FragRDF: Um Fragmentador de dados RDF baseado em Esquemas

Vinicius Gasparini[1], **Rebeca Schroeder**[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC)
Centro de Ciências Tecnológicas – Joinville – SC – Brasil

viniciuszeiko@gmail.com, rebeca.schroeder@udesc.br

Abstract. *The increasing volume of data published in RDF format requires distributed databases to address such demand. One challenge in this context is the data fragmentation so that it can be distributed later. This paper presents FragRDF, a tool for RDF data fragmentation. Unlike related approaches, FragRDF is able to fragment the database from a previously defined fragmentation schema. The experiments presented in this paper demonstrate that a centralized version of FragRDF is able to generate distributed databases up to 35 million of RDF triples.*

Resumo. *O volume crescente de dados que têm sido publicados em RDF gera a necessidade de utilizar bancos de dados distribuídos capazes de tratar esta demanda. Um desafio neste contexto é a fragmentação destes dados para que possam ser posteriormente distribuídos. Este artigo apresenta FragRDF, uma ferramenta para a fragmentação de dados RDF. Diferentemente de abordagens correlatas, FragRDF é capaz de fragmentar a base a partir de um esquema de fragmentação previamente definido. Os experimentos apresentados neste artigo demonstram que uma versão centralizada da ferramenta é capaz de gerar bases distribuídas de até 35 milhões de triplas RDF.*

1. Introdução

Atualmente, RDF (*Resource Description Framework*) é o padrão para a publicação de dados na Web [W3C 2017b]. O modelo RDF permite descrever recursos baseados em triplas, cada qual constituída por um sujeito, um predicado e um objeto. Fontes de dados RDF são definidas por conjuntos de triplas interligadas, e que podem ser generalizadas por uma estrutura de dados em grafos. Diante da disseminação deste padrão, diversos conjuntos de dados têm sido publicados neste modelo. No entanto, o excessivo volume de dados neste formato gera desafios de gerenciamento destes dados, onde muitas fontes tem seu volume qualificado como Big Data por atingirem a casa dos *zettabytes* [W3C 2017a].

O problema relacionado ao volume de dados vem sendo tratado através da adoção de sistemas de bancos de dados distribuídos ou paralelos [Zeng et al. 2013]. Entretanto, existem diversos métodos para fragmentação de dados neste contexto. Em resumo, os métodos existentes podem ser classificados em duas categorias. A primeira delas aplica métodos de particionamento baseados na estrutura dos dados. Esses métodos empregam desde fragmentações verticais e horizontais que agregam componentes de uma tripla [Abadi et al. 2009], até fragmentação baseada na conectividade de grafos

RDF [Huang and A. 2011]. A segunda classe, aplica conhecimentos da carga de trabalho dos bancos para determinar como os dados relacionados podem ser melhor agrupados [Aluç et al. 2014]. Nesta categoria, uma abordagem é a definição prévia da fragmentação sobre um esquema RDF conhecido. De posse do esquema, alguns trabalhos analisam as principais consultas e determinam como os elementos do esquema são acessados conjuntamente ([Curino et al. 2010], [Papadomanolakis and Ailamaki 2004], [Schroeder and Hara 2015]).

Este artigo tem por objetivo apresentar uma ferramenta, denominada *FragRDF*, que implementa um método de fragmentação baseado em esquemas. O método implementado foi proposto pelo trabalho [Schroeder and Hara 2015], e apresenta uma abordagem particularmente interessante para o contexto RDF onde constantes inserções de dados são esperadas. Uma vez definido como os dados de um esquema RDF devem ser fragmentados, novas inclusões são fragmentadas de acordo com este esquema previamente definido. Em outras abordagens, estas novas inclusões podem ocasionar a re-fragmentação de toda a base, ou uma definição *ad-hoc* da estratégia de fragmentação.

Na Seção 2 deste artigo são apresentados conhecimentos essenciais do modelo RDF e de extração de esquemas. Técnicas de fragmentação RDF são apresentadas na seção seguinte, tendo como foco a identificação de ferramentas relacionadas. Na Seção 4 é apresentada a ferramenta *FragRDF* e detalhes de seu projeto. Em seguida, experimentos sobre o *FragRDF* são apresentados para avaliar o desempenho do fragmentador em uma única máquina. Finalmente, trabalhos futuros e conclusões do trabalho são apresentados pela Seção 6.

2. O Modelo RDF

O *Resource Description Framework* (RDF) foi desenvolvido através do consórcio W3C (*World Wide Web Consortium*). Os autores de [Lima and Carvalho 2005] enfatizam que RDF representa uma arquitetura genérica de metadados, permitindo que informações possam ser representadas na forma de recursos na *Web*. Com RDF, qualquer recurso existente na *Web* pode ser utilizado como forma de representação de informações através de seu conteúdo associado.

Para que os recursos possam ser adequadamente representados, RDF proporciona um modelo para realizar declarações sobre estes recursos. O modelo apresenta a estrutura na forma <sujeito> <predicado> <objeto>, e expressa dessa forma a relação presente entre os recursos sujeito e objeto. O predicado inserido na declaração tem o objetivo de descrever a essência deste relacionamento. Portanto, as declarações efetuadas em RDF se darão pelo formato de triplas (sujeito, predicado, objeto). Assim, utilizando esse tipo de declaração que estabelece um modelo de recursos, propriedades e seus valores objeto correspondentes, o RDF tenta proporcionar uma maneira clara e não ambígua de representar a semântica dos dados por meio de uma codificação compreendida pela máquina [Miller 1998].

A partir deste modelo descrito, um documento RDF pode representar declarações de várias coisas, pessoas, objetos, lugares, que tem propriedades como “é dono”, “tem”, “pertence”, e que se relacionam com valores dos objetos como um livro, um valor numérico, uma outra pessoa. Existem alguns formatos para definição de documentos RDF, diferenciando-se apenas na sintaxe. Dentre os formatos disponíveis destacam-se

principalmente *N-Triples* (NT) e o RDF/XML. Ambos os formatos foram escolhidos para serem utilizados neste trabalho. Isso se deve a sua simplicidade, visto que as triplas são declaradas diretamente e seguem o fluxo de um documento de texto comum. Além disso, suas respectivas sintaxes não possuem novos elementos que necessitam ser interpretados, sendo assim, os URIs podem ser expressos diretamente nas triplas.

O formato *N-Triples* é composto por uma sequência de triplas RDF separadas por um ponto. Um exemplo do formato N-Triples é mostrado na Figura 1. Cada linha do documento representa uma tripla na qual cada elemento da tripla é delimitado por "<" e ">". O ponto no final da linha indica o final da tripla, e a separação entre sujeito, predicado e objeto é designado pelo espaço em branco.

```
<http://www4.wiwiss.de/Product1> <http://www4.wiwiss.de/#type> <http://www4.wiwiss.de/Product> .
<http://www4.wiwiss.de/Product1> <http://www4.wiwiss.de/#label> "tableY" .
<http://www4.wiwiss.de/Product1> <http://wiwiss.de/#dueDate> "2014-05"<http://wiwiss.de/#date> .
<http://www4.wiwiss.de/Offer1> <http://www4.wiwiss.de/#type> <http://www4.wiwiss.de/Offer> .
<http://www4.wiwiss.de/Offer1> <http://www4.wiwiss.de/product> <http://www4.wiwiss.de/Product1> .
<http://www4.wiwiss.de/Offer1> <http://www4.wiwiss.de/vendor> <http://www4.wiwiss.de/Vendor1>
<http://www4.wiwiss.de/Offer1> <http://www4.wiwiss.de/#price> "35.00"<http://www4.wiwiss.de/USD> .
<http://www4.wiwiss.de/Vendor1> <http://www4.wiwiss.de/#type> <http://www4.wiwiss.de/Vendor> .
<http://www4.wiwiss.de/Vendor1> <http://www4.wiwiss.de/#label> "Lider" .
<http://www4.wiwiss.de/Offer2> <http://www4.wiwiss.de/#type> <http://www4.wiwiss.de/Offer> .
<http://www4.wiwiss.de/Offer2> <http://www4.wiwiss.de/product> <http://www4.wiwiss.de/Product1> .
<http://www4.wiwiss.de/Offer2> <http://www4.wiwiss.de/vendor> <http://www4.wiwiss.de/Vendor2>
<http://www4.wiwiss.de/Offer2> <http://www4.wiwiss.de/#price> "50.00"<http://www4.wiwiss.de/USD> .
<http://www4.wiwiss.de/Vendor2> <http://www4.wiwiss.de/#type> <http://www4.wiwiss.de/Vendor> .
<http://www4.wiwiss.de/Vendor2> <http://www4.wiwiss.de/#label> "Proger" .
```

Figura 1. Exemplo de formato RDF *N-Triples*

Por meio das declarações na Figura 1, é possível observar que um recurso pode ser utilizado em múltiplas triplas RDF. Um recurso pode ser referenciado como sujeito em uma tripla bem como pode ser referenciado como objeto em outra tripla, o que torna possível fazer conexões entre triplas. Em um conjunto de dados RDF as relações destacadas entre triplas se tornam visíveis através da representação em formato de grafo, contendo nós (vértices) e arcos (arestas). Tanto o sujeito como o objeto são representados por vértices e a relação entre estes recursos, que é efetuada pelo predicado, é representada pela aresta. A representação do predicado é rotulada e a aresta dirigida da origem (sujeito) para o destino (objeto) da relação. A Figura 2 mostra a representação em grafo do documento RDF apresentado na Figura 1.

Embora todos os elementos do documento NT estejam apresentados no grafo da Figura 2, a ilustração destaca um esquema de dados RDF que pode ser extraído a partir do documento. Assim, um dos mecanismos que o modelo utiliza para prover uma especificação mais qualificada de dados é a utilização do *RDF Schema*. Este complemento do modelo RDF é uma extensão semântica e provê um vocabulário para modelagem de dados [Brickley and Guha 2014]. Com o *RDF Schema* é possível descrever determinadas propriedades para recursos, permitindo que os mesmos possam ser categorizados, definir limites através de domínios, agrupamentos, especificação de tipos, entre outros.

Esta definição de características para recursos RDF é importante, pois o principal objetivo do RDF é a declaração de recursos. Estes, então, estão relacionados entre si por meio do modelo RDF, porém, sem realizar algum tipo de organização ou tratamento. O

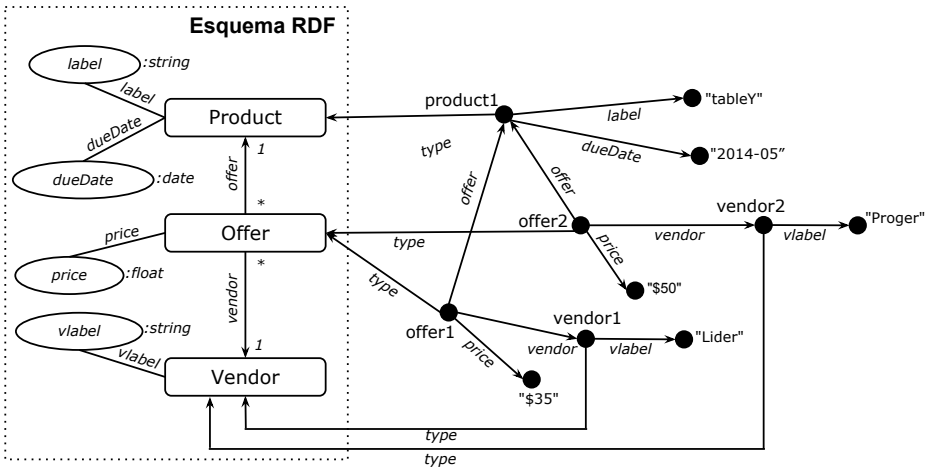

Figura 2. Esquema e Conjunto de Dados RDF (Adaptado de [Schroeder and Hara 2015])

RDF Schema tem justamente este objetivo, ou seja, providenciar meios que tratem aspectos de recursos para que os mesmos sejam representados adequadamente. É necessário salientar que os termos definidos pelo *RDF Schema* não são obrigatórios para a declaração de recursos. Este está disponível apenas como alternativa para qualificar um conjunto de recursos e designá-los de acordo com propriedades. Por esta razão o modelo RDF é conhecido como um modelo livre de esquema, pois documentos RDF podem ser definidos sem o uso de um RDF Schema. Conforme apontado por [Pham and Boncz 2013], apesar de RDF constituir um modelo livre de esquema, é possível a extração de estruturas de dados a partir de diversas fontes RDF. A Figura 2 destaca um representação do esquema RDF extraído do documento da Figura 1.

Um processo de extração de esquemas pode ser definido a partir da identificação de tipos pelos predicados *type*. Observe que *product1*, *offer1*, *offer2*, *vendor1* e *vendor2* constituem sujeitos de triplas contendo a propriedade *type* com os objetos *Product*, *Offer* e *Vendor*. Logo, os referidos objetos podem ser utilizados para definir entidades do esquema. Na sequência uma análise das triplas que relacionam instâncias dessas entidades podem identificar valores literais associados às entidades, bem como os relacionamentos entre entidades. Apesar do conjunto RDF do exemplo estar reduzido, ainda é possível identificar a cardinalidade dos relacionamentos entre entidades. No exemplo, 1 instância de *Offer* está associada a 1 instância de *Product* e a 1 instância de *Vendor*. No entanto, 1 instância de *Product* poderá envolver N instâncias de *Offer*, assim como 1 instância de *Vendor* N instâncias de *Offer*.

Existem alguns trabalhos na literatura que propõem soluções para a extração de esquemas RDF. A estratégia descrita no exemplo está definida no trabalho de [Maia 2005]. Alguns outros métodos são apresentados em [Pham