dos dados, de estratégias para a criação de plano de consulta e de processamento de consultas sobre as bases. Este capítulo apresenta uma análise sobre trabalhos desenvolvidos para a execução de tais consultas. Por fim, os trabalhos são comparados com o sistema proposto nesta dissertação. Foram descritos seis trabalhos relacionados que se assemelham ao sistema apresentado nesta dissertação.

3.1 PROPRIEDADES DE AVALIAÇÃO

Diversas propriedades podem ser usadas na categorização de sistemas de busca distribuída. Algumas das propriedades de avaliação usadas aqui são as mesmas utilizadas em (Rakhmawati et al., 2013), que faz uma análise comparativa entre diversos sistemas de consulta federada. Porém, neste capítulo a análise é estendida com o modelo de armazenamento de dados e o processamento da consulta apresentados pelo sistema, além do modelo de dados e a interface de comunicação utilizados. A Tabela 3.1 mostra as seis propriedades usadas neste trabalho para a categorização de sistemas de busca distribuída.

3.2 SISTEMAS DE CONSULTA SOBRE BASES DE DADOS

Os trabalhos relacionados estão organizados a seguir em duas seções. Na seção 3.2.1 estão contidos os sistemas que consideram apenas bases de dados distribuídas. Os sistemas que apresentam dados armazenados tanto de maneira distribuída quanto centralizada são descritos na seção 3.2.2. Por fim, na seção 3.2.3 os sistemas são comparados com relação às características da execução de uma consulta SPARQL e modelos de armazenamento e processamento.

3.2.1 Bases distribuídas

Nesta seção são descritos trabalhos que processam consultas sobre bases distribuídas pertencentes a terceiros. Logo, tais sistemas apresentam localidade remota. As bases são vistas como caixas-pretas e são acessadas através de busca federada. São descritos quatro trabalhos. O sistema SPLENDID, descrito na seção 3.2.1.1, utiliza o catálogo de dados VoID e as

	Propriedades	**Descrição**
1	Armazenamento de dados	Modelo de armazenamento físico de dados
2	Modelo de processamento	Localidade do moderador e formas de acesso às bases.
3	Modelo de dados	Modelo de dados lógico utilizado na base
4	Interface da consulta	Meios de comunicação entre o sistema e o *endpoint*
5	Seleção de fontes	Métodos para selecionar as bases relevantes
6	Seletividade	Uso da seletividade da ordenação das subconsultas

Tabela 3.1: Propriedades de avaliação

consultas ASK do SPARQL na seleção de fontes (Görlitz e Staab, 2011). Uma estratégia para a decomposição de consultas baseado em ontologias é abordado pelo sistema oLinDa e apresentado na seção 3.2.1.2 (da Cunha e Lóscio, 2014). Na seção 3.2.1.3 é descrito o sistema Lusail, que utiliza variáveis globais para determinar quais as bases de dados relevantes (Abdelaziz et al., 2017a). Variações na interface da consulta e nos modelos de dados são abordados pelo sistema Ephedra na seção 3.2.1.4 (Nikolov et al., 2017). Este último sistema processa a busca federada sobre bases de terceiros e sobre uma base proprietária distribuída, vista como uma caixa-preta.

3.2.1.1 SPLENDID

O sistema SPLENDID realiza buscas sobre bases de dados RDF distribuídas e utiliza a linguagem SPARQL como interface de comunicação com os *endpoints* (Görlitz e Staab, 2011). Na fase de planejamento este sistema apresenta uma abordagem mista para selecionar as bases relevantes (VoID e consultas SPARQL ASK). Neste sistema também é proposto um método de agrupamento de padrões de triplas. Os planos de consulta são criados dinamicamente utilizando dois algoritmos de junção (*Hash* e *Bind Join*).

Durante a análise sintática a consulta SPARQL é representada internamente por uma árvore sintática. As informações estatísticas sobre cada base são retiradas de seus respectivos descritores VoID e armazenados localmente usando duas estruturas de indexação. Informações sobre a base são coletadas e armazenadas em uma das estruturas de indexação. Informações sobre predicados e tipos (predicado `rdf:type`) são armazenados usando indexação inversa, na qual cada predicado (ou tipo) contém uma lista de bases que os armazenam. Com a árvore sintática e as informações de cada base proporcionada pelo VoID, o otimizador de consulta gera um plano de consulta otimizado. O primeiro passo para criar tal plano é a reescrita da consulta, que modifica a árvore sintática produzida na entrada. Um exemplo de modificação é dividir filtros complexos (FILTER do SPARQL).

O segundo passo é selecionar as bases de dados relevantes para cada padrão de tripla. O SPLENDID utiliza os dois vetores com índices invertidos, sobre o predicado e sobre o tipo, para fazer a seleção. Para cada padrão de tripla com um predicado mapeado (com um valor da base de dados) é utilizado o vetor de índices invertidos sobre os predicados para encontrar quais as bases relevantes. Triplas tendo como predicado `rdf:type` e o objeto mapeado utilizam o vetor de índices invertidos sobre os tipos para encontrar as bases relevantes. Quando as triplas não possuem um predicado mapeado todas as bases de dados são consideradas relevantes. A seleção de fontes é refinada usando consultas SPARQL ASK. Para todas os padrões de triplas que possuem variáveis mapeadas e que não foram analisadas usando o VoID (devido aos limites do descritor) são enviadas consultas SPARQL ASK com o intuito de retirar as bases, pré-selecionadas na etapa do VoID, que não respondem à consulta. Como exemplo considere o padrão de tripla (`?s rdfs:label "ID"`). Neste exemplo primeiro são procuradas as bases contendo o predicado `rdfs:label` nos vetores criados a partir do VoID. Para cada uma das bases pré-selecionadas são enviadas consultas SPARQL ASK contendo a tripla em questão, pois o objeto da tripla foi mapeado com o valor "ID" e isto não pode ser avaliado durante a etapa anterior.

Para a criação das subconsultas, no trabalho é proposto o método de agrupamento baseado na propriedade `owl:sameAs` e nas ligações entre bases de dados utilizando tal recurso. A propriedade `owl:sameAs` determina que dois recursos, localizados em bases distintas, são equivalentes. A partir desta ligação é possível agrupar os padrões de triplas que inicialmente não foram selecionados para a base de dados que determina estas ligações.

3.2.1.2 oLinDa

O protótipo oLinDa apresenta um método de decomposição de uma consulta SPARQL sobre uma federação composta por bases de dados RDF distribuídas pertencentes a terceiros (da Cunha e Lóscio, 2014; Cunha e Lóscio, 2015). Este método explora o conceito de dados ligados (*linked data*). OLinDa considera esquemas estruturais e regras de mapeamento na seleção de fontes e na decomposição da consulta.

As ontologias que descrevem as bases são usadas como os esquemas estruturais. O moderador contém a ontologia correspondente ao esquema global das bases distribuídas, ou seja, o grafo de estrutura da base. A partir deste esquema global, o usuário consegue criar a consulta distribuída SPARQL sem a necessidade de conhecimento sobre a organização interna dos dados. O acesso à base de dados é feita por meio de *endpoints* considerados como caixas-pretas. Bases de dados de terceiros podem ser adicionados ao sistema incluindo seus respectivos *endpoints* à camada de federação. O método de decomposição da consulta proposto no trabalho é baseado em regras de mapeamento estabelecidas entre os esquemas locais e global. A Figura 3.1 exemplifica os esquemas global e locais de bases de dados reais, que apresentam informações sobre a comunidade científica (e.g., título do artigo e nome do autor). Na Figura 3.1(a) é possível ver o esquema global representado pela ontologia `SWRC` e na Figura 3.1(b) são apresentadas as bases `DBPedia`, `Kisti` e `DBLP`, onde as duas primeiras usam suas próprias ontologias e a `DBLP` utiliza a ontologia `AKT` para descrever o esquema local.

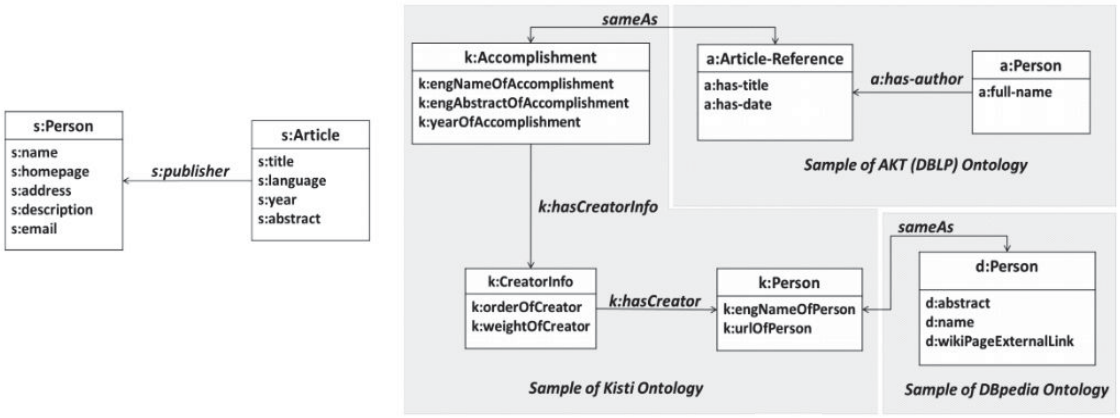


Figura 3.1: Esquema global SWRC(a) e esquemas locais Kisti, DBLP(AKT) e DBPedia(b) (Cunha e Lóscio, 2015)

Antes de efetuar a decomposição da consulta é realizada a seleção de fontes por esquemas estruturais. oLinDa cria um conjunto de mapeamentos entre o esquema global e locais, produzindo um conjunto de entidades equivalentes entre os esquemas.

A primeira etapa do mapeamento consiste em procurar classes ou propriedades correspondentes entre o esquema global com o esquema de cada base de dados. Uma otimização apresentada neste trabalho é a possibilidade de mapeamento entre bases com esquemas heterogêneos. Um exemplo de mapeamento entre ontologias heterogêneas é mostrado na Figura 3.1, no qual é possível observar que no esquema global a ligação entre um artigo (`s:Article`) a seu autor (`s:Person`) é feita apenas pela propriedade `s:publisher` e no esquema da base `Kisti` o artigo (representado pela classe `k:Accomplishment`) primeiro é ligado com a classe `k:CreatorInfo` pela propriedade `k:hasCreatorInfo`, para depois ser relacionado com o autor (classe `k:Person`) pela propriedade `k:hasCreator`. Logo, é possível observar que a propriedade `s:publisher` corresponde à combinação das propriedades `k:hasCreatorInfo` e `k:hasCreator`. Essas duas propriedades são definidas como um caminho, que é descrito como um conjunto de uma ou mais propriedades pertencentes

a uma mesma ontologia local. Também é possível observar que as classes `s:Article` e `k:Accomplishment`, referentes a artigo, são equivalentes. Essas correspondências, ou mapeamentos, são identificados e armazenados como quádruplas no formato (`v1, e1, v2, e2`). Nesta quádrupla `v1` e `v2` são os predicado do esquema global e o caminho do esquema local, respectivamente. Os parâmetros `e1` e `e2` são classes de suas respectivas propriedades. Utilizando o exemplo, a quádrupla para mapeamentos entre predicados tem o formato (`s:publisher, s:Article, [k:hasCreator.k:hasCreatorInfo], k:Accomplishment`). A criação dos mapeamentos neste trabalho pode ser um processo semi-automático, ou seja, necessita de intervenção humana para criar ou verificar a veracidade dos mapeamentos.

Após criado o conjunto de mapeamentos são geradas as regras para eles. Estas regras têm o formato `s:Person(p)` ← `a:Person(p)`, no qual o elemento a esquerda da seta representa uma entidade (predicado ou classe) do esquema global e o elemento a direita pode conter uma ou mais entidades dos esquemas locais (uma entidade global pode apresentar mais de um mapeamento local). Neste exemplo a instância p da classe `Person` referente a base `DBLP` tem uma instância p equivalente na classe `Person` do esquema global `SWRC`.

A decomposição da consulta tem três etapas. Na primeira é utilizado o analisador sintático para transformar a consulta SPARQL da entrada em um grafo formado pelos padrões de triplas. O sistema considera as cláusulas OPTIONAL e FILTER. Na segunda etapa é utilizado o conjunto de mapeamentos para determinar quais as entidades correspondentes entre o esquema global e os esquemas locais. A partir destas correspondências é possível decompor o grafo da consulta original em subgrafos da consulta destinados aos *endpoints* relevantes. Ou seja, para cada base relevante é elaborado um grafo de consulta. Na última etapa, cada subgrafo da consulta é transformado novamente em uma consulta SPARQL, os filtros são atribuídos às subconsultas e ao final são enviadas aos *endpoints*.

3.2.1.3 Lusail

O Lusail (Abdelaziz et al., 2017a,b) é um sistema que processa consultas SPARQL federadas sobre grafos RDF distribuídos entre *endpoints* acessados remotamente. O Lusail não utiliza informações sobre as bases na seleção de fontes, mas executa consultas SPARQL ASK para pré-selecionar as bases relevantes. A seleção das fontes é baseada no método de busca por variáveis de junção entre *endpoints* descrito no Capítulo 2. Com as variáveis globais determinadas, é possível decompor a consulta adequadamente. Para tal, o grafo da consulta é explorado, tendo como vértice inicial uma variável de junção. A partir do ponto inicial é realizada uma expansão em profundidade para adicionar novas triplas a uma subconsulta. Este processo é realizado enquanto as triplas pertencerem ao mesmo *endpoint* (1) e não possuírem variáveis de junção global (2). Quando não encontrar mais triplas que satisfazem estas duas condições, uma nova variável global é introduzida e expandida. Finalizado o grafo de consulta, o sistema agrupa as subconsultas que pertence ao mesmo *endpoint*. Podem existir várias formas de decompor o grafo da consulta considerando diferentes variáveis de junção como ponto inicial. Uma estimativa de custo é usada para determinar a melhor forma de decomposição.

A heurística abordada na fase de execução é baseada na estimativa da quantidade de resultados intermediários das subconsultas. Subconsultas que produzem menos resultados intermediários são classificadas como subconsultas sem atraso. Ao contrário, subconsultas com atraso possuem menor seletividade. Para determinar o atraso das subconsultas é utilizada uma heurística que estima a cardinalidade delas.

Na etapa de execução o primeiro passo é estabelecer a ordenação dos padrões de triplas. Primeiro são processadas as subconsultas com pouco atraso. Elas são requisitadas aos *endpoints* relevantes em paralelo. Os resultados obtidos nas subconsultas com pouco atraso são utilizados

para estimar o custo das subconsultas com atraso. Dentre elas é escolhida a de menor custo. Esta subconsulta é processada e os resultados intermediários são utilizados para determinar o próximo padrão de tripla. Tais subconsultas são executadas de maneira sequencial. Este processo se repete até não haver mais padrões de triplas com atraso.

3.2.1.4 Ephedra

O sistema Ephedra propõe otimizações que focam na recuperação de informações heterogêneas, as quais podem ser obtidas da bases de dados ou a partir de serviços Web (Nikolov et al., 2017). Na composição interna, as bases de dados podem ser formadas por diferentes modelos de dados como RDF ou outros modelos (p.ex., relacional) traduzidos para serem processadas utilizando um *endpoint* SPARQL. Através dos serviços Web é possível requisitar dados utilizando diferentes técnicas de processamento (p.ex., busca por dados similares e busca por palavra-chave). O Ephedra foi desenvolvido baseado na plataforma Metaphacts[1], a qual gerencia grafos de conhecimento. Esta plataforma possui uma interface gráfica, que facilita a comunicação com o usuário. A busca no sistema Metaphacts pode ocorrer sobre uma base RDF proprietária contendo o grafo de conhecimento ou uma base de dados virtualmente integrada composta por este grafo, os serviços Web e as bases RDF de terceiros (Haase et al., 2017). Esta base proprietária utiliza o Blazegraph para armazenar as triplas de maneira distribuída e escalável em um *cluster*. Mesmo possuindo uma base proprietária, ela é acessada como uma caixa-preta. O Ephedra ordena os padrões de triplas considerando uma federação formada por bases de dados RDF e serviços Web. Na junção dos resultados são considerados diferentes algoritmos.

O Ephedra utiliza a cláusula SERVICE da versão SPARQL 1.1 tanto para acessar os *endpoints* SPARQL remotos quanto para requisitar os serviços Web. A requisição dos serviços é embutida na consulta SPARQL. O acesso aos *endpoints* é realizado pelo *framework* RDF4J[2]. Para configurar um serviço como um integrante da federação são necessárias informações sobre o seu tipo e instância. O acessos aos serviços acontece por meio de wrappers (*REST API*) implementados pelo Ephedra, que considera um serviço como um módulo da federação contido no *framework* RDF4J (*SAIL module*). Este módulo tem o objetivo obter os valores das entradas localizadas dentro da cláusula SERVICE, acionar o serviço usando esses valores e mapear os resultados para as variáveis da saída.

Um exemplo de busca utilizando serviços sobre dados farmacêuticos é mostrada na Figura 3.2. O serviços são especificados através de padrões pré-determinados. Esta consulta busca pelo gene que codifica a proteína `reelin` e outros genes semelhantes. Abaixo é possível observar a utilização da busca pela palavra `reelin` (API habilitada pela base `Wikidata`) na Figura 3.2(a), uma busca pelo gene sobre uma base de dados RDF na Figura 3.2(b) e a requisição do serviço `BLAST`, que encontra entidades semelhantes, na Figura 3.2(c).

O serviços considerados pelo Ephedra são do tipo: extensão e agregação. O serviço de extensão recebe como entrada um conjunto de mapeamentos e estende este conjunto com outros mapeamentos. O serviço de agregação recebe um conjunto de resultados como entrada e produz como saída um conjunto de resultados distintos dos que foram requisitados. É utilizado um descritor de serviço para obter informações como os tipos da entrada e da saída esperadas, a cardinalidade e o formato em que tais dados devem ser expressos na cláusula SERVICE. Para o serviço de extensão o formato de entrada deste serviço é especificado como um padrão de grafo. Para o serviço de agregação o formato é representado por uma função de agregação, descrito como uma IRI.

[1] `https://www.metaphacts.com/`
[2] `https://rdf4j.eclipse.org/`

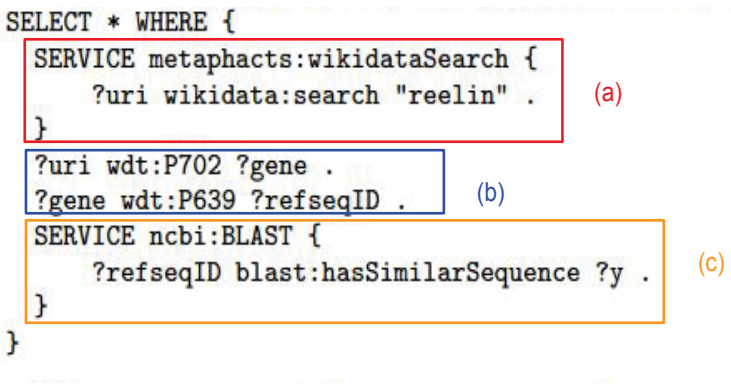


Figura 3.2: busca por palavra-chave(a), acesso à base de dados RDF(b) e serviço para encontrar similaridade(c) (Nikolov et al., 2017)

A consulta SPARQL é criada interativamente com o cliente. Devido a utilização do SPARQL 1.1 para o acesso as bases de dados RDF e serviços Web, os *endpoints* são especificados antecipadamente e descritos na cláusula SERVICE. Este processo é realizado pelo Metaphacts durante a elaboração da consulta SPARQL pelo usuário. Logo, a seleção de fontes não faz parte do Ephedra, o qual recebe a consulta SPARQL estabelecida no sistema Metaphacts.

O plano de consulta do Ephedra é adaptado para considerar os serviços Web como uma base de dados na federação. Esta modificação é necessária porque os padrões que descrevem as entradas e saídas esperadas destes serviços não disponibilizam sua estrutura RDF interna. Assim como os *endpoints* SPARQL, os serviços são acessados como caixas-pretas. Cada serviço possui sua própria cardinalidade, dificultando a criação do plano de consulta. Durante a análise de consulta é necessário adicionar os serviços à representação interna do grafo da consulta. Uma maneira é incluir estes serviços na álgebra da consulta SPARQL. A definição algébrica dos serviços inclui informações como a função implementada pelo serviço e os conjuntos de mapeamentos entre as variáveis do padrão de grafo requisitado pela cláusula SERVICE e a interface do serviço.

Após incluir os serviços na álgebra da consulta SPARQL é criado o plano de consulta. A identificação dos argumentos de junção na consulta SPARQL. Esses pode ser padrões de triplas da base RDF ou das cláusulas SERVICE. Cada SERVICE no sistema Ephedra representa uma subconsulta SPARQL ou um serviço Web. A próxima etapa da fase de planejamento é definir a ordem de junção dos argumentos. Esta escolha tem limitações devido a natureza dos serviços. O sistema considera quatro regras para a ordenação de consultas, que são aplicadas após os argumentos referentes ao mesmo *endpoint* serem agrupados. A primeira regra determina que os argumentos contendo serviços de extensão devem ser executados assim que suas entradas forem determinadas. A segunda determina que um serviço de extensão tem preferência na ordenação caso possua uma variável que possa ser mapeada usando este serviço e uma subconsulta. A terceira regra determina que um argumento tem prioridade sobre outro operando se este possui variáveis já mapeadas. A última regra está relacionada à seletividade das triplas, a qual é encontrada através da técnica de contar variáveis (grafo da consulta) ou pelo descritor de serviço.

3.2.2 Bases Centralizada e Distribuídas

Nesta seção são descritos os sistemas que processam uma consulta SPARQL tanto sobre bases distribuídas quanto sobre uma base centralizada. Logo a localidade do moderador em relação às bases dados é categorizada como local e remota. As bases de terceiros são acessadas como caixas-pretas. Ambas processam dados RDF através de consultas SPARQL. No resto da seção são descritos dois trabalhos com este tipo de processamento. Na seção 3.2.2.1 é descrito o

sistema FedX que processa uma consulta sobre dados de terceiros e utiliza consultas SPARQL ASK na seleção de fontes. O SIHJoin, descrito na seção 3.2.2.2, apresenta otimizações sobre uma base de dados proprietária centralizada e acessa as bases de terceiros através de links transversais.

3.2.2.1 FedX

FedX (Schwarte et al., 2011) é um *framework* que permite a integração virtual de bases de dados RDF heterogêneas. Este *framework* realiza busca federada sobre *endpoints* SPARQL remotos e uma base centralizada composta por cópias das bases de dados armazenadas em repositórios locais. As bases desta federação são vistas como caixas-pretas. Na seleção de fontes do FedX são utilizadas consultas SPARQL ASK para verificar as bases de dados relevantes, junto a um cache para armazenamento recente. É proposto o uso do método de grupos exclusivos, descrito no Capítulo 2, no agrupamento das subconsultas. A ordem de junção aborda a seletividade dos padrões de triplas através de uma heurística sobre as variáveis. Na junção de resultados é proposto o *Bound Join*. As otimizações propostas por este *framework* aparecem tanto na fase de planejamento quanto na fase da execução da consulta. FedX foi implementado em JAVA e utiliza o Sesame[3] para criar a base da federação. O Sesame (atualmente RDF4J) é um *framework* padrão para armazenamento, buscas e inferências em bases RDF heterogêneas. Este *framework* integra virtualmente *endpoints* SPARQL remotos junto com uma base centralizada do próprio Sesame (Rakhmawati et al., 2013). Além do acesso aos *endpoints*, o Sesame é utilizado para realizar as operações como análise da consulta SPARQL do cliente, componentes de entrada e saída (I/O) e mapeamentos do JAVA.

Na fase de planejamento da consulta o FedX analisa uma consulta SPARQL recebida do usuário e cria uma plano de consulta não otimizado. Este plano de consulta contém a árvore sintática da consulta. A próxima etapa da busca é selecionar as bases de dados relevantes. A Abordagem proposta para a seleção utiliza as consultas SPARQL ASK Para amenizar a quantidade de requisições destas consultas SPARQL ASK para os *endpoints* é utilizada um cache local para armazenar quais as bases relevantes para cada parte da consulta.

Na fase de planejamento é proposto o agrupamento de padrões de triplas por grupos exclusivos. Este agrupamento é utilizado na etapa de criação das subconsultas. O plano de consulta otimizado é criado a partir de modificações feitas na ordem em que as triplas são avaliadas. Esta nova ordem utiliza a seletividade dos padrões de tripla para criar uma lista iniciada pelas triplas mais seletivas. A seletividade de cada tripla é calculada utilizando um método semelhante ao apresentado em (Stocker et al., 2008). Este método conta as variáveis não mapeadas nos padrões de triplas. Os grupos exclusivos são colocados no início da lista, pois são considerados mais seletivos.

As subconsultas podem ser requisitadas para diferentes bases de dados, possibilitando a paralelização no envio dessas subconsultas. As subconsultas são mantidas em uma fila (FIFO) e são executadas quando possível. Também foram implementados as cláusula JOIN e UNION. Ambos levam em consideração se os dados estão armazenados na base centralizada do Sesame ou nos *endpoints* SPARQL remotos. Mesmo requisitando de maneira direta as triplas através de funções do Sesame, os repositórios contidos na base centralizada são acessados como caixas-pretas. Ou seja, o FedX requisita os padrões de triplas usando a API do Sesame, que processa a consulta de maneira autônoma e retorna o resultado ao *framework*.

[3] `http://archive.rdf4j.org/`

3.2.2.2 SIHJoin

No *Symmetric Index Hash Join* (SIHJoin) é proposto um novo operador de junção (Ladwig e Tran, 2011). Este operador acessa uma base centralizada proprietária como uma caixa-branca. Esta base proprietária utiliza indexação de dados para armazenar e ordenar as triplas. A indexação foi realizada com o Semplore (Zhang et al., 2007). Este motor utiliza índices invertidos para armazenar as triplas, nos quais os predicados (termos) possuem uma lista de sujeitos (documentos), e cada sujeito apresenta uma lista de objetos relacionados junto ao predicado (posições dos termos nos documentos). A base de triplas e a indexação do Semplore foram implementadas com o motor de busca textual Lucene[4]. Ele possui métodos de indexação, no qual o motor é instalado em uma máquina local e as triplas RDF são armazenadas de maneira centralizada. Como modelo de processamento de consulta são utilizados links transversais sobre *endpoints* SPARQL remotos acessados como caixas-pretas e busca por índices sobre uma base proprietária vista como uma caixa-branca. O SIHJoin considera, no processamento por links transversais, que algumas das bases de dados de terceiros são conhecidas previamente. Uma abordagem semelhante de processamento distribuído é apresentado em (Umbrich et al., 2012), que também utiliza links transversais sobre *endpoints* SPARQL e uma base centralizada, porém esta última é vista como caixa-preta.

A otimização principal proposta pelo SIHJoin (*Symmetric Index Hash Join*) é baseada na criação da base centralizada proprietária, com o objetivo de melhorar o desempenho da consulta (diminuir o tempo de execução). Esta base armazena as triplas e aborda o conceito de indexação para realizar acesso direto sobre as triplas. Através dos índices, é possível instanciar padrões de triplas com dados obtidos durante a execução. A finalidade é diminuir a quantidade de resultados intermediários criando subconsultas que produzam resultados intermediários relevantes. Ou seja, diminuindo o processamento de subconsultas menos seletivas. O operador proposto no trabalho recebe o nome de SIHJ.

A base proprietária é requisitada usando um método de acesso contido em cada operador SIHJ da árvore sintática. O modelo de acesso tem a função de encontrar os mapeamentos das variáveis contidas no padrão de tripla requisitado. A busca é realizada com auxílio da indexação dos dados. Diferente das bases de terceiros que enviam seus resultados diretamente ao operador, para obter os resultados sobre a base proprietária é necessário realizar uma requisição ao método de acesso contendo o padrão de tripla procurado (*Pull based*).

A entrada é uma consulta SPARQL, a qual é analisada e transformada em uma árvore sintática construída com os operadores SIHJ. Esta árvore é usada para criar o plano de consulta exemplificado na Figura 3.3. Operadores em níveis mais baixos na árvore enviam seus resultados para os operadores no nível superior (*push based*) à medida que são encontrados. A entrada do lado direito do operador está ligado às bases de dados e a entrada do lado esquerdo recebe os valores passados pelo operadores inferiores. A execução de uma operação de junção usando SIHJoin começa no operador à esquerda, que recebe os resultados intermediários do nível inferior da árvore e armazena em uma tabela *hash*. Esta tabela é utilizada para verificar os mapeamentos da variável de junção do padrão de tripla contido no operador corrente (tripla localizada na entrada à direita). Para um mapeamento ainda não analisado, o sistema utiliza esse mapeamento para determinar o valor da variável de junção do padrão de tripla. Então as subconsultas são examinadas na base proprietária, através de uma requisição ao módulo de acesso, e nos *endpoints* remotos. Os resultados obtidos são armazenados na tabela *hash* do lado direito. O objetivo deste processo é diminuir a quantidade de resultados intermediários utilizando os mapeamentos

[4] `http://lucene.apache.org/core/`

enviados pela camada inferior para elaborar subconsultas mais seletivas. Ao final, as tabelas são comparadas e os resultados são reunidos e enviados à camada superior.

Na Figura 3.3 é apresentado um exemplo de execução do operador SIHJoin. As operações começam nas folhas devido ao seu processamento *push based*. Na primeira iteração é possível observar uma exceção, no qual as duas entradas estão conectadas às bases de dados. Nesta iteração são avaliados os padrões de triplas (`?x worksAt dbpedia:KIT`) e (`?x knows ?y`). Os mapeamentos obtidos são armazenados na tabela *hash* à esquerda da segunda iteração. Estes resultados são utilizados para mapear a variável `?y` do padrão de tripla (`?y name ?n`). Ao final da segunda iteração os resultados das duas tabelas são reunidas e enviadas ao operador seguinte.

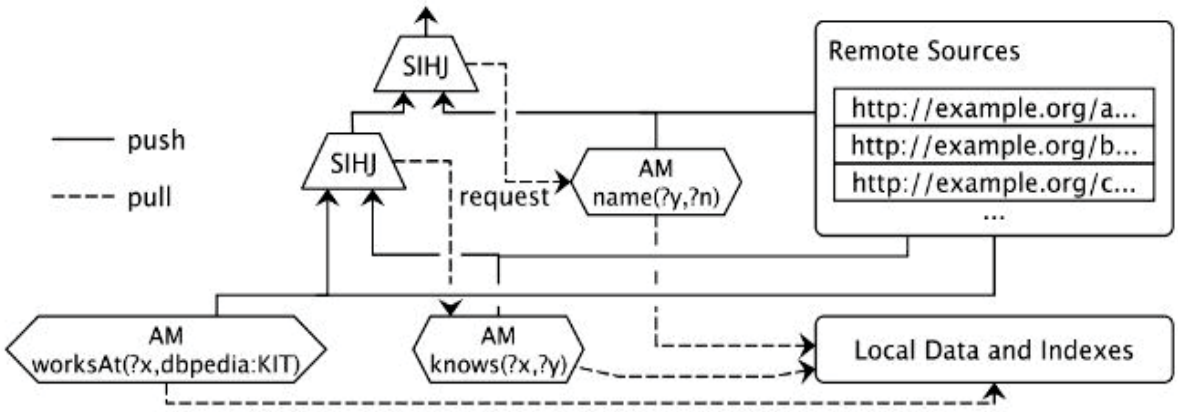


Figura 3.3: Plano de consulta com operadores SIHJoin (Ladwig e Tran, 2011)

Um modelo para determinar o custo do plano de consulta pode ser usado para otimizar a ordem de junção. No modelo de custo proposto no SIHJoin, o custo de um operador SHIJoin é a soma do custo de junção dos resultados à esquerda, dos resultados à direita e das requisições aos métodos de acesso. O custo de junção dos mapeamentos à esquerda relacionam o peso de inserir dados na tabela *hash*, a seletividade da junção, a criação dos resultados, a requisição de acesso à base proprietária e a quantidade de resultados intermediários. O custo de junção à direita é semelhante, porém não há necessidade de requisições aos métodos de acesso, pois tais requisições foram realizadas na junção à esquerda. O custo do método de acesso considera a análise dos índices locais e a transferências dos resultados.

3.3 CONSIDERAÇÕES

Existem várias abordagens para a realização de uma consulta distribuída sobre bases de dados RDF integradas. Neste capítulo foram apresentadas seis abordagens que efetuam busca distribuída. A Tabela 3.2 mostra as propriedades 3–6 nestes sistemas, de acordo com a Seção 3.1.

É possível observar que a maioria dos sistemas tem como modelo de dados o padrão RDF e todos utilizam a linguagem de consulta SPARQL. Em contrapartida, o Ephedra admite o modelo relacional e o uso da cláusula SERVICE para criar requisições a serviços Web embutidas nas consultas SPARQL. Na seleção de fontes, alguns sistema utilizam as consultas SPARQL ASK para determinar as bases relevantes e outros utilizam informações e metadados da base. Ephedra não possui seleção de fontes devido a utilização da cláusula SERVICE do SPARQL 1.1, na qual as bases são especificadas durante a elaboração da consulta SPARQL pelo usuário. A maioria dos sistema utiliza a seletividade para a ordenação das subconsultas. Alguns dos sistemas utilizam a contagem de variáveis para determinar a seletividade dos padrões de triplas.

Sistemas de busca distribuída				
Trabalhos	**Modelo Dados**	**Seleção Fontes**	**Seletividade**	**Interface**
FedX	RDF	ASK SPARQL	Sim	SPARQL
SPLENDID	RDF	VoiD, Indexação e ASK SPARQL	Sim	SPARQL
oLinDa	RDF	Esquemas e regras de mapeamento	N. E.	SPARQL
Lusail	RDF	ASK SPARQL e variáveis de junção global	Sim	SPARQL
Ephedra	RDF, Relacional	N.E.	Sim	SPARQL, serviços Web
SIHJoin	RDF	Links transversais	N.E.	SPARQL

Tabela 3.2: Etapas de uma busca em sistemas distribuídos. **N.E.** Não Especificado.

Outros sistemas apresentam heurísticas mais complexas ou utilizam metadados das bases para determinar a ordenação dos padrões de triplas. O Ephedra adicionalmente utiliza descritores de serviço aos metadados para organizar as subconsultas.

Na tabela 3.3 são apresentadas as características principais abordadas nesta dissertação, apresentados nas propriedades 1–2. Nesta tabela é possível observar que alguns dos sistemas utilizam uma mesclagem na abordagem de armazenamento de dados, composta por bases proprietárias e pertencentes a terceiros. Todos os trabalhos relacionados processam sobre bases RDF de terceiros. Porém apenas algumas apresentam bases proprietárias em sua composição. Dentre os sistemas, FedX processa parte da busca sobre repositórios armazenados em uma base centralizada. Estes repositórios contém cópias das bases de dados de terceiros, que são acessadas como caixas-pretas. O Ephedra consegue armazenar um grafo de dados de maneira distribuída em uma base proprietária e processar a consulta sobre *endpoints* SPARQL remotos. Mesmo possuindo uma base proprietária na federação, ele acessa todas as bases através de requisições SPARQL sobre *endpoints* considerados caixas-pretas. Em SIHJoin, é integrada ao sistema uma base proprietária centralizada criada a partir da indexação das triplas de dados. Neste sistema a base proprietária é vista como uma caixa-branca. Desta forma a busca por triplas é feita de maneira direta (busca por índices).

A última linha da tabela é apresentado o sistema proposto nesta dissertação. Ele realiza uma busca federada sobre bases de dados de terceiros e uma base proprietária distribuída vista como caixa-branca. Sobre esta base são realizadas otimizações com o objetivo de descentralizar o processamento da consulta realizado pelo moderador, que tem a finalidade de otimizar o tempo da consulta. Para o processamento da consulta sobre os *endpoints* SPARQL remotos são criadas subconsultas dentro dos servidores de dados da base proprietária. Essas subconsultas são requisitadas aos *endpoints* e os resultados são juntados no próprio servidor que realizou a

requisição. Sobre as características de uma consulta distribuída, o sistema FeSHyD utiliza o modelo RDF. Para a Seleção de fontes é utilizado o método casamento de esquemas estruturais.

	Base Proprietária		**Base de Terceiros**	
	Armazenamento	**Acesso**	**Armazenamento**	**Acesso**
SPLENDID	X	X	Distribuída	Caixa-Preta (SPARQL)
oLinDa	X	X	Distribuída	Caixa-Preta (SPARQL)
Lusail	X	X	Distribuída	Caixa-Preta (SPARQL)
FedX	X	X	Centralizada e Distribuída	Caixa-Preta (SPARQL)
Ephedra	Distribuída	Caixa-Preta (SPARQL)	Distribuída	Caixa-Preta (SPARQL)
SIHJoin	Centralizada	Caixa-Branca	Distribuída	Caixa-Preta (SPARQL)
FeSHyD	Distribuída	Caixa-Branca	Distribuída	Caixa-Preta (SPARQL)

Tabela 3.3: Sistemas que executam consultas SPARQL em bases RDF distribuídas

4 SISTEMA FESHYD

Este capítulo apresenta a proposta do sistema denominado FeSHyD *(**Fe**derated **S**earch on **Hy**brid **D**atabases)*, que viabiliza o processamento de consultas SPARQL federadas em uma base de dados RDF distribuída e híbrida composta por uma base proprietária distribuída remota (tratada como caixa-branca) e por bases de terceiros (acessadas como caixas-pretas por meio de *endpoints* SPARQL).

De uma maneira geral, há 2 alternativas para o processamento de consultas SPARQL que envolvam dados de uma base proprietária e bases de terceiros. As alternativas são: *(i)* utilizar um sistema de consultas federado onde todas as bases, proprietária ou de terceiros, são tratadas como uma caixa-preta. Nesse caso o servidor mediador requisita dados das bases e gerencia todo o processamento da consulta; e, *(ii)* ter uma intervenção do usuário para obter resultados intermediários da base proprietária da empresa para encaminhar aos *endpoints* SPARQL. O FeSHyD oferece uma terceira alternativa, que é viabilizar uma comunicação direta da base proprietária com os *endpoints* durante o processamento de consultas. A hipótese é que o tratamento da base proprietária distribuída como caixa-branca ofereça oportunidades de otimização do processamento da consulta que não são possíveis nas demais alternativas.

O capítulo está organizado da seguinte forma. A Seção 4.1 apresenta a arquitetura do sistema. A Seção 4.2 apresenta o modelo de dados e a linguagem de consulta utilizada no FeSHyD. A Seção 4.3 descreve as etapas envolvidas no planejamento de consultas. A Seção 4.4 apresenta o processo de execução de consultas e a Seção 4.5 conclui o capítulo apresentando uma discussão sobre a proposta.

4.1 ARQUITETURA DO FESHYD

A Figura 4.1 mostra a arquitetura do FeSHyD. Seus principais componentes são o moderador, os servidores que compõem a base proprietária distribuída e as bases externas. Para o detalhamento de como uma consulta é processada pelo sistema, considere os fluxos numerados na figura. Uma requisição SPARQL é recebida pelo moderador (1) e enviada para o *Módulo de Planejamento*, responsável pela geração de um plano de consulta, utilizando as informações armazenadas na base de metadados (2). Esta base contém, dentre outras informações, o esquema das bases proprietária e externas. A geração do plano inicia com a análise léxica e sintática (módulo *Análise da consulta*), seguida pela seleção das fontes de dados envolvidas na consulta e geração do plano de consulta (módulo *Geração do plano de consulta*). O plano define um percurso no grafo RDF. Ele é enviado para o *Módulo de execução* da consulta (3), que o encaminha para todos os servidores que compõem a base proprietária distribuída (4). Todos os servidores executam o mesmo plano de consulta em paralelo, com os dados armazenados localmente (G_{D_i}). Durante a execução de consultas, quando o percurso no grafo envolve dados armazenados externamente, as trocas de mensagens são realizadas sem a intervenção do moderador. Caso o dado necessário esteja em um servidor da base proprietária, o acesso direto é possível devido a existência de índices ($Índices_{D_i}$), que permitem que um servidor obtenha o endereço físico dos demais servidores (5). Caso o dado necessário esteja localizado em uma base de terceiros, o módulo de *Geração da subconsulta e junção dos resultados*, gera a consulta SPARQL (6), que é enviada ao *endpoint* (7). Assim que os resultados são recebidos, é feita então a junção. Ao final da execução do plano, cada servidor envia seus resultados para o moderador (8). O moderador é

responsável por unir todos estes resultados e encaminhá-los ao usuário (9). Nas próximas seções, o modelo de dados e os principais módulos do FeSHyD são detalhados.

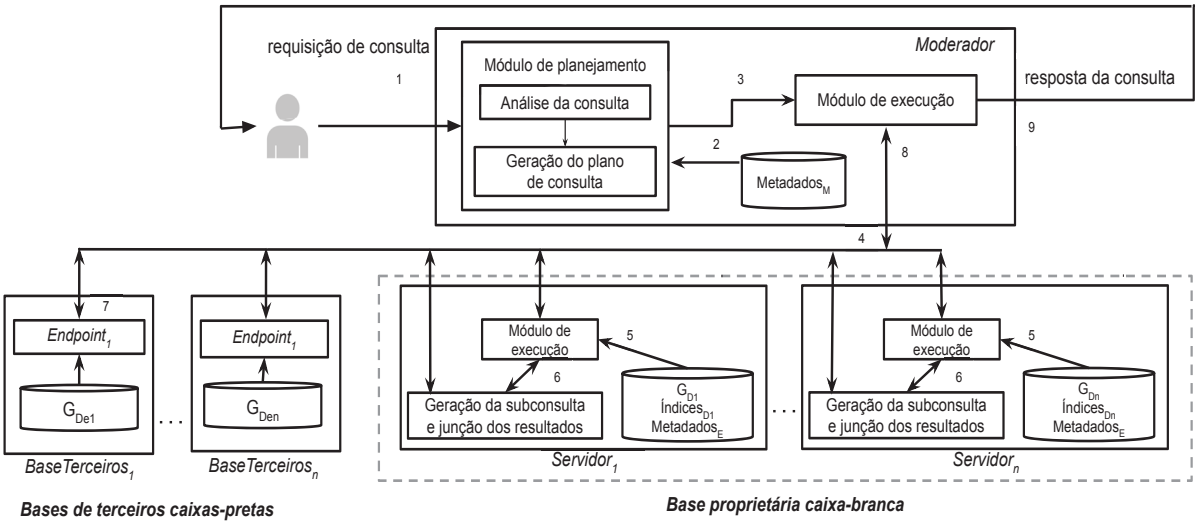


Figura 4.1: Arquitetura do sistema FeSHyD estendida a partir do sistema PAbS

4.2 MODELO DE DADOS E LINGUAGEM DE CONSULTA

O modelo de dados utilizado pelo FeSHyD é o RDF. Nesta dissertação assume-se que os sujeitos estão armazenados com os seus objetos literais, bem como com a sua lista de adjacência, que inclui as arestas que partem e chegam ao vértice; ou seja, a base está fragmentada seguindo o padrão estrela. Um exemplo de grafo RDF de uma aplicação de comércio eletrônico, baseado no benchmark Berlin, é apresentado na Figura 4.2(a). Cada fragmento no formato estrela da base proprietária está delimitado com um fundo cinza. Observe que na base proprietária os dados estão distribuídos em dois servidores, W e X, e que há três bases de terceiros.

Embora o FeSHyD só tenha acesso às bases de terceiros através de consultas SPARQL, assume-se que a parte do esquema que é de interesse da aplicação é conhecida. O grafo de estrutura da Figura 4.2(b) ilustra tanto o esquema da base proprietária, como das bases de terceiros. Assume-se que as ligações conhecidas envolvem a base proprietária e uma base de terceiros, mas as ligações entre duas bases de terceiros distintas não são conhecidas. O FeSHyD mantém em seus metadados o endereço (url_{ep}) de cada base de terceiros ep. É considerada a existência da função *end(v)* sobre os vértices do grafo de estrutura, definida da seguinte forma: $end(v) = url$, caso v pertença a uma base de terceiros com endereço url; e $end(v) = \texttt{prop}$, caso v pertença à base proprietária, onde `prop` é uma constante.

O FeSHyD dá suporte à porção conjuntiva da linguagem de consultas SPARQL, na qual predicados são constantes, enquanto sujeitos e objetos são variáveis, que podem ser associadas a constantes na cláusula FILTER. Um exemplo de consulta é apresentada na Figura 4.3(a). Ela obtém os valores de ofertas com valores menores ou iguais a 58000 com os nomes de seus respectivos vendedores. A representação de um grafo de consulta é similar à representação da base RDF, como ilustra a Figura 4.3(b), com filtros associados a nodos. Assim, uma consulta q é definida como um par $(G_q, result_q)$, onde G_q é um conjunto de pares $((s, p, o), F)$, nos quais (s, p, o) é uma aresta do grafo e F é um filtro aplicado sobre s ou o; e $result_q$ é o conjunto de variáveis retornados pela consulta. A consulta da Figura 4.3(a), é definida como: $q = (\{ ((?offer, value, ?valueOffer), \{?valueOffer \leq 58000\}), ((?offer, is\_sold, ?vendor), \{\}), ((?vendor, name, ?nameVendor), \{\})\}, \{?valueOffer, ?nameVendor\})$. Nesta dissertação assume-se que o grafo (G_q) é conexo.

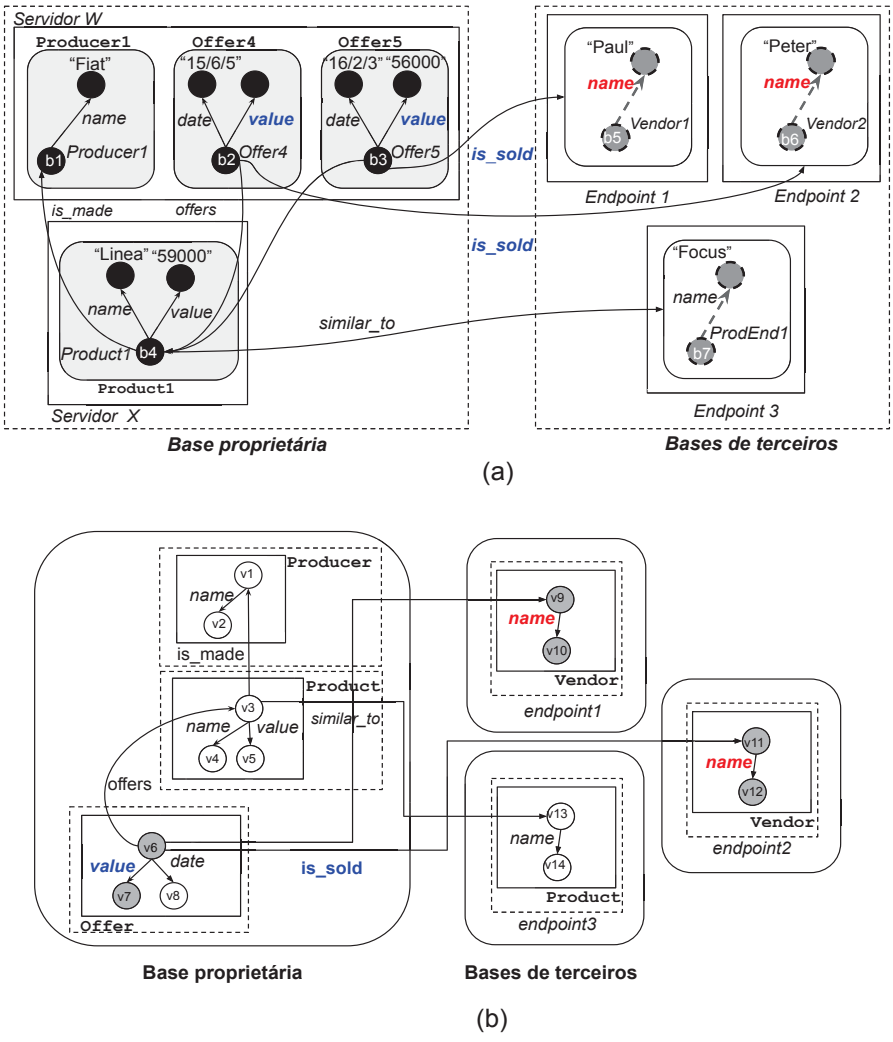


Figura 4.2: Base RDF distribuída (a) e seu respectivo grafo de estrutura (b)

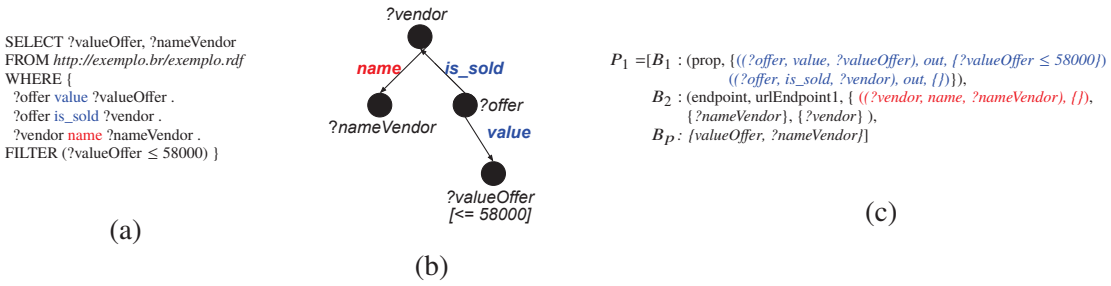


Figura 4.3: Consulta SPARQL (a); Grafo da consulta (b); Plano de execução (c)

4.3 PLANEJAMENTO DE CONSULTAS

Dada a arquitetura proposta na Figura 4.1 e uma consulta SPARQL, o *módulo de análise* realiza a análise léxica e sintática da consulta, gerando o grafo da consulta. Por exemplo, o grafo da Figura 4.3(b) seria gerado para a consulta da Figura 4.3(a). O grafo gerado é utilizado na etapa seguinte, executada pelo *módulo de geração do plano*. Esta etapa inicia com a seleção de fontes, que tem como objetivo determinar quais são as bases de dados relevantes para o seu processamento. Utilizando as informações obtidas na seleção de fontes, um plano de consulta é gerado. O plano determina as triplas processadas nas bases proprietária e de terceiros. Continuando com o exemplo, o plano da Figura 4.3(c) seria gerado para a consulta da Figura

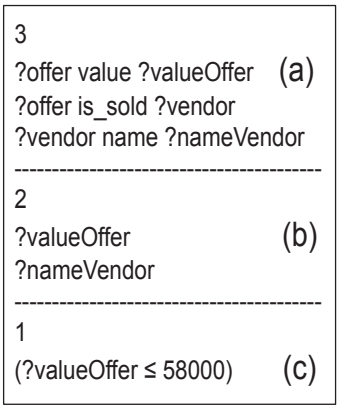


Figura 4.4: Representação interna da consulta: Padrões de triplas (a); Variáveis de projeção (b); Filtros (c)

4.3(a). Nesse plano os padrões de triplas são agrupados em blocos. No exemplo, o bloco B_1 contém padrões de triplas da base proprietária e o bloco B_2 contém um padrão de tripla dos *endpoints*. Nesta seção são detalhados os passos para a geração do plano de consulta.

4.3.1 Análise de consultas SPARQL

A análise de uma consulta SPARQL pode ser dividida em duas etapas: análise (léxica e sintática) e a geração da representação interna da consulta. Na primeira etapa, um analisador léxico e sintático verifica a gramática da linguagem SPARQL. O analisador léxico identifica as palavras-chaves da linguagem SPARQL dentro da consulta, como `SELECT` e `WHERE`, no qual ele foi configurado para reconhecer consultas SPARQL conjuntivas. Este tipo de consulta contempla os padrões de triplas contendo variáveis e recursos IRI, como descrito no Capítulo 2. Também é possível reconhecer outros operadores como o `PREFIX`, `FROM`, `BASE` e `FILTER`. Os filtros identificáveis na análise são os que utilizam comparadores algébricos. A Figura 4.3 (a) apresenta um exemplo de consulta SPARQL conjuntiva que contém um filtro e um grafo base determinado pelo operador `FROM`. A Figura 4.3 (b) apresenta o grafo de consulta correspondente à consulta da Figura 4.3(a). Durante análise da consulta são identificados os padrões de triplas, variáveis de projeção e filtros da consulta.

Durante a segunda etapa da análise, a representação interna da consulta é armazenada em um arquivo, que é utilizado para a criação do plano de consulta estendido. A Figura 4.4 ilustra o arquivo gerado para a consulta exemplo. Além dos padrões de triplas que compõem o grafo da consulta (a), as informações sobre as variáveis de projeção (b) e os filtros (c) são armazenados junto à representação interna do grafo de consulta.

4.3.2 Seleção de fontes

Como descrito no Capítulo 2, na fase de planejamento de uma busca federada há o desafio de selecionar as bases de dados relevantes para o processamento da consulta. Este desafio tem o objetivo de diminuir o custo de comunicação do sistema com os *endpoints*, que não contribuem para a geração dos resultados. Algumas das técnicas de seleção foram descritas no Capítulo 2. Elas envolvem metadados sobre as bases ou consultas booleanas SPARQL para determinar a existências das triplas de dados. Como no FeSHyD a parte da estrutura das bases de terceiros que é de interesse da aplicação é conhecida, a seleção de fontes é baseada em um algoritmo para identificar os subgrafos homomórficos ao grafo da consulta no grafo de estrutura da base de dados.

A seleção de fontes é executada pelo módulo de *geração do plano de consulta* do FeSHyD. Esse processo tem como finalidade determinar as bases de dados relevantes. Para isso primeiramente são encontrados os mapeamentos do grafo da consulta G_q para o grafo de estrutura G_s, ou seja, são associados valores de G_s para as variáveis de G_q. Para a atribuição dos padrões de tripla às bases proprietária e de terceiros, foram definidas as funções *map* e *end* neste trabalho. A função *map* determina o mapeamento para uma variável de G_q para um valor de G_S. A função *end* determina o tipo de base a partir de uma vértice de G_S.

O processo de mapeamento é ilustrado pelo Algoritmo 1 e Algoritmo 2. A função `FindMappings` do Algoritmo 1 retorna um conjunto de mapeamentos `M` usando como entrada um grafo de estrutura da base distribuída e híbrida (G_S) e um grafo da consulta (G_Q). A função `FindMappings` utiliza a função recursiva `FindSubgraph` do Algoritmo 2.

A primeira etapa do Algoritmo 1 é a reordenação dos padrões de triplas da consulta com a finalidade de identificar seus padrões estrela. Este processo é realizado na linha 6 pela função `reorder`. Essa função avalia o padrão de tripla inicial e inclui os padrões de triplas correspondentes ao padrão estrela inicial em uma lista denominada S_{en}. Ou seja, todos os padrões que possuem o mesmo sujeito que a tripla inicial são incluídos em S_{en}. A partir dessa lista, os objetos das triplas que são variáveis são analisadas para determinar se formam novos padrões estrela. Em caso afirmativo, os padrões de triplas que os compõem são inseridos em S_{en}. Esse processo é repetido até que todos os padrões de triplas sejam analisados. Considere a consulta do exemplo da Figura 4.3(a). O padrão inicial é o (?*offer*, $value$, ?*valueOffer*). Ele faz parte de um padrão estrela com o sujeito (?*offer*) e composto por mais um padrão de tripla, que no caso é o padrão (?*offer*, is_sold, ?$vendor$). A partir deste padrão são exploradas as variáveis (?*valueOffer*) e (?$vendor$). No exemplo, apenas a variável (?$vendor$) está contida em um padrão de tripla (?$vendor$, $name$, ?$nameVendor$), portanto o padrão estrela desta variável é adicionado a s_{en}. Logo, S_{en} = {{(?*offer*, $value$, ?*valueOffer*), ?*valueOffer* ≤ 58000}, {(?*offer*, is_sold, ?$vendor$)}, {(?$vendor$, $name$, ?$nameVendor$)}}. Como o objetivo é ordenar os padrões de triplas, não foi utilizada uma heurística complexa para determinar o padrão de tripla inicial. Portanto foi estabelecido como padrão inicial o padrão estrela de maior grau.

Continuando com a descrição do Algoritmo 1, o primeiro padrão de tripla da consulta, representado por (?s_1, p, ?o_1), é identificado, junto a um possível filtro `F`, e armazenado em S_1 (linha 7). No exemplo, S_1 = `(?offer, value, ?valueOffer)`. Nas linhas 8 a 10, o predicado `p` do primeiro padrão de tripla é utilizado para determinar os pontos inicias para a exploração no grafo de estrutura. É possível que diferentes triplas do grafo de estrutura possuam o predicado inicial. Logo, diferentes vértices do grafo de estrutura podem ser utilizados como ponto inicial de exploração. Para tal, são buscadas todas as triplas (?s', p, ?o') em G_S, e o sujeito desta tripla é considerado como um ponto inicial de exploração `pi`. Considerando o grafo de estrutura da Figura 4.2(b) e S_1, é verificado que tanto `Product` quanto `Offer` possuem o predicado `value` em G_S. Portanto, `v3` e `v6` representam pontos iniciais de exploração e *pontosIniciais* = {$v3$, $v6$}.

Nas linhas 11 a 16 são inicializadas as buscas para cada ponto inicial de exploração. As variáveis já visitadas são armazenadas em `vars` (linha 12), a fim de encontrar variáveis de junção entre o grafo da consulta parcialmente explorado e os padrões de triplas. O conjunto `map` contém os mapeamentos entre G_q e G_S. Na linha 13 é criado o primeiro mapeamento entre o sujeito `?s` do padrão de tripla inicial da consulta e um dos pontos iniciais `pi`. No exemplo, considerando o ponto inicial $v6$, `vars` = {*?offer*} e `map` = {*?offer* ↦ $v6$}.

Na linha 14 a função `FindSubgraph` é chamada para o primeiro padrão de tripla. A função tem os parâmetros: (1) conjunto de todos os conjuntos de mapeamentos `M`; (2) conjunto de variáveis usadas `vars`; (3) conjunto de mapeamentos criado durante a recursão `map`; (4) o padrão

Algoritmo 1: Detecção de mapeamentos

```
1  Function FindMappings((G_S,G_Q))
2      pontosIniciais:= {};
3      vars:= {};
4      M:= {};
5      map:= {};
6      S_I = reorder (G_Q);
7      let S_1 = {(?s_1,p,?o_1),F};
8      for each edge (s',p,o') ∈ G_S do
9          insert s' in pontosIniciais;
10     end
11     for each pi ∈ pontosIniciais do
12         vars = {?s_1};
13         map = {?s_1 ↦ pi};
14         FindSubgraph( M, vars, map, S_1, 1, G_s, S_en);
15         return M;
16     end
17 End Function
```

de tripla S_i analisado em determinado passo da recursão; (5) o contador de padrões de triplas analisadas `i`; (6) o grafo de estrutura G_S; (7) a lista ordenada S_{en}. No exemplo, a chamada da função seria $FindSubgraph(\{\}, \{?offer\}, \{?offer \mapsto v6\}, \{?offer, value, ?valueOffer\}, 1, G_S, S_{en})$. Por fim, na linha 15 o conjunto de todos os mapeamentos é retornado.

O Algoritmo 2 descreve a função recursiva `FindSubgraph`. Esta função recursiva determina os diferentes caminhos no grafo de estrutura para uma mesma aresta (predicado). A cada chamada recursiva do algoritmo (linhas 9, 19 ou 26) um novo padrão de tripla da consulta é considerado. Dessa forma, as chamadas recursivas viabilizam a detecção de subgrafos homomórficos no grafo de estrutura dado o grafo de uma consulta.

A função `FindSubgraph` inicia verificando se foi possível encontrar um subgrafo homomórfico ao grafo da consulta a partir da exploração de um caminho no grafo de estrutura (linhas 2 a 5). Na linha 2 é verificado se todos os padrões de triplas foram analisados através da comparação entre o contador `i` e a quantidade de padrões de triplas em S_{en}. Em caso afirmativo, o mapeamento é incluído no resultado M. Caso contrário, o padrão de tripla S_i=`((?s,p,?o),F)` é avaliado (linha 6). No exemplo, o primeiro padrão de tripla analisado é S_i = ((?*offer*, *value*, ?*valueOffer*), ?*valueOffer* ≤ 58000). É verificado se existe pelo menos uma variável de ligação do padrão de tripla com variáveis já processadas (`vars`). No exemplo. o padrão de tripla S_i possui como sujeito a variável *?offer* e como objeto a variável *?valueOffer*, que são comparadas com as variáveis exploradas `vars` = {*?offer*}. Há trés possibilidades: o sujeito do padrão de tripla pertence à `vars` (linha 14); o objeto pertence à `vars` (linha 21); ou, ambos pertencem à `vars` (linha 7).

Nas linhas 7 a 11, caso o sujeito e o objeto pertençam a `vars` e existe uma tripla (`v1,p,v2`) com mapeamentos correspondentes a estas variáveis, e com o mesmo predicado `p` no grafo de estrutura G_s então a tripla S_i é considerada analisada pois nenhum novo mapeamento é encontrado. Portanto, uma nova chamada recursiva é realizada (linha 9). Nas linhas 12 a 13 são criadas cópias de `vars` e `maps` utilizadas para as possíveis chamadas recursivas durante o processo de encontrar o subgrafo homomórfico. Nas linhas 14 a 20 é tratado o caso em que apenas o sujeito do padrão de tripla S_i pertence a `vars`. No exemplo, o sujeito *?offer* pertence tanto a `vars` quanto ao padrão de tripla S_i. Como o sujeito já foi analisado previamente e teve seu mapeamento adicionado a `maps`, o sujeito então é mapeado com o vértice correspondente no grafo de estrutura (linha 15). Com a atribuição realizada, o vértice correspondente ao sujeito `?s` e o predicado `p`, pertencentes a S_i, são buscados em G_s (linha 16). No exemplo, o sujeito *?offer* é mapeado com o valor *v*6 e a tripla S_i = {v6, *value*, ?*valueOffer*} é buscada em G_s. Caso exista uma tripla `(v1,p,v2)` homomórfica ao padrão de tripla s_i no grafo de estrutura G_s então

Algoritmo 2: Detecção de subgrafos

```
1  Function FindSubgraph (( M, vars, map, S_i, i, G_s, S_en))
2      if i > |S_en| then
3          insert map in M;
4          return;
5      end
6      let S_i = {(?s,p,?o),F};
7      if (?s ∈ vars) and (?o ∈ vars) then
8          if there exists an edge (v1,p,v2) ∈ G_S then
9              FindSubgraph ( M, vars, map, S_i+1, 1+1, G_s, S_en);
10         end
11     else
12         vars_ant := vars;
13         map_ant := map;
14         if ?s ∈ vars then
15             let (?s ↦ v1) ∈ map;
16             for each edge (v1,p,v2) ∈ G_S do
17                 vars = vars_ant ∪ {?o};
18                 map = map_ant ∪ {?o ↦ v2};
19                 FindSubgraph ( M, vars, map, S_i+1, 1+1, G_s, S_en);
20             end
21         else
22             let (?o ↦ v2) ∈ map;
23             for each edge (v1,p,v2) ∈ G_S do
24                 vars = vars ∪ {?s};
25                 map = map_ant ∪ {?s ↦ v1};
26                 FindSubgraph ( M, vars, map, S_i+1, 1+1, G_s, S_en);
27             end
28         end
29     end
30 End Function
```

um mapeamento entre `v2` e o objeto do padrão de tripla `?o` analisado é criado. No exemplo, ?*valueOffer* é mapeado com o valor $v7$.

Posteriormente, o objeto `?o` é adicionado a `vars` e o mapeamento correspondente é adicionado a `maps`. Uma chamada recursiva é criada para este caminho na exploração de G_s. Seguindo o exemplo, a próxima chamada recursiva para o padrão de tripla seguinte é $FindSubgraph(\{\}, \{?offer, ?valueOffer\}\}, \{?offer \mapsto v6, ?valueOffer \mapsto v7\}, \{?offer, is\_sold, ?vendor\}, 2, G_S, S_{en})$. Caso exista mais de uma tripla `(v1,p,v2)` homomórfica outro caminho de exploração é determinado com um valor diferente para `v2` e um outro conjunto de mapeamentos é criado. Consequentemente, uma nova chamada recursiva é criada. O processo de exploração análogo para o objeto de s_i é apresentado entre as linhas 21 a 28.

Continuando nosso exemplo, que considera o ponto inicial de exploração `v6`, o próximo padrão de tripla S_i é $\{?offer, is\_sold, ?vendor\}$. Nesse padrão de tripla é verificado que o sujeito ?*offer* pertence ao conjunto `vars={`*?offer*, *?valueOffer*`}`. Logo, o sujeito é mapeado com o valor correspondente ($v6$) e o padrão de tripla $\{v6, is\_sold, ?vendor\}$ é buscado em G_S. Esse padrão de tripla apresenta duas possibilidades de mapeamento para o objeto, ($?vendor \mapsto v9$) e ($?vendor \mapsto v11$), um para cada *endpoint* SPARQL. Através do algoritmo recursivo é possível identificar dois conjuntos de mapeamentos a partir do ponto da recursão atual. Para o `Endpoint1` o grafo de estrutura é explorado e a tripla homomórfica $\{v6, is\_sold, v9\}$ é descoberta. Para o `Endpoint2` também é encontrado um mapeamento para o objeto $\{v6, is\_sold, v11\}$ Para ambas as possibilidade é dada a continuidade na recursão. Para a execução sobre o `Endpoint1` o padrão de tripla $(?vendor, name, ?nameVendor)$ é analisado. Do mesmo modo que os padrões anteriores, o sujeito dessa tripla pertence a `vars`. Logo, a variável $?vendor$ é associado ao valor $v9$. Durante a exploração de G_s a tripla homomórfica $\{v9, name, v10\}$ ao padrão de tripla é encontrada. Como é possível observar na Figura 4.2 (a) ambos os *endpoints* apresentam a mesma estrutura. No `Endpoint2` o processo de busca é o mesmo, e o resultado da exploração do grafo

de G_s é a tripla $\{v11, name, v12\}$. Ao final, os dois conjuntos de mapeamentos são adicionados ao conjunto M: $map_1 = \{?offer \mapsto v_6, ?valueOffer \mapsto v_7, ?vendor \mapsto v_9, ?nameVendor \mapsto v_{10}\}$ e $map_2 = \{?offer \mapsto v_6, ?valueOffer \mapsto v_7, ?vendor \mapsto v_{11}, ?nameVendor \mapsto v_{12}\}$. .

O mesmo processo executado para v6 seria executado para o ponto inicial v3. Entretanto, quando executada a busca de mapeamento com o padrão de tripla $\{?offer, is\_sold, ?vendor\}$ essa busca não retornaria resultado. Logo, o ponto inicial seria desconsiderado no planejamento de consultas.

4.3.3 Geração do plano de consulta

O plano de consulta é gerado a partir dos subgrafos identificados no processo de seleção de fontes. Um exemplo de plano de consulta é ilustrado na Figura 4.3(c). Ele é composto por uma sequência de blocos, onde cada bloco representa o percurso dentro de um padrão estrela na base proprietária ou um acesso a um *endpoint*. Cada bloco tem um identificador, B_i, e o valor do primeiro elemento da sua tupla determina o tipo do bloco: *prop* para o processamento na base proprietária e *endpoint* para o processamento em uma base de terceiros. Blocos do tipo *prop* definem um percurso no grafo; já blocos do tipo *endpoint* contém todas as informações necessárias para a geração de uma consulta SPARQL para ser submetida a uma bases externa. O plano da Figura 4.3(c) é composto por 2 blocos, um de cada tipo. O formato da tupla que define o bloco do tipo é *prop* tem a forma (prop, *percursoProp*), onde *percursoProp* é o conjunto de padrões de triplas a serem processados no bloco. Cada padrão tem o formato *(tripla, direcao, filtro)*, onde a *tripla* e *filtro* correspondem ao padrão de tripla e predicado definidos na consulta, e *direcao* pode ter o valor *in* ou *out*, indicando se o percurso é do objeto para o sujeito ou do sujeito para o objeto, respectivamente.

A tupla que define o bloco do tipo *endpoint* tem a forma (endpoint, *url, padraoTriplas, varRes, varJuncao*), onde: (1) url é o endereço do *endpoint*; (2) *padraoTriplas* contém os padrões de triplas a serem processados pela base externa, juntamente com os filtros aplicados sobre eles; (3) *varRes* são as variáveis no resultado da consulta obtidas da base externa; e (4) *varJuncao* são as variáveis que dão sequência ao percurso da base proprietária para a base externa. O plano de consulta é finalizado com um bloco de projeção B_p, que contém as variáveis no resultado da consulta ($result_q$).

O plano de execução da consulta define uma ordem para o percurso sobre a grafo RDF. Para gerar esta ordem, o FeSHyD utiliza uma heurística simples: o percurso sobre a base proprietária deve ser o maior possível antes que as bases de terceiros sejam acessadas. A intuição é que a latência nas conexões externas pode afetar o desempenho da consulta. Além disso, o percurso sobre a base proprietária pode filtrar os valores de interesse a serem obtidos da base externa, minimizando o volume de dados a ser transferido. A escolha do ponto inicial de exploração do grafo na base proprietária também pode influenciar no desempenho da consulta. Porém, este não é o foco desta dissertação.

A geração do plano de execução da consulta é realizada pelo Algoritmo 3. A entrada do Algoritmo 3 é o grafo de estrutura (G_s) e uma consulta $q = (G_q, result_q)$. O resultado do algoritmo é um conjunto de planos de consulta $planSet$. O algoritmo inicia com a chamada à função *findMappings* (Linha 2), que retorna um conjunto de mapeamentos *map* que definem homomorfismos de G_q para subgrafos em G_S. Tal algoritmo foi descrito na Seção 4.3.2. Em seguida, para cada subgrafo, o algoritmo gera um plano de consulta (Linhas 3-34), que é inserido em *planSet* (Linha 35). Utilizando o mesmo exemplo da Seção 4.3.2, considere o grafo de estrutura da Figura 4.2(b) e o grafo de consulta na Figura 4.3(b). A função *findMappings* retorna dois mapeamentos: $map_1 = \{?offer \mapsto v_6, ?valueOffer \mapsto v_7, ?vendor \mapsto v_9, ?nameVendor \mapsto v_{10}\}$ e $map_2 = \{?offer \mapsto v_6, ?valueOffer \mapsto v_7, ?vendor \mapsto v_{11}, ?nameVendor \mapsto v_{12}\}$.

Algoritmo 3: Geração do plano para uma consulta q

Entrada: $G_S, q = (G_q, result_q)$
Saída: *planSet*: conjunto de planos de consulta

```
1  planSet := {};
2  M := FindMappings(G_S, G_q);
3  for each map in M do
4      G_prop := {((?s, p, ?o), F) ∈ G_q | end(map(?s)) = prop or end(map(?o)) = prop} ;
5      if G_prop = {} then
6          seja ((?s, p, ?o), F) um elemento em G_q;
7          plan := epBlock(1, end(map(?s)), G_q, result_q, {});
8      else
9          G_ep := G_q − G_prop;
10         ?s_1 := nodo em G_prop de maior grau;
11         plan := [ ]; bNum := 1; varList := [?s_1]; varInd := 1;;
12         for each endpoint ep_i do varToEp[i] := {};
13         while G_prop ≠ {} or G_ep ≠ {} do
14             if varInd ≤ |varList| then
15                 suj := varList[ind];
16                 block := createPropBlock(suj, map, G_prop, varList, varToEp);
17                 if block ≠ [] then
18                     plan := plan + propBlock(bNum, block);
19                     bNum := bNum + 1 ;
20                 end
21                 varInd := varInd + 1;
22             else
23                 for each endpoint ep_i do
24                     while varToEp[i] ≠ {} do
25                         ini := uma variável em varToEp[i];
26                         block := createEpBlock(ini, map, G_ep, usedVars, outVars);
27                         if block ≠ [] then
28                             vJoinIn := varList ∩ usedVars;
29                             vJoinOut := outVars − varList;
30                             vRes := (usedVars ∩ result_q) ∪ vJoinOut;
31                             plan := plan+
32                             epBlock(bNum, url_epi, block, vRes, vJoinIn);
33                             insere vJoinOut em varList;
34                             bNum := bNum + 1;
35                         end
36                         varToEp[i] := varToEp[i] − usedVars;
37                     end
38                 end
39             end
40         end
41     end
42     insert plan in planSet ;
43 end
44 return planSet;
```

Na geração de um plano, a partir de cada mapeamento é possível determinar as bases de dados relevantes como descrito na Seção 4.3.2 utilizando as funções `end` e `map` apresentadas no início da seção. As triplas da base proprietária são armazenadas em G_{prop} e as triplas referentes as bases de terceiros são armazenadas em G_{ep}. Considerando o mapeamento map_1 para o ponto inicial $v6$, por exemplo, teremos: $end(map_1(?offer)) = end(v_6) =$ `prop` e $end(map_1(?vendor)) = end(v_9) = url_{Endpoint1}$. Portanto, para este mapeamento a etapa de seleção de fontes definida para o exemplo seria: $G_{prop} = \{(?offer, value, ?valueOffer), \{?valueOffer \leq 58000\}), (?offer, is\_sold, ?vendor), \{\})\}$ e $G_{ep} = \{((?vendor, name, ?nameVendor), \{\})\}$. Para o mapeamento map_2, a seleção de fontes retornaria o mesmo conjunto de mapeamentos, devido aos `Endpoint 1` e `Endpoint 2` apresentarem uma única tripla de dados com o mesmo predicado. Para facilitar a discussão, todos os exemplos a partir deste ponto são baseados em map_1.

Caso a parte da consulta sobre a base proprietária G_{prop} esteja vazia, a consulta deve ser completamente executada sobre uma base de terceiros. Assim, o plano de consulta conterá um

único bloco contendo todas as triplas da consulta, indicado pela função `epBlock`. Esta função recebe como entrada todos os componentes do bloco do tipo *endpoint*: o número do bloco, a url do *endpoint*, os padrões de triplas (G_q), as variáveis do resultado e as variáveis de junção (Linhas 5-7). Tais informações são utilizadas para selecionar o bloco adicionado no plano de consulta. Tais informações também são incluídas no plano. De maneira semelhante, a função `propBlock` inclui um bloco referente a base proprietária.

Caso G_{prop} não seja vazio, o plano da consulta conterá blocos do tipo *prop* (gerados nas Linhas 14-20) e possivelmente blocos do tipo *endpoint* (Linhas 22-34). O plano sempre inicia com um percurso sobre a base proprietária. É escolhido o ponto inicial de exploração do grafo ($?s_1$). No algoritmo, é escolhido o nodo em G_{prop} de maior grau (Linha 10), mas outras estratégias podem ser adotadas. Embora a estratégia possa afetar o desempenho da consulta, ela não interfere na geração do plano. Os vértices em G_{prop} já visitados são guardados na variável $varList$, que inicia com $?s_1$. O índice $varInd$ indica a variável em $varList$ para a qual um novo bloco do plano de consulta será gerado. Ou seja, todas as variáveis em $varList$ anteriores à $varInd$ já foram consideradas na geração do plano. No algoritmo, todas as variáveis em $varList$ são processadas antes de gerar blocos do tipo *endpoint* (Linha 14). Isso garante que o plano gerado percorre o maior grafo conexo em G_{prop} que contém $?s_1$. Para cada variável, o bloco é gerado pela função *createPropBlock*, apresentada no Algoritmo 4. Além da geração do bloco, a função remove de G_{prop} as triplas processadas, insere em $varList$ as variáveis visitadas em G_{prop}, e insere em $varToEp[i]$ as variáveis que fazem ligações com a base externa $endpoint_i$.

Retornando ao exemplo, a variável *?offer* é escolhida como ponto inicial de exploração. Após a execução da função *createPropBlock*, o bloco B_1 da Figura 4.3(c) é gerado, G_{prop} estará vazio, $varList$ = [?*offer*, ?*valueOffer*], $varInd = 2$ e $varToEp[1]$ = {?*vendor*}. A função será chamada novamente para a variável *?valueOffer*, mas como *block* retorna vazio, o plano não é alterado. Como todos as variáveis em $varList$ foram processadas, o algoritmo passa a processar variáveis localizadas em G_{ep}.

Para cada *endpoint* ep_i, os pontos de entrada na base estão armazenadas na variável *varToEp[i]*. Como um bloco do tipo *endpoint* não define um percurso na base, mas apenas identifica quais os padrões de triplas que devem ser processadas externamente, o algoritmo inicia obtendo aleatoriamente uma variável em *varToEp[i]*, chamada de *ini* (Linha 24). A partir desta variável a função *createEpBlock*, apresentada no Algoritmo 5, gera um bloco para ser inserido no plano. Ao contrário da função *createPropBlock*, que limita-se a visitar os vizinhos de um nodo, esta função obtém o maior grafo conexo em G_{ep} que contém *ini*. Assim, o percurso neste grafo na base externa será requisitado em uma única consulta SPARQL durante o processamento da consulta. Além de gerar o bloco, a função *createEpBlock* remove de G_{ep} os padrões de triplas processados e retorna dois conjuntos de variáveis: as variáveis visitadas em ep_i (*usedVars*) e as variáveis na base proprietária ligadas à ep_i (*outVars*). A variável *usedVars* é utilizada para determinar as variáveis de junção do resultado da consulta SPARQL com os resultados dos blocos anteriores na consulta (Linha 27), bem como identificar quais destas variáveis compõem o resultado da consulta q (Linha 29). Além disso, é possível que nem todos os valores necessários de ep_i formem um único grafo conexo. Dessa forma, após gerar um bloco a partir de uma variável *ini*, as variáveis em *usedVars* são removidas de $varToEp[i]$ (Linha 34). Se ainda restarem variáveis, novos blocos podem ser gerados a partir delas (Linha 25).

A execução da função *createEpBlock* no exemplo apresentado nesta seção, $varToEp[1]$ assumiria o valor {?*vendor*} após a análise sobre a base proprietária. A partir desse ponto inicial seriam acrescentados os padrões de triplas referentes ao `endpoint1`. Após a execução G_{ep} = {}, *usedVars* = {?*vendor*, ?*nameVendor*} e, como não há reentrada na base proprietária, *outVars* = {}.

Algoritmo 4: Geração de bloco para percurso na base proprietária

```
1  Function createPropBlock(node, map, G_prop, varList, varToEp)
2      block := [];
3      for each t = ((?s, p, ?o), F) in G_prop do
4          if ?s = node or ?o = node then
5              if ?s = node then
6                  block := block + ((?s, p, ?o), out, F));
7                  adjNode :=?o;
8              else
9                  block := block + ((?s, p, ?o), in, F);
10                 adjNode :=?s;
11             end
12             remove t from G_prop;
13             baseAdjNode := end(map(adjNode));
14             if baseAdjNode = prop then
15                 insertDistinct(varList, adjNode);
16             else
17                 insert adjNode in varToEp[baseAdjNode];
18             end
19         end
20     end
21     return block;
```

Já a variável *outVars* é importante quando o grafo em G_{prop} é desconexo. Ou seja, embora é assumido que o grafo da consulta G_q seja conexo, a ligação entre eles pode percorrer arestas em G_{ep}. Assim, *outVars* contém as variáveis que fazem a "re-entrada" na base proprietária a partir de uma base de externa. Portanto, estas variáveis devem ser retornadas no resultado da consulta (Linhas 28-29) e são inseridas na variável $varList$ para que o algoritmo prossiga com a geração de novos blocos do tipo *prop* (Linha 32).

As funções que geram blocos do tipo *prop* e *endpoint* são apresentadas nos Algoritmos 4 e 5, respectivamente. A função *createPropBlock* determina a direção do percurso a partir de um vértice *node* em G_{prop}, que pode ser *out*, quando *node* é o sujeito da tripla (Linha 6), ou *in*, quando *node* é o objeto da tripla (Linha 9). O padrão de tripla então é removido de G_{prop} (Linha 12). Considerando a execução sobre o padrão de tripla *t*=(?*offer*, *value*, ?*valueOffer*), o *node* é o sujeito ?*offer* e a direção para esse *node* é *out*. Após, a variável adjunta do padrão de tripla analisado é adicionada em $varList$, caso a base adjunta for a base proprietária, ou em $varToEp[i]$, caso for um *endpoint* (Linhas 13 a 18). Para o padrão de tripla t, a variável adjunta ao *node* é o objeto ?*valueOffer*, que está armazenada na base proprietária. Logo essa variável é armazenado em $varList$.

Já a função *createEpBlock*, cria um bloco com todos vértices alcançáveis a partir do vértice *node* juntamente com os seus caminhos. A variável *usedVars* é inicializada com *node* (Linha 2). Os padrões de tripla são adicionados no bloco (Linha 8) e eles são removidos de G_{end} (Linha 9). Durante a análise dos padrões de triplas, todos os vizinhos de variáveis em *usedVars* são inseridos ou em *usedVars* (Linha 12) ou em *outVars* (Linha 14). Considera o padrão de tripla *q*=(?*vendor*, *name*, ?*nameVendor*) executado em *createEpBlock*. O *node* para *q* é a variável ?*vendor*. Como a variável adjunta ?*nameVendor* ao *node* esta contida no *endpoint* então ela é adiciona a *usedVars*. O processo termina quando atinge um ponto fixo, ou seja, quando o bloco para determinado *endpoint* é criado (Linhas 4-15).

Sendo assim, o plano P_1 da Figura 4.3(c) é gerado a partir do grafo homomórfico representado por map_1. A partir de map_2 é gerado um plano similar, chamado de P_2, com acesso ao *Endpoint2*.

Algoritmo 5: Geração de bloco para percurso em uma base de terceiros

```
1  Function createEpBlock(node, map, G_ep, usedVars, outVars)
2      block := []; outVars := {}; usedVars := {node};
3      epUrl := end(map(node));
4      repeat
5          blockAnt := block;
6          for each t = ((?s, p, ?o), F) in G_ep do
7              if ?s ∈ usedVars or ?o ∈ usedVars then
8                  block := block + t;
9                  remove t from G_ep;
10                 if ?s ∈ usedVars then adjNode :=?o else adjNode :=?s;
11                 if end(map(adjNode)) = epUrl then
12                     insere adjNode em usedVars
13                 else
14                     insere adjNode em outVars
15                 end
16             end
17         end
18     until block = blockAnt;
19     return block;
```

4.4 PROCESSADOR DO PLANO DE EXECUÇÃO DA CONSULTA

O conjunto de planos gerados são enviados para o módulo de execução do moderador. Ele analisa cada plano e para cada um, determina se ele envolve apenas bases de terceiros. Neste caso, as requisições são feitas diretamente pelo moderador. Isso ocorre quando o primeiro bloco da consulta é do tipo *endpoint*. Caso contrário, a consulta envolve dados da base proprietária. Assim, o moderador envia o plano de consulta para todos os servidores que compõem a base proprietária, que iniciam sua execução em paralelo com os dados armazenados localmente. O diferencial do FeSHyD é que toda a comunicação entre os servidores da base proprietária, bem como com as bases de terceiros é realizada sem a interferência do moderador. Isso evita que o moderador seja um gargalo no processamento das consultas.

É importante observar que a base de dados proprietária é fragmentada seguindo o padrão estrela e que os blocos do plano de consulta são gerados seguindo esta mesma estrutura. Esta coincidência entre as estruturas permite que apenas no início do processamento de cada bloco seja necessário verificar se há necessidade de comunicação entre servidores. Isso é possível porque o modelo de fragmentação garante que todos os vértices do fragmento estão alocados em um mesmo servidor.

A comunicação entre os servidores da base proprietária é viabilizada pelo armazenamento de índices e metadados em cada servidor. Assim, no início de cada bloco do tipo *prop*, o módulo de execução verifica se ele acessa dados em outro servidor. Em caso afirmativo, os resultados intermediários são enviados para este servidor, que continua o processamento do plano. Cada resultado intermediário da consulta é representado por um mapeamento das variáveis para vértices da base de dados. Para exemplificar, considere o plano de consulta P_1 da Figura 4.3(c) e a base de dados da Figura 4.2(a). No servidor W, a execução do bloco B_1, gera como resultado os mapeamentos $r_1 = \{?\mathit{offer}? \mapsto b_2, ?\mathit{valueOffer} \mapsto 57000, ?\mathit{vendor} \mapsto b_6\}$ e $r_2 = \{?\mathit{offer}? \mapsto b_3, ?\mathit{valueOffer} \mapsto 56000, ?\mathit{vendor} \mapsto b_5\}$. Os dois mapeamentos são mantidos no resultado porque ambos satisfazem o filtro $?\mathit{valueOffer} \leq 58000$. Neste exemplo são utilizados identificadores para representar vértices em bases de terceiros (b_5 e b_6). Contudo, eles representam a url do recurso armazenado na base externa. A execução do plano P_2 no servidor W gera o mesmo resultado ($\{r_1, r_2\}$). No servidor X, nenhum resultado é gerado, uma vez que não existe nenhum vértice que contenha os predicados *value* e *is_sold*. Assim, o processamento da consulta prossegue apenas no servidor W.

Como o bloco seguinte é do tipo *endpoint*, o módulo de execução envia os resultados intermediários e o bloco para o módulo de geração de subconsulta e junção dos resultados. O processo é apresentado no Algoritmo 6. Antes deste processo iniciar, os resultados intermediários enviados pela base proprietária passam por uma etapa de preparação dos dados. Ela envolve o envio dos resultados intermediário para o módulo de geração de subconsultas, assim como o armazenamento deles em um arquivo ($Results_{prop}$).

A consulta SPARQL é gerada da seguinte forma. Dados o bloco B = *(endpoint, url, padraoTriplas, varRes, varJuncao)*, a cláusula select contém todas as variáveis em *varRes* e *varJuncao* (Linhas 5-10); a cláusula where contém as triplas em *padraoTriplas* (Linha 9); e a cláusula filter contém os filtros em *padraoTriplas* (Linha 12), além de uma disjunção de cláusulas conjuntivas que envolvem variáveis em *varJuncao* (Linhas 14 a 19). Mais detalhadamente, cada resultado intermediário com variáveis em *varJuncao* associados a recursos com a url do *endpoint*, gera uma conjunção na cláusula filter. Dando sequência à consulta utilizada como exemplo, *varJuncao* contém uma única variável $?vendor$. O resultado intermediário gerado pela bloco B_1 é composto pelos mapeamentos $\{r_1, r_2\}$. Na execução do plano P_1, r_2 é descartado porque P_1 tem $url_{Endpoint1}$ e $r_2(?vendor) = b_6$ e b_6 não tem $url_{Endpoint1}$ como seu prefixo. Assim, P_1 gera, a partir de r_1, a expressão $(?vendor = b_5)$. Similarmente, P_2 descarta o resultado r_1 e gera a expressão $(?vendor = b_6)$. Observe que se os dois recursos pertencessem ao *Endpoint1*, o plano P_1 geraria a expressão $((?vendor = b_5)or(?vendor = b_6))$. O processo de geração dos filtros envolve também a eliminação de cláusulas conjuntivas repetidas. $uVetor$ contém os resultados já usados.

Como resultado final, o plano P_1 gera a consulta "SELECT *?vendor*, *?nameVendor* WHERE *?vendor name ?nameVendor* FILTER $(?vendor = b_5)$)", que é submetida ao *Endpoint1*. O plano P_2 gera uma consulta similar, que é submetida ao *Endpoint2* (Linha 20).

Os resultados da consulta são recebidos pelo servidor que submeteu a consulta, que realiza então uma equijunção com os resultados intermediários anteriores sobre os atributos em *varJuncao* (Linhas 21 a 29). Assim, os resultados intermediários são estendidos com todas as variáveis encontrados nos *endpoints*. Na consulta do exemplo, $res = \{?nameVendor\}$. Assim, o resultado da equijunção, armazenado em $results_q$, para P_1 é a extensão de r_1 com o mapeamento $(?nameVendor \mapsto "Paul")$ e em P_2, o resultado é a extensão de r_2 com $(?nameVendor \mapsto "Peter")$. O plano de consulta sempre termina com um bloco de projeção. Assim, apenas as variáveis presentes neste bloco são mantidos nos resultados da consulta, que são encaminhados para o moderador. O moderador apenas faz a união dos resultados recebidos por todos os servidores para gerar o resultado final.

O diferencial do FeSHyD é que, ao contrário da arquitetura tradicional de sistemas federados, nos quais a junção entre dados de bases distintas é realizada de maneira centralizada no moderador, ele trata a base proprietária como uma caixa-branca. Isso permite que o processamento desta operação, que em geral é custosa, seja realizada em paralelo nos servidores que compõem a base proprietária. O estudo experimental apresentado na próxima seção analisa o efeito desta estratégia no tempo de execução das consultas.

4.5 DISCUSSÃO

Neste capítulo foi apresentado o sistema FeSHyD que realiza o processamento de uma busca SPARQL distribuída entre bases de dados RDF híbridas, envolvendo uma base distribuída proprietária (tratado como caixa-branca) e *endpoints* SPARQL (tratados como caixa-preta). Conforme descrito no Capítulo 2, a maioria dos sistemas processa junção entre dados de bases diferentes de maneira centralizada no moderador. Entretanto, a exploração da base proprietária

Algoritmo 6: Geração da consulta e junção dos resultados

```
   Entrada: B_e,Results_prop
   Saída: results_q
 1 results_q := {};
 2 consulta := {};
 3 uVetor := {};
 4 let B_e = (endpoint, url, padraoTriplas,varRes, varJuncao);
 5 for each ?v ∈ varRes do
 6 |   concat ?v in (SELECT, consulta);
 7 end
 8 for each v ∈ varJuncao do
 9 |   concat v in (SELECT, consulta);
10 end
11 for each t ∈ padraoTriplas do
12 |   concat t in (WHERE, consulta);
13 end
14 for each f ∈ padraoTriplas do
15 |   concat f in (FILTER, consulta, AND);
16 end
17 for each rprop ∈ results_prop do
18 |   if rprop ∉ uVetor then
19 |   |   concat rprop in (FILTER, consulta, OR, varJuncao);
20 |   |   insert rprop in uVetor;
21 |   end
22 end
23 Results_end = requisitaEndpoint (consulta, url);
24 for each t1 ∈ results_prop do
25 |   for each t2 ∈ results_end do
26 |   |   if for all att in varJuncao: t1.att = t2.att then
27 |   |   |   insert t1 ⋃ t2 in results_q;
28 |   |   end
29 |   end
30 end
31 return results_q
```

como caixa-branca viabilizou o processamento em paralelo dessas junções em servidores de dados distintos, descentralizando o processamento comumente feito pelo moderador.

A extensão na base proprietária pelo sistema proposto expandiu a capacidade de processamento do sistema inicial, que antes trabalhava somente com dados proprietários. A extensão do grafo de estrutura da base proprietária viabilizou de maneira simples a execução de consultas SPARQL que integrem dados de diferentes bases e contextos. Por fim, a capacidade de requisitar dados a bases externas (*endpoints*) estendeu a capacidade de resposta dos servidores de dados, que antes contavam somente com suas bases locais.

5 IMPLEMENTAÇÃO E EXPERIMENTOS

Neste capítulo são apresentados os detalhes de implementação do sistema FeSHyD, assim como os experimentos realizados sobre a abordagem proposta nesta dissertação. A Seção 5.1 descreve a implementação do FeSHyD. A Seção 5.2 descreve a abordagem usada como *Baseline* nos experimentos. A Seção 5.3 apresenta todos os elementos envolvidos nos experimentos, assim como os resultados da comparação entre o sistema FeSHyD e a abordagem *baseline*.

5.1 IMPLEMENTAÇÃO

A implementação do FeSHyD baseou-se no sistema gerenciador de base de dados proprietária distribuída *PAbS* (*Pattern Allocation-based System* (Penteado et al., 2019), que é descrito nesta seção. Também é apresentado como o FeSHyD estendeu o processamento do *PAbS*, possibilitando o processamento de uma consultas sobre uma base de dados distribuída e híbrida.

5.1.1 *PAbS*

O sistema *PAbS* adota uma arquitetura *Mestre-Escravo shared-nothing*, conforme mostra a Figura 5.1(a) *PAbS* (Penteado et al., 2019). A arquitetura do sistema adota o paralelismo entre servidores, no qual cada escravo executa requisições em paralelo com outros escravos, sem a exigência de sincronismo entre eles. Para a aplicação das otimizações propostas nesta dissertação este sistema é acessado como uma caixa-branca.

No *PAbS* os dados da base RDF são fragmentados e distribuídos entre os escravos de um *cluster*. Um exemplo de uma base de dados distribuída é apresentado na Figura 5.1(b). Padrões de alocação (PAs) definem padrões estruturais usados para particionar uma base RDF. Fragmentos são instâncias de PAs que seguem sua estrutura. Por exemplo, os fragmentos *PaOffer4* e *PaOffer5* da Figura 5.1(b) são instâncias do padrão de alocação *PAOffer* da Figura 5.2(a). Os fragmentos são as unidades de comunicação do sistema e garantem co-alocação no armazenamento. Ou seja, nodos em um grafo RDF G_D que pertencem ao mesmo fragmento são alocados em um mesmo servidor e qualquer transmissão de dados entre os servidores envolve no mínimo um fragmento. A Figura 5.2(a) apresenta o grafo de estrutura G_E da base G_D da Figura 5.1(b). G_E possui três PAs que estão representados por linhas tracejadas e nomeados de acordo com a classe que os compõem. Os PAs adotam o padrão estrutural do tipo estrela. Vale destacar que arestas de G_E que conectam nodos de um mesmo PA garantem que os nodos correspondentes da base G_D são alocados em um mesmo servidor. Arestas que conectam nodos de PAs distintos não fornecem esta garantia.

Durante o inserção de dados nos escravos, índices são criados para dar suporte às funcionalidades do sistema. Além dos índices, cada escravo armazena localmente os metadados necessários para o seu funcionamento (Metadados$_E$). O mestre armazena em seu repositório o grafo de estrutura de G$_D$, dando suporte ao *módulo de planejamento* (Metadados$_M$).

O *PAbS* dá suporte a consultas SPARQL que contém a cláusula SELECT, a cláusula AND (.), a cláusula FILTER e permite variáveis somente em sujeitos e objetos nos padrões de triplas. Para o processamento da consulta no *PAbS*, os padrões de triplas na consulta SPARQL contém um predicado fixo com uma IRI, e sujeitos e objetos devem ser variáveis. As variáveis

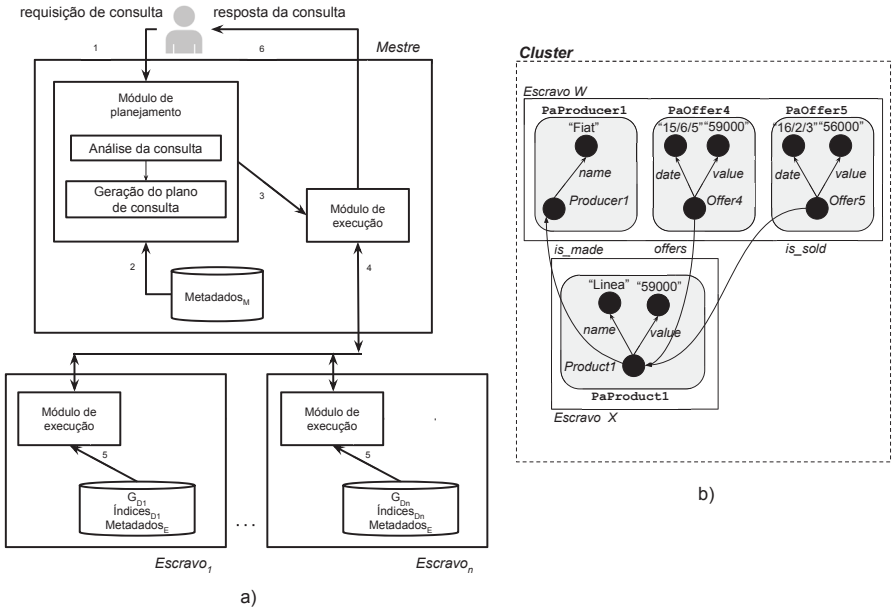


Figura 5.1: Arquitetura do sistema(a); Base RDF distribuída(b) (Penteado et al., 2019)

são mapeadas usando FILTER, ou seja, podem ser atribuídos valores às variáveis por meio dessa cláusula. A consulta da Figura 5.2 (b) ilustra uma consulta deste tipo e a Figura 5.2 (c) ilustra o grafo representando esta consulta. O *PAbS* considera grafos de consultas conexos.

Voltando à arquitetura do sistema, dada uma requisição de consulta (1), o servidor mestre elabora o seu plano de execução (2) e dispara a execução da consulta (3) enviando o plano da consulta para todos os escravos (4). A partir deste ponto, cada escravo inicia a execução do plano em paralelo usando o seu repositório local, trocando mensagens no *cluster* conforme necessário, escolhendo uma estratégia de comunicação (5). Ao final, o servidor mestre recebe os resultados gerados pelos escravos (4), retornando o resultado final da consulta ao usuário (7). O processamento está dividido em duas partes, o planejamento (*módulo de planejamento*) e a execução de consultas (*módulo de execução*).

O planejamento baseia-se nas estruturas dos fragmentos definidos pelos PAs de G_E a fim de minimizar a comunicação entre servidores durante o processamento de consultas. Como todos os nodos de um fragmento estão armazenados em um mesmo servidor, o plano de consulta gerado percorre as arestas internas do fragmento antes de passar para um novo fragmento. O subgrafo destacado na Figura 5.2 a) representa o subgrafo homomórfico ao grafo G_Q da consulta exemplo. A Figura 5.2(d) mostra o plano gerado para a consulta Q da Figura 5.2(b). Cada passo s_i do plano é uma tupla (a, dir, pat, id, f), onde: (**1**) a é um padrão de tripla de Q, (**2**) $dir \in \{in, out\}$; se $dir = out$ a exploração é do sujeito para o objeto. Caso contrário, a exploração é do objeto para o sujeito; (**3**) pat é o PA que contém a; (**4**) id é um identificador único associado ao PA. Todos os passos com um mesmo id compõem um bloco de execução dentro de um mesmo fragmento. (**5**) f é um conjunto de filtros definidos em a.

Para o processamento, cada escravo do *cluster* inicia o plano de execução em paralelo. Considere o plano da Figura 5.2(d). O primeiro passo do plano é s_1: *((?product, name, ?nameProduct), out, PaProduct, 1, {})*. Ele define que a exploração de uma tripla com predicado *name* em fragmentos de *PaProduct* deve ser realizada do sujeito para o objeto (direção *out*). A execução deste plano no escravo *X* da Figura 5.1(b) utiliza índices para obter o nodo *Product1* e associá-lo à variável *?product*. Ele passa a ser um ponto inicial de exploração. A partir daí, cada passo do plano adiciona novas associações de variáveis a nodos, caminhando no grafo RDF. Se o

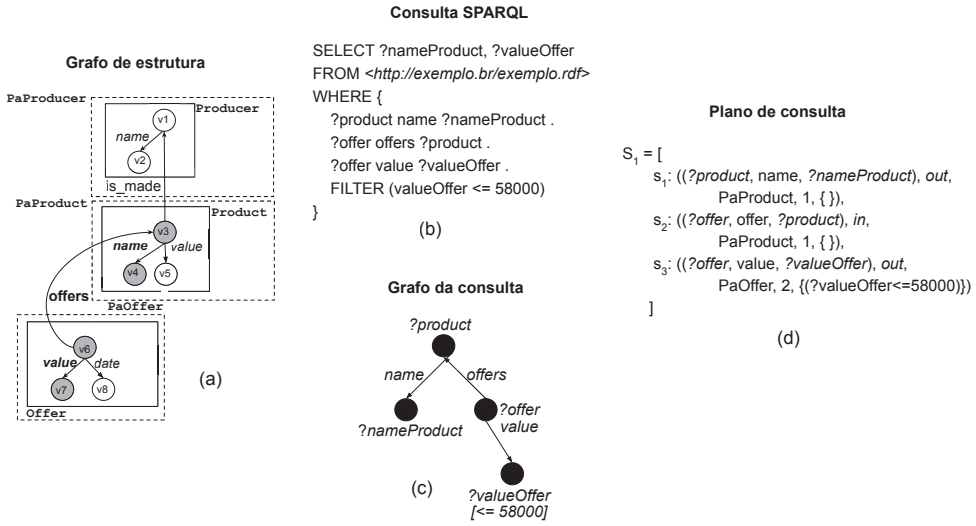


Figura 5.2: Grafo de estrutura (a); Consulta SPARQL (b); Grafo da consulta (c); Plano de consulta (d) (Penteado et al., 2019)

padrão de tripla não é encontrado ou se a nova associação não satisfaz os filtros, ele é removido do conjunto resultado.

Continuando o exemplo, o escravo X processa o passo s_2, gerando o resultado parcial $\{(?product \mapsto Product1, ?nameProduct \mapsto \text{“}Linea\text{”}, ?offer \mapsto \{Offer4, Offer5\})\}$. O processamento de s_3 pode continuar ou não em X uma vez que este passo envolve um PA distinto do passo anterior. No exemplo, as ofertas estão armazenadas no servidor W. Logo, X se comunica com W continuando a execução da consulta. Neste momento, ou X requisita os fragmentos das ofertas para W e continua o processamento localmente (usando a estratégia *get-frag*), ou o mapeamento gerado é enviado para W continuar a execução do plano (usando a estratégia *send-result*). A escolha da estratégia é feita por meio de funções de custo e depende do número de mensagens e do volume de dados a ser transmitido entre os escravos. O objetivo é minimizar o custo de comunicação entre os escravos. Por fim, os resultados parciais gerados para *Offer4* são descartados em s_3, uma vez que a valor da oferta não satisfaz o filtro $s_3.f$. O resultado da consulta é somente *{(nameProduct: “Linea"; valueOffer: 56000)}*. Mais detalhes sobre o processamento e estratégias de comunicação podem ser encontrados em (Penteado et al., 2016).

O *PAbS* foi implementado em linguagem JAVA adotando o protocolo de comunicação TCP/IP para a troca de mensagens entre os servidores do seu *cluster*. O sistema usa o Berkeley DB [1] como repositório para o armazenamento de dados.

5.1.2 Sistema FeSHyD

O FeSHyD estendeu a implementação do sistema *PAbS* com o processamento da consulta SPARQL sobre bases de dados de terceiros acessadas como caixas-pretas através de *endpoints* SPARQL. Logo, as fases de planejamento e execução da consulta do sistema *PAbS* foram adaptadas para elaborar e executar um plano de consulta sobre uma base de dados distribuída e híbrida.

A Figura 4.2(a) mostra um exemplo de uma base distribuída e híbrida. Nela é possível visualizar que a base de dados proprietária do *PAbS*, ilustrada na Figura 5.1(b), é estendida com

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

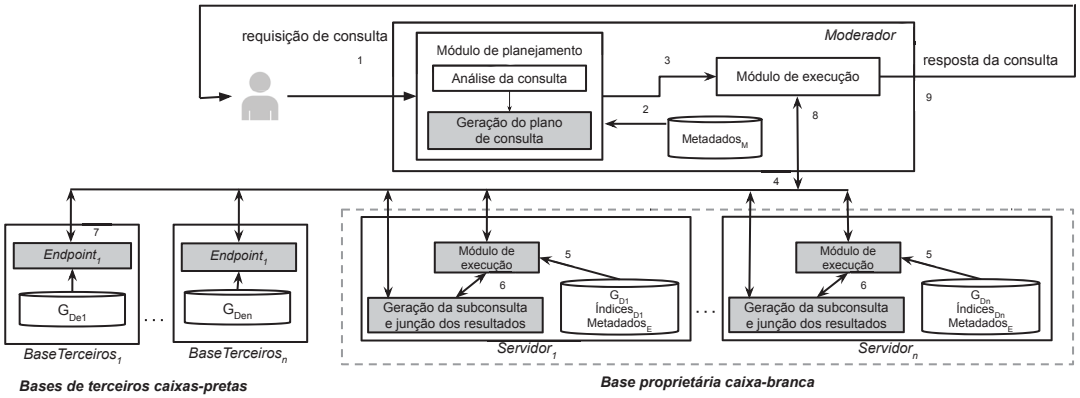


Figura 5.3: Módulos da arquitetura do FeSHyD modificados

três *endpoints* SPARQL. Os `Endpoint1` e `Endpoint2` contém dados sobre vendedores e o `Endpoint3` contém dados sobre um produto semelhante à `Product1`. Esse tipo de extensão tornou necessária a integração do grafo de estrutura do *PAbS* com os grafos de estrutura dos *endpoints*. É possível observar esta integração na comparação entre a Figura 5.2(a), que ilustra o grafo de estrutura do *PAbS*, e a Figura 4.2(b), que apresenta o grafo de estrutura estendido. Lembrando que o padrão de alocação usado no *PAbS* é o padrão estrela, o mesmo padrão de fragmento usado no FeSHyD.

Para realizar o processamento sobre uma base distribuída e híbrida, a arquitetura de processamento de consulta do *PAbS* foi alterada. Na Figura 5.3 estão destacados em cinza os módulos que foram estendidos ou criados na arquitetura do *PAbS*. Na arquitetura do FeSHyD, o moderador do *PAbS* passar a ser considerado como o moderador do sistema. Já os servidores escravos do *PAbS* passam a ser os servidores de dados do FeSHyD.

Analisando a arquitetura do FeSHyD, o primeiro módulo alterado foi o responsável pela `Geração do plano de consulta` na fase de planejamento. Neste módulo, o grafo de estrutura estendido passou a ser utilizado na criação de um plano de consulta capaz de explorar uma base de dados distribuída e híbrida. Este plano pode ser composto por blocos contendo padrões de triplas tanto da base proprietária quanto da de terceiros. Na comparação entre as Figuras 5.2(d) e 4.3(c) é possível notar esta extensão no plano de consulta. Na Figura 5.2(d) é descrito um plano de consulta do *PAbS* contendo um bloco de execução S_1 com três padrões de triplas. Este bloco S_1 é processado somente na base proprietária. O plano estendido elaborado nesta dissertação é descrito na Figura 4.3(c). Como no *PAbS*, o bloco de execução B_1 neste plano é processado sobre a base proprietária. Diferente do plano de consulta do *PAbS*, o bloco B_2 estende o plano de consulta do FeSHyD indicando que o padrão de tripla contido nele deve ser executado sobre um *endpoint*. Além disso, o novo plano de execução deve indicar os padrões de tripla que devem ser submetidos para a base proprietária ou para a base de terceiros.

Continuando, na fase de execução do sistema FeSHyD, o `Módulo de execução` nos servidores de dados foi modificado com a finalidade de processar o plano de consulta estendido. Os blocos referentes à base proprietária são processadas de maneira idêntica ao sistema *PAbS*. Porém, quando o FeSHyD identifica que o bloco de execução deve ser processado sobre um *endpoint*, o servidor de dados informa ao `Módulo de geração das consultas e junção dos resultados` a necessidade de requisição de dados à base de terceiros. Este módulo não pertence ao *PAbS* e foi elaborado com a finalidade de: criar as subconsultas submetidas aos *endpoints*; requisitar essas subconsultas aos *endpoints* das bases de dados de terceiros e providenciar a junção dos resultados obtidos das bases de terceiros com os resultados intermediários alocados nos servidores de dados.

As extensões realizadas no *PAbS* foram feitas usando a linguagem de programação C. Na extensão, o módulo da análise de consulta apresentada na Figura 4.1 foi implementado com o analisador sintático *Bison* (Levine, 2009) e o analisador léxico *Flex* (Levine, 2009), os quais analisam a gramática da linguagem. Ambos os analisadores foram configurados para reconhecer a linguagem SPARQL. Para tal uma árvore sintática, criada usando o analisador `Bison`, foi configurada para verificar a sintaxe da gramática SPARQL. No módulo de geração da consulta da Figura 4.1, para acessar os *endpoints* SPARQL foi utilizada a biblioteca para transferência de arquivos LibCurl[2]. Por meio desta biblioteca é possível requisitar uma consulta SPARQL via requisição HTTP. Arquivos foram utilizados para o armazenamento dos resultados intermediários gerados na etapa de preparação que antecede a chamada de *endpoints*. O mesmo recurso foi usado para o armazenamento dos resultados recebidos dos *endpoints* e do resultado final das consultas.

5.2 ABORDAGEM *BASELINE*

A abordagem *baseline* usada nos experimentos foi elaborada como um sistema federado sobre bases de dados acessadas como caixas-pretas, semelhante à maioria dos sistemas apresentados no Capítulo 3. Dessa forma foi possível analisar o sistema proposto nesta dissertação, que processa consultas sobre uma base de dados distribuída e híbrida.

A abordagem *baseline* estabelecida para a comparação do sistema FeSHyD é representada na Figura 5.4 pelas setas na cor azul. Ao contrário do FeSHyD, o sistema *baseline* acessa a base proprietária como uma caixa-preta. Logo, o módulo de preparação dos dados, de criação das subconsultas para os *endpoints* e de junção dos resultados intermediários fica localizado no moderador. Este módulo foi implementado de forma similar ao apresentado no Capítulo 4.

Dois modelos de implementações foram considerados a partir da abordagem *baseline*, sendo eles:

- `MB1`: pode realizar diversas chamadas aos *endpoints* durante a execução de uma consulta. O número de chamadas depende do número de servidores de dados e o número de padrões de alocação do PAbS envolvidos na consulta. Cada recebimento de resultados intermediários pelo moderador, gera uma chamada a um ou mais *endpoints*.

- `MB2`: realiza uma única chamada a um *endpoint* durante a execução de uma consulta. Aqui o moderador espera os resultados intermediários de todos os servidores. Após receber e armazenar todos os resultados intermediários, o moderador elabora e requisita a subconsulta SPARQL aos *endpoints*, conforme necessário. Embora este modelo diminui a quantidade de requisições ao *endpoint*, cada chamada requer a sincronização entre os servidores de dados, que requer espera desses servidores na transmissão dos resultados intermediários.

.

O primeiro modelo pode se aproveitar do uso de *threads*, realizando mais de uma ação ao mesmo tempo. O moderador pode efetuar uma requisição aos *endpoints* enquanto realiza uma junção de resultados, com ambas as ações vindas de servidores diferentes.

Na implementação do modelo `MB2` foi considerado que a quantidade de resultados intermediários recebidos dos servidores é conhecida antecipadamente à execução da consulta. Dessa forma, o moderador não necessita verificar se todos os resultados intermediários foram enviados, diminuindo o tempo de execução da consulta. Entretanto, em um ambiente real de processamento de consultas SPARQL o moderador não possui esta informação prévia.

[2] `https://curl.haxx.se/libcurl/`

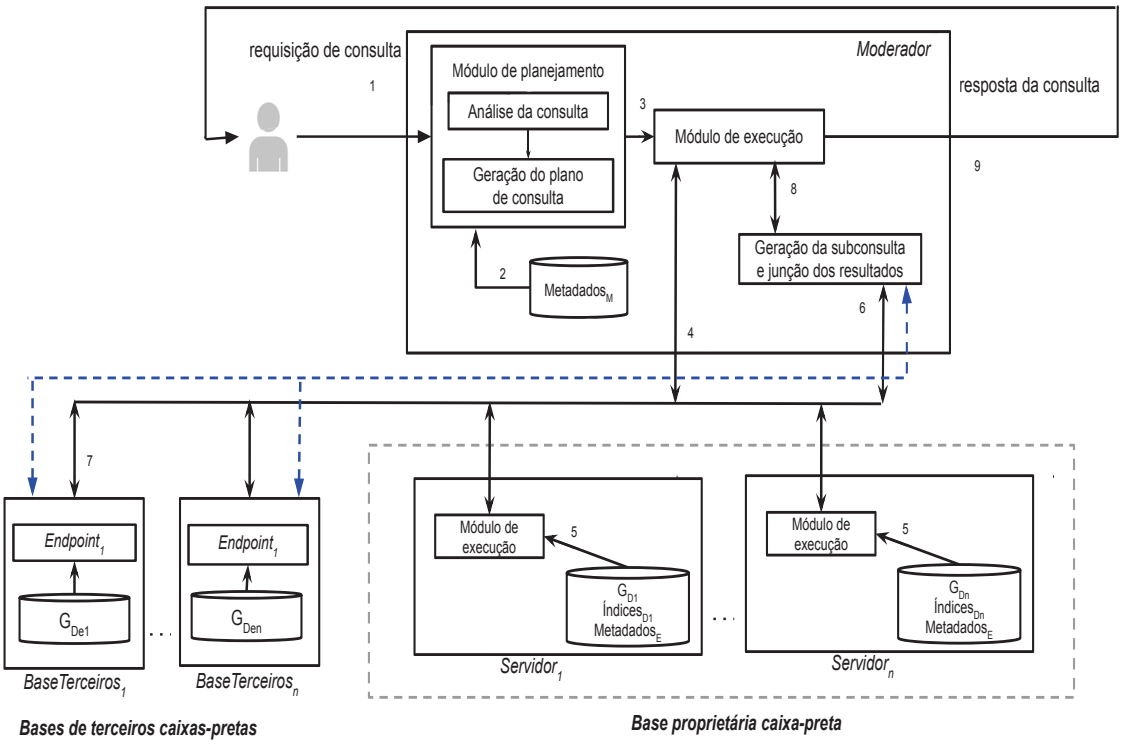


Figura 5.4: Processamento da consulta

5.3 ESTUDO EXPERIMENTAL

Para a avaliação da abordagem proposta nesta dissertação, o sistema FeSHyD é comparado com ambos os modelos de implementação da abordagem *baseline*, `MB1` e `MB2`. Os experimentos analisaram o desempenho no processamento de consultas dos sistemas FeSHyD, `MB1` e `MB2`. Além disso, o efeito da variação no número de servidores na base proprietária do FeSHyD também foi analisado.

5.3.1 Ambiente de experimentos

A base de dados usada nos experimentos contou com uma base proprietária gerada pelo *benchmark* Berlin e com uma base de dados de terceiros. O gerador de dados do Berlin usa o número de produtos como fator de escala. Foi utilizado o fator 1000, resultando em uma base de (107,17 MB). As instâncias de cada entidade foram distribuídas de maneira balanceada, ou seja, com cada servidor armazena uma quantidade similar de dados.

Foram utilizadas duas composições de *cluster* nos experimentos, uma com dois servidores de dados (`C2`) e outra com três servidores de dados (`C3`), além do moderador. O moderador utilizou uma máquina pessoal com Linux Ubuntu e possui as configurações Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz com 4 GB de RAM. O *cluster* `C2` é composto por duas máquinas que utilizam o Linux Ubuntu e processadores Intel(R) Core(TM) i5 com 4 GB de RAM cada. O *cluster* `C3` utiliza os mesmos servidores do *cluster* `C2` mais um servidor que também utiliza Linux Ubuntu e utilizam processadores Intel(R) Core(TM) i5 com 8 GB de RAM.

A base de dados de terceiros, que compõe a outra parte da base de dados híbrida, utilizada durante a realização dos experimentos foi a base Wikidata[3]. Esta base de dados contém dados sobre temas diversos e pode ser acessada via consulta SPARQL. Um *endpoint* é disponibilizado, através de uma URL, para processar as consultas.

[3]`https://www.wikidata.org/wiki/Wikidata:Main_Page`

Quatro consultas SPARQL ($Q1$–$Q4$), baseadas nos casos de uso do Berlin, foram usadas nos experimentos. As consultas foram definidas levando em consideração o número de resultados intermediários e totais que elas geram. $Q1$ recupera dados de um determinado produto e do seu produtor; $Q2$ recupera dados de todos os produtos e seus produtores; $Q3$ recupera revisões com avaliação menor que 3 e seus revisores; e $Q4$ recupera dados das ofertas e seus vendedores. Em todas as consultas é gerada uma única subconsulta SPARQL que acessa o *Wikidata* e que retorna um único resultado com 264 bytes. $Q1$, $Q2$, $Q3$ e $Q4$ retornam 1, 1000, 1432 e 19014 resultados, respectivamente. Tais consultas SPARQL são apresentadas no Apêndice A. O grafo de estrutura utilizado nos experimentos também é apresentado no Apêndice A. Cada consulta foi executada cinco vezes nos sistemas `MB1` e `MB2`. Consequentemente, cada consulta foi executada dez vezes no FeSHyD. O tempo mediano de execução das execuções foi coletado durante os experimentos. Foi usada uma execução de cada consulta para o aquecimento dos sistemas (*warmup*). A Tabela 5.1 apresenta a ficha técnica dos experimentos.

		Descrição
Sistema Operacional		Linux Ubuntu
Máquinas	Moderador	Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz com 4 GB de RAM
	servidor 1	Intel(R) Core(TM) i5 com 4 GB de RAM
	servidor 2	Intel(R) Core(TM) i5 com 4 GB de RAM
	servidor 3	Intel(R) Core(TM) i5 com 8 GB de RAM
Bases	Proprietária	benchmark Berlin
	Distribuída	Wikidata
Consultas (Resultados)		Q1(1), Q2(1000), Q3(1032), Q4(19014)

Tabela 5.1: Tabela com ficha técnica dos experimentos

5.3.2 Desempenho

Durante os experimentos foram coletados os tempos totais e parciais de execução durante o processamento das consultas $Q1$–$Q4$ pelos sistemas FeSHyD, `MB1` e `MB2`. A comparação do FeSHyd com os sistemas *baseline* foi feita inicialmente no *cluster* `C2` e, na sequência no *cluster* `C3`.

Três fatores se destacaram durante a análise, sendo eles: *i)* a *preparação* necessária pelos servidores antes da chamada aos *endpoints*, *ii)* a *junção* executada nos servidores após chamada aos *endpoints*, e *iii)* a *latência* durante a chamada aos *endpoints*. Os dois primeiros fatores estão relacionados ao paralelismo proporcionado pelo FeSHyD durante a execução de consultas. O último fator está relacionado ao tempo de resposta do *endpoint*.

Análise dos sistemas no cluster `C2`

A Tabela 5.2 apresenta os tempos de execução do FeSHyD e os sistemas *baseline* `MB1` e `MB2`. As linhas da tabela correspondem ao tempo mediano do conjunto de execuções realizado sobre cada sistema (FeSHyD, `MB1` e `MB2`) e para cada consulta SPARQL (`Q1`, `Q2`, `Q3` ou `Q4`). A coluna 1 indica a consulta. A coluna 2 mostra o sistema analisado. Nas colunas 3 a 6 são apresentados os dados gerados por um determinado servidor de dados durante o experimento (uma vez que os servidores de dados apresentaram comportamento semelhante durante os experimentos), sendo eles: o tempo de processamento na base proprietária (coluna 3); o tempo entre a chamada

ao *endpoint* até a junção realizada pelo servidor de dados dos resultados recuperados com os resultados intermediários previamente armazenados em arquivo (coluna 4); a quantidade de resultados intermediários enviados para o moderador (coluna 5); e o volume aproximado dos dados transmitidos (coluna 6). A coluna 7 mostra o tempo entre a chamada ao *endpoint* até a junção, com os dados encontrados na base proprietária, realizada pelo moderador. Vale destacar que o tempo de latência pode influenciar os tempos das colunas 4 e 7. A coluna 8 mostra a quantidade de resultados finais obtida em cada consulta. A coluna 9 destaca o tempo total de execução de cada consulta. A coluna 10 apresenta a variação percentual do tempo de execução das abordagens *baseline* em relação ao tempo de execução do sistema FeSHyD. O `"X"` na tabela significa que a abordagem não realiza tal processo. Em todas as consultas realizadas neste experimento o *endpoint* retorna apenas um resultado intermediário.

		Servidor de dados				Moderador			
Consulta	Abordagem	Tempo de processamento no base proprietária (ms)	Tempo de conexão/ resposta do *endpoint* (ms)	Nro de resultados enviados para o Moderador	Volume em bytes dos resultados enviados	Tempo de conexão/ resposta do *endpoint* (ms)	Nro de resultados finais	Tempo total da consulta (ms)	Variação Desempenho (%)
Q1	MB1	12	x	1	89	2109	1	**2302**	**+12,2%**
Q1	MB2	11	x	1	89	1946	1	**2148**	**+5,9%**
Q1	FeSHyD	7	1818	1	165	x	1	**2021**	**x**
Q2	MB1	145	x	484	75200	2337	1000	**2865**	**-1,0%**
Q2	MB2	148	x	484	75200	2020	1000	**2598**	**-11,4%**
Q2	FeSHyD	152	2174	485	89336	x	1000	**2895**	x
Q3	MB1	300	x	722	157111	2151	1432	**2910**	**15,2%**
Q3	MB2	300	x	722	157111	1493	1432	**2372**	**-3,9%**
Q3	FeSHyD	306	1787	723	180380	x	1432	**2466**	x
Q4	MB1	1434	x	9586	1345782	2161	19014	**5441**	**+30,5%**
Q4	MB2	1356	x	9586	1345782	2116	19014	**4961**	**+23,8%**
Q4	FeSHyD	1507	1811	9587	1801708	x	19014	**3777**	x

Tabela 5.2: Tabela comparativa entre os sistemas FeSHyD, `MB1` e `MB2` no `cluster C2`

Na Tabela 5.2 é possível notar que a execução paralela da operação de junção entre os servidores de dados favorece o FeSHyD quando o volume de resultados intermediários é grande. Isso ficou evidente na consulta $Q4$, onde o tempo envolvido com a junção foi de 1811 ms no FeSHyD (coluna 4) e de 2161 em `MB1` e 2116 em `MB2` (coluna 7). Ou seja, o FeSHyD obteve um ganho no desempenho de 30,5% e 23,8% em relação à `MB1` e `MB2`, respectivamente. O motivo da vantagem é a quantidade de valores envolvidos na operação de junção do FeSHyD quando comparada com os outros sistemas. Na Tabela 5.2, a quantidade de resultados intermediários processados por cada servidor de dados do FeSHyD é igual à quantidade de resultados enviados ao moderador (coluna 5). Já, a quantidade de resultados intermediários processados nos sistemas *baseline* é igual à quantidade de resultados finais gerados (coluna 8). Ou seja, enquanto a operação de junção de $Q4$ no FeSHyD envolveu 9586 resultados intermediários em paralelo nos servidores de dados, os sistemas *baseline* envolveram 19014 resultados intermediários de maneira centralizada no moderador. Em outro extremo, $Q1$ não explorou o paralelismo nos servidores da base proprietária, uma vez que ela processa um único resultado intermediário.

É possível notar que a latência prejudicou a execução de algumas consultas, uma variável sobre á qual os sistemas não tem controle. Essa questão ficou evidente em $Q1$, uma vez que os três sistemas executaram exatamente as mesmas operações durante a execução da consulta. Neste caso, o `MB1` foi prejudicado pela latência, dado que houve uma diferença de 281 ms no tempo de conexão/resposta do FeSHyD (coluna 4) para o tempo de conexão/resposta do `MB1` (coluna 7). A partir dos dados coletados nos experimentos, é possível estipular um tempo de latência médio de

200 ms aproximadamente nos experimentos. A latência também prejudicou o FeSHyD em $Q2$ e $Q3$.

Outro fator que também envolveu o paralelismo é a preparação que antecede a chamada aos *endpoints*, conforme citado na Seção 4.4. Somando os tempos parciais da Tabela 5.2, o tempo de execução na base proprietária (coluna 2) com o tempo de conexão/resposta do *endpoint* (coluna 4 ou 7), é possível notar que existe uma lacuna de tempo se comparada com o tempo total de execução. Este tempo contempla o envio dos dados ao moderador, em ambos os sistemas, e a preparação dos dados. Por exemplo, na consulta `Q4` para o sistema `MB1`, a soma dos tempos parciais é igual a 3595 ms. Comparado com o tempo total do processamento de `Q4` (5441 ms) é possível notar que esta lacuna de tempo tem 1846 ms. No sistema FeSHyD, para a mesma consulta, esta lacuna é de 459 ms. Logo, o FeSHyD obteve um ganho de desempenho de 75,1% em relação à *MB1* na preparação paralela nos servidores de dados.

O gráfico da Figura 5.5 mostra que o desempenho do FeSHyD melhora com o aumento do número de resultados intermediários processados em paralelo nos servidores de dados. A quantidade de resultados intermediários processada nas consultas $Q2$-$Q4$ no FeSHyD em cada servidor foi de aproximadamente 484, 722 e 9586. No entanto, em `MB1` e`MB2`, a quantidade total de resultados intermediários recebidos foi de 1000, 1432 e 19014, respectivamente. Ou seja, as 2 tarefas que são realizadas em paralelo pelos servidores de dados no FeSHyD são realizadas pelo moderador nos sistemas *baseline* para um volume muito maior, impactando no tempo total de processamento. A vantagem do paralelismo proporcionado pelo FeSHyD ficou evidente em `Q4`. Já, a latência influenciou na análise das outras consultas, favorecendo o FeSHyD em $Q1$ e prejudicando o sistema em $Q2$ e $Q3$. Logo, o FeSHyD tende a apresentar melhor desempenho a medida que a quantidade de resultados intermediários aumenta.

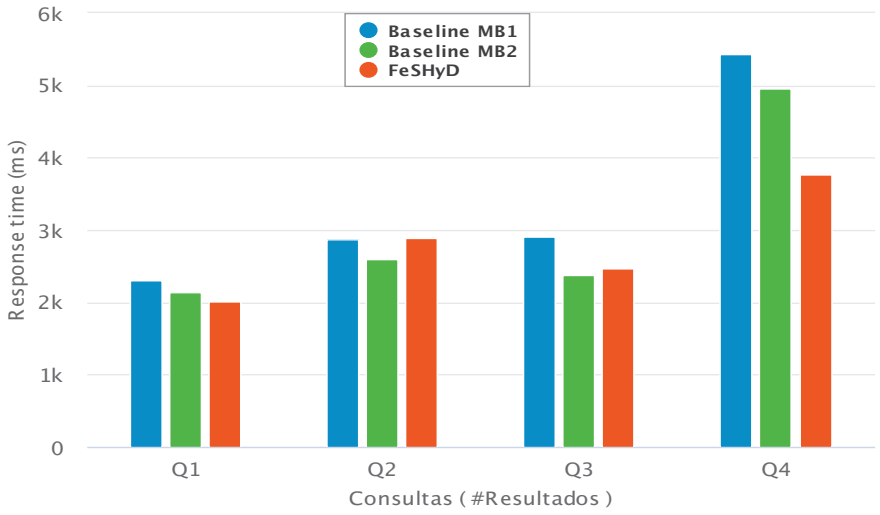


Figura 5.5: Comparação do tempo de execução no sistema FeSHyD e na *Baseline*, `MB1` e `MB2` no *cluster* `C2`

.

Análise dos sistemas no cluster `C3`

O tempo de execução das consultas no *cluster* $C3$ é apresentado na Tabela 5.3. O gráfico referente a esta tabela é apresentado na Figura 5.6. É possível notar que o FeSHyD apresenta melhor desempenho, comparado a `MB1` e `MB2`, exceto para a consulta $Q1$. Como já explicado durante a análise do *cluster* $C2$, o motivo para tal diferença é a latência na conexão aos *endpoints*, uma vez que a consulta $Q1$ processa um único resultado intermediário. Na consulta $Q1$, para o FeSHyD, do total de 2138ms (coluna 9), 1877ms referem-se ao tempo de comunicação com o *endpoint* e junção dos resultados (coluna 3); para `MB1` este tempo foi de 1893ms (coluna 7) do

total de 2052ms e para MB2 foi 1623ms (coluna 7) do total de 1847. Assim, desconsiderando a diferença nos tempos de conexão, o FeSHyD tem uma pequena desvantagem com relação aos demais sistemas. Isso se deve ao fato da junção do resultado com o único resultado da consulta ser realizado no servidor. Assim, ao contrário de MB1 e MB2, que enviam para o moderador apenas o resultado intermediário, no FeSHyD a transmissão contém também o resultado da consulta retornado pelo *endpoint*.

Assim como em $C2$, o desempenho do FeSHyD melhora com o aumento do número de resultados em $C3$. O ganho para as consultas $Q2$-$Q4$ foi de 18%, 14% e 45% com relação a MB1 e de 14%, 16% e 40% com relação a MB2. Como os dados estão distribuídos de forma balanceada entre os 3 servidores, o ganho se deve ao paralelismo nos servidores na execução das tarefas de preparação dos dados, acesso ao *endpoint* e junção dos resultados. A quantidade de resultados intermediários gerados para as consultas $Q2$-$Q4$ no FeSHyD em cada servidor foi de aproximadamente 323, 476 e 6373 (coluna 5). No entanto, em MB1 e MB2, a quantidade total de resultados intermediários recebidos foi de 1000, 1432 e 19014 (coluna 8), respectivamente. Como no *cluster* $C2$, o moderador realiza o processamento de um volume maior de dados se comparados com o volume de dados processados em cada servidor individualmente.

Pode-se observar também que quanto maior o número de resultados, maior o volume de dados transmitidos dos servidores para o moderador no FeSHyD, uma vez que cada resultado intermediário é estendido com os valores retornados pelo *endpoint*. No entanto, este custo adicional não é maior que o ganho gerado pelo paralelismo no processamento dos dados armazenados em cada servidor. Esta diferença reflete no crescimento do tempo de execução no FeSHyD com relação a quantidade de resultados. No FeSHyD, ele cresce em um ritmo menor que em MB1 e MB2, como pode ser observado na Figura 5.6.

		Servidor de dados				Moderador			
Consulta	Abordagem	Tempo de processamento na base proprietária (ms)	Tempo de conexão/ resposta do *endpoint* (ms)	Nro de resultados enviados para o Moderador	Volume em bytes dos resultados enviados	Tempo de conexão/ resposta do *endpoint* (ms)	Nro de resultados finais	Tempo total da consulta (ms)	Variação Desempenho (%)
Q1	MB1	9	x	1	89	1626	1	**1847**	**-15,7%**
Q1	MB2	8	x	1	89	1863	1	**2052**	**-4,1%**
Q1	FeSHyD	9	1877	1	165	x	1	**2138**	x
Q2	MB1	118	x	343	53669	1965	1000	**2548**	**+18,6%**
Q2	MB2	111	x	343	53669	1935	1000	**2409**	**+13,9%**
Q2	FeSHyD	117	1659	344	63700	x	1000	**2074**	x
Q3	MB1	225	x	476	104535	1945	1432	**2691**	**14,4%**
Q3	MB2	245	x	476	104535	1975	1432	**2745**	**+16,1%**
Q3	FeSHyD	201	1983	477	119820	x	1434	**2303**	x
Q4	MB1	737	x	6373	894696	2542	19014	**5036**	**+45,5%**
Q4	MB2	747	x	6373	894696	2157	19014	**4610**	**+40,4%**
Q4	FeSHyD	851	1641	6374	1197800	x	19014	**2747**	x

Tabela 5.3: Tabela comparativa entre os sistemas FeSHyD, MB1 e MB2 no cluster C3

.

Análise dos sistemas MB1 e MB2

Analisando especificamente MB1 e MB2, é possível notar uma pequena vantagem para o sistema MB2. A maior diferença aconteceu na consulta Q3 da Tabela 5.2. Entretanto, na maioria das consultas o tempo de execução foi semelhante em ambas as tabelas. Com relação ao número de requisições para as consultas $Q2$-$Q4$ em $C3$, MB1 realizou 3 requisições ao *endpoint*, enquanto MB2 realizou uma única requisição. Em $Q4$ nota-se que apesar da espera para o recebimento de 19014 resultados intermediários em MB2, a execução de 3 requisições ao *endpoint* penalizou MB1. Logo, esta diferença no desempenho dos sistemas *baseline* está relacionada ao balanceamento na

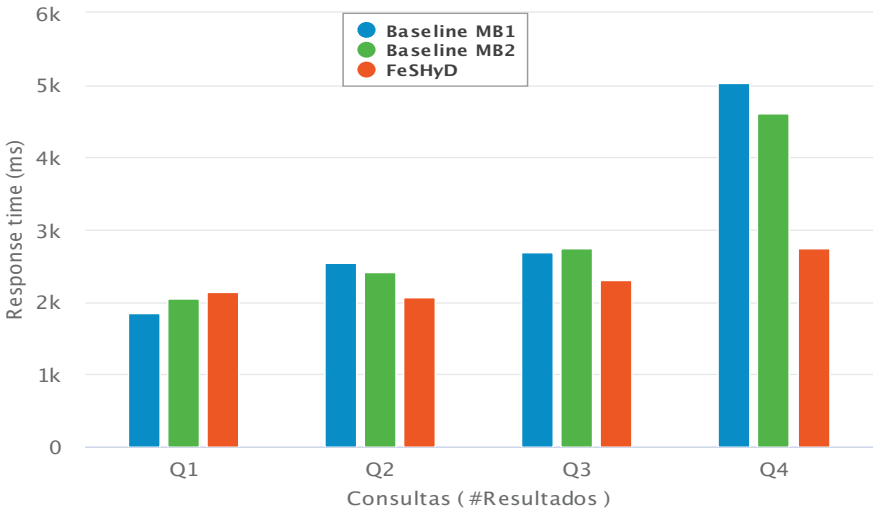


Figura 5.6: Comparação do tempo de execução no sistema FeSHyD e na *Baseline* MB1 e MB2 no *cluster* C3

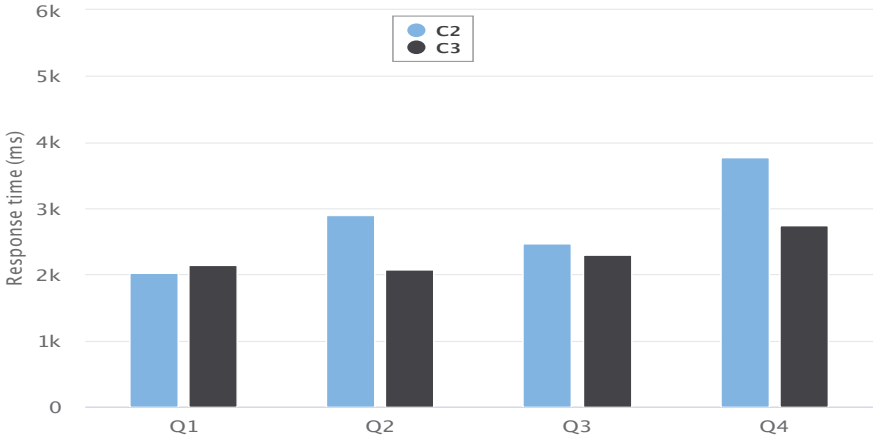


Figura 5.7: Comparação entre os *cluster* C2 e o *cluster* C3

quantidade de resultados intermediários entre realizar várias chamadas com menos resultados e processar mais resultados em uma única chamada. É provável que quando forem necessárias mais requisições aos *endpoints* durante a execução de uma consulta, o sistema MB2 tenha o desempenho afetado, devido a espera dos resultados enviados pelos servidores. Vale reforçar que a implementação de MB2 foi otimizada, em que é considerado que a quantidade de resultados intermediários é conhecido. Logo, o moderador não necessita esperar a confirmação do envio de todos os dados por parte dos servidores.

5.3.3 Análise do efeito da variação no número de servidores

O número de servidores da base proprietária influencia diretamente no paralelismo explorado pelo FeSHyD durante a execução de consultas. A Figura 5.7 mostra o tempo de execução das consultas $Q1$-$Q4$ nos *clusters* $C2$ e $C3$ pelo FeSHyD. Como esperado, $Q1$ não obteve vantagem com o aumento de servidores, uma vez que retorna um único resultado. Mas o tempo total da execução das demais consultas diminuiu de $C2$ para $C3$. Esta diminuição foi de 28%, 6% e 27% para $Q2$-$Q4$, respectivamente. $Q3$ apresentou uma redução menor devido à latência da conexão com o *endpoint*, Para uma distribuição balanceada dos dados, um número maior de servidores favorece o paralelismo e reduz o tempo de resposta das consultas, principalmente quando há um grande número de respostas.

Analisando os dados do sistema FeSHyD nas tabelas 5.2 e 5.3, é possível notar que os tempos de execução parcial e total das consultas diminuem quando a execução envolve mais servidores. O motivo é a distribuição dos dados entre os servidores. Lembrando que a mesma base de dados foi distribuída nos *clusters* `C2` e `C3`. Analisando a consulta `Q4`, por exemplo, a quantidade de resultados transmitidos do servidor de dados ao moderador é maior no *cluster* `C2` que no *cluster* `C3`. Comparando as tabelas, a quantidade varia de 9587 para 6374 resultados, respectivamente (colunas 5). Da mesma forma, o tempo de preparação e envio de dados ao moderador diminui se os dados forem divididos e processados em paralelo entre mais servidores. Voltando à consulta `Q4`, o tempo de preparo dos dados é aproximadamente de 255 ms em `C3` e de 459 ms em `C2`. Portanto, o *cluster* `C3` teve um ganho de 44,4%. Com relação ao tempo de conexão com os *endpoints* (colunas 4), para a consulta `Q4` houve uma diminuição de 9,3% de `C2` para `C3`. Apesar da influência de uma possível latência na execução da consulta, o aumento no paralelismo auxiliou na melhora do desempenho da consulta no *cluster* `C3`.

Para a consulta `Q2` houve uma melhora no desempenho dessa consulta quando executada no *cluster* `C3`. Na conexão com os *endpoints* e junção dos resultados houve uma diminuição de 23%. Na preparação dos dados o tempo foi de 569 ms para `C2` e 298 ms para `C3`. Logo, o tempo foi reduzido em 47,6% em `C3`. Mesmo com a influência da latência, o aumento de um servidor no *cluster* melhorou o desempenho do FeSHyD. Ao contrário da consulta `Q2`, a consulta `Q3` apresentou um tempo de conexão com os *endpoint* 10,9% maior em `C3` devido a influência da latência na resposta do *endpoint*. Porém, como nas consulta `Q2` e `Q4`, houve uma melhora no desempenho de `Q3` em `C3` devido a redução do tempo de preparação dos dados em 68%. Esse tempo variou de 373 ms para 119 ms entre `C2` e `C3`.

O estudo experimental realizado mostrou que o FeSHyD melhora o desempenho de consultas, principalmente quando elas envolvem uma grande quantidade de resultados. Para uma distribuição balanceada dos dados, um número maior de servidores favorece o paralelismo e reduz o tempo de resposta das consultas.

6 CONCLUSÃO

Este capítulo apresenta as considerações finais dessa dissertação na Seção 6.1. Nela são apresentados o objetivo, as contribuições e os resultados obtidos. Os trabalhos futuros são apresentados na Seção 6.2.

6.1 CONSIDERAÇÕES FINAIS

Esta dissertação apresentou o sistema denominado FeSHyD. Nesse sistema consultas SPARQL são processadas e executadas sobre uma base de dados híbrida, composta por uma base proprietária e bases de terceiros acessadas como *endpoints* SPARQL. Na base proprietária é implementada uma otimização que visa diminuir a sobrecarga de processamento sobre o moderador do FeSHyD, dividindo parte de suas responsabilidades entre os servidores de dados.

Uma das contribuições dessa dissertação é a criação de um plano de consulta sobre uma base de dados híbrida. Para identificar as bases de dados relevantes é utilizado um algoritmo de exploração de grafos na comparação do grafo de consulta e do grafo de estrutura, que descreve a base de dados híbrida. Na geração do plano de consulta, os padrões de triplas são agrupados em blocos de execução, tanto para a base proprietária quanto para as bases de terceiros, de forma que o maior subgrafo armazenado no mesmo servidor seja percorrido antes que dados de outros servidores sejam armazenados. Na ordenação dos blocos de execução, primeiro são processados os blocos referentes à base proprietária. Os resultados intermediários encontrados são mapeados nas subconsultas geradas utilizando os blocos das bases de terceiros. Desse modo, é possível diminuir a quantidade de resultados retornados pelos *endpoints*.

Outra contribuição foi a otimização proposta na base proprietária. Nessa otimização, os servidores de dados são responsáveis por elaborar e requisitar as subconsultas aos *endpoints* SPARQL, além de juntar os resultados intermediários retornados. Ou seja, os servidores de dados são responsáveis pelo processamento dos blocos das bases de terceiros. Esse processo pode ser distribuído entre servidores de dados, que executam a busca em paralelo, ao invés de concentrar esta tarefa no moderador.

As duas últimas contribuições são uma implementação do sistema FeSHyD e uma análise experimental sobre esse sistema. Na implementação, o FeSHyD utilizou como base proprietária a base distribuída gerenciada pelo sistema PAbS (Penteado et al., 2019). Essa base é particionada entre diferentes servidores de dados seguindo o padrão estrela.

Na análise experimental, o FeSHyD foi comparado com uma Baseline que simula um sistema de busca federada tradicional, ou seja, acessa todas as bases como caixas-pretas. Nos experimentos foi possível perceber que: (1) o sistema FeSHyD melhora o desempenho da execução da consulta, em até 45%, quando a quantidade de resultados intermediários é grande. O motivo é a distribuição do processamento da consulta entre os servidores de dados, enquanto que na baseline todo o processamento é centralizado no moderador; (2) quando são encontrados poucos resultados intermediários a latência de comunicação com os *endpoints* tem a maior influência no tempo final da consulta. Nesse caso o sistema FeSHyD apresentou desvantagem em parte dos experimentos; (3) com o acréscimo de um servidor o sistema FeSHyD obteve uma melhora de desempenho no tempo de execução. O motivo é o processamento de menos resultados intermediários por cada servidor de dados.

Foram aceitos três artigos relacionados a esta dissertação:

- Takiuchi, H. P. B., Penteado, Raqueline R. M. Hara, P. e Hara, C. S. (2019). Feshyd: Busca federada sobre bases de dados rdf híbridas. Em SBBD WTDBD, páginas 148–154.
- Penteado, R. R. M., Takiuchi, H. P. B. e Hara, C. S. (2019). PAbS: Um processador de consultas sparql sobre bases distribuídas. Em SBBD Demo, páginas 18–23
- Takiuchi, H. P. B., Penteado, Raqueline R. M. Hara, P. e Hara, C. S. (2020). Integrando bases autônomas externas no processamento de consultas em bases rdf distribuíıdas. Em SBBD.

.

6.2 TRABALHOS FUTUROS

Os trabalhos futuros para esta dissertação envolvem:

- Expandir ao conjunto de cláusulas analisadas durante a fase de análise da consulta, como a cláusula OPTIONAL. Assim será possível dar suporte a número maior de consultas SPARQL.
- Criar um plano de consulta capaz de processar diferentes ordenações dos blocos de execução (base proprietária ou base de terceiros). Desse modo é possível utilizar funções de custo para calcular a melhor composição dos blocos de execução.
- Utilizar consultas ASK SPARQL como método de refinamento na seleção das fontes, criando assim um método misto. Ou seja, um método que utiliza esquemas estruturais junto a consultas SPARQL.
- Realizar experimentos com mais servidores. O objetivo é conseguir testar a escalabilidade do sistema FeSHyD. Nesta dissertação foi avaliado o acréscimo de um apenas servidor.
- Submeter consultas aos *endpoints* remotos que retornem mais resultados intermediários. Com isso é esperado que a otimização sobre os servidores de dados seja mais evidente devido ao paralelismo na junção dos resultados intermediários.
- Considerar no plano de consulta a possibilidade de ligação entre as bases de dados de terceiros distintas. Dessa forma, será possível testar o sistema sobre bases com dados externos interligadas (e.g., através da propriedade $owl : sameAs$).
- Realizar experimentos com uma base de dados não balanceada e avaliar o impacto no desempenho do sistema.

.

REFERÊNCIAS

Abdelaziz, I., Mansour, E., Ouzzani, M., Aboulnaga, A. e Kalnis, P. (2017a). Lusail: a system for querying linked data at scale. *Proceedings of the VLDB Endowment*, 11(4):485–498.

Abdelaziz, I., Mansour, E., Ouzzani, M., Aboulnaga, A. e Kalnis, P. (2017b). Query optimizations over decentralized rdf graphs. Em *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, páginas 139–142. IEEE.

Cunha, D. R. e Lóscio, B. F. (2015). An approach for query decomposition on federated sparql query systems. *JIDM*, 6(2):106–117.

da Cunha, D. R. B. e Lóscio, B. F. (2014). oLinDa: uma abordagem para decomposição de consultas em federações de dados interligados. Em *Anais do Simpósio Brasileiro de Banco de Dados*, páginas 137–146.

Görlitz, O. e Staab, S. (2011). Splendid: Sparql endpoint federation exploiting void descriptions. Em *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*, páginas 13–24. CEUR-WS. org.

Haase, P., Nikolov, A., Trame, J., Kozlov, A. e Herzig, D. M. (2017). Alexa, ask wikidata! voice interaction with knowledge graphs using amazon alexa. Em *International Semantic Web Conference (Posters, Demos & Industry Tracks)*.

Hendler, J. (2001). Agents and the semantic web. *IEEE Intelligent systems*, 16(2):30–37.

Ladwig, G. e Tran, T. (2011). Sihjoin: querying remote and local linked data. Em *Extended Semantic Web Conference*, páginas 139–153. Springer.

Levine, J. (2009). *Flex & Bison: Text Processing Tools*. O'Reilly Media, Inc.

Nikolov, A., Haase, P., Trame, J. e Kozlov, A. (2017). Ephedra: Efficiently combining RDF data and services using SPARQL federation. Em *International Conference on Knowledge Engineering and the Semantic Web*, páginas 246–262. Springer.

Oguz, D., Ergenc, B., Yin, S., Dikenelli, O. e Hameurlain, A. (2015). Federated query processing on linked data: a qualitative survey and open challenges. *The Knowledge Engineering Review*, 30(5):545–563.

Özsu, M. T. (2016). A survey of rdf data management systems. *Frontiers of Computer Science*, 10(3):418–432.

Penteado, R. R. d. M. (2017). *Otimização de consultas SPARQL em bases RDF distribuídas*. Tese de doutorado, Universidade Federal do Paraná.

Penteado, R. R. M., Schroeder, R. e Hara, C. S. (2016). Exploring controlled RDF distribution. Em *IEEE CloudCom 2016, Luxembourg, Dezembro 12-15, 2016*, páginas 160–167.

Penteado, R. R. M., Takiuchi, H. P. B. e Hara, C. S. (2019). PAbS: Um processador de consultas sparql sobre bases distribuídas. Em *SBBD Demo*, páginas 18–23.

Rakhmawati, N. A., Umbrich, J., Karnstedt, M., Hasnain, A. e Hausenblas, M. (2013). Querying over federated sparql endpoints—a state of the art survey. *arXiv preprint arXiv:1306.1723*.

Saleem, M., Khan, Y., Hasnain, A., Ermilov, I. e Ngonga Ngomo, A.-C. (2016). A fine-grained evaluation of sparql endpoint federation systems. *Semantic Web*, 7(5):493–518.

Schwarte, A., Haase, P., Hose, K., Schenkel, R. e Schmidt, M. (2011). Fedx: Optimization techniques for federated query processing on linked data. Em *International semantic web conference*, páginas 601–616. Springer.

Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C. e Reynolds, D. (2008). Sparql basic graph pattern optimization using selectivity estimation. Em *Proceedings of the 17th international conference on World Wide Web*, páginas 595–604. ACM.

Umbrich, J., Karnstedt, M., Hogan, A. e Parreira, J. X. (2012). Hybrid sparql queries: fresh vs. fast results. Em *International Semantic Web Conference*, páginas 608–624. Springer.

Wylot, M., Hauswirth, M., Cudré-Mauroux, P. e Sakr, S. (2018). Rdf data storage and query processing schemes: A survey. *ACM Computing Surveys (CSUR)*, 51(4):84.

Zhang, L., Liu, Q., Zhang, J., Wang, H., Pan, Y. e Yu, Y. (2007). Semplore: an ir approach to scalable hybrid query of semantic web data. Em *The Semantic Web*, páginas 652–665. Springer.

APÊNDICE A – CONSULTAS SPARQL E GRAFO DE ESTRUTURA

Neste Apêndice A são apresentadas as quatro consultas SPARQL utilizadas durante a fase de experimentos. Estas Consultas são citadas na Seção 5.3. Também é apresentado o grafo de estrutura utilizado nessa Seção 5.3.

Consulta 1

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?labelProduct ?producer ?classProducer
WHERE {
        ?product rdfs:label ?labelProduct.
        ?product bsbm:producer ?producer.
        ?producer wdt:P31 ?classProducer.
        FILTER (?product = Product1-1)
}
```

Figura A.1: Consulta Q1

Consulta 2

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?labelProduct ?property ?producer ?classProducer
WHERE {
        ?product rdfs:label ?labelProduct.
        ?product bsbm:productPropertyNumeric1 ?property.
        ?product bsbm:producer ?producer.
        ?producer wdt:P31 ?classProducer.
}
```

Figura A.2: Consulta Q2

Consulta 3

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rev: <http://purl.org/stuff/rev#>

SELECT ?rating ?title ?reviewer ?classReviewer
WHERE {
        ?review bsbm:rating2 ?rating.
        ?review dc:title ?title.
        ?review rev:reviewer ?reviewer
        ?reviewer wdt:P31 ?classReviewer
        FILTER (rating <= 3)
}
```

Figura A.3: Consulta Q3

Consulta 4

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?price ?vendor ?classVendor
WHERE {
        ?offer bsbm:price ?labelProduct.
        ?offer bsbm:vendor ?vendor.
        ?vendor wdt:P31 ?classVendor.
}
```

Figura A.4: Consulta Q4

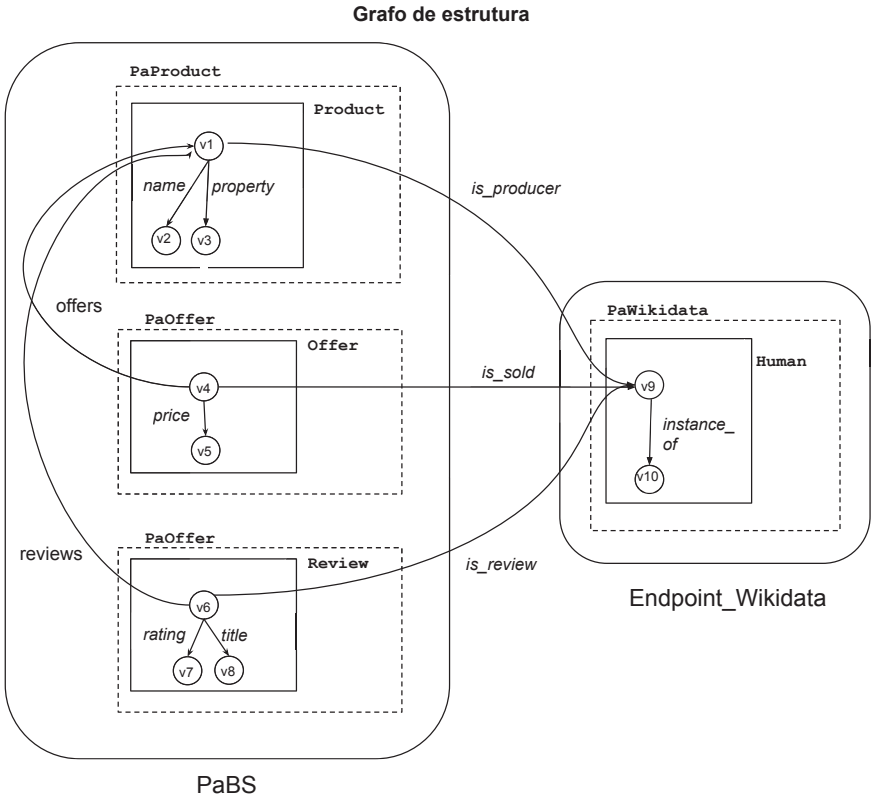


Figura A.5: Grafo de estrutura experimentos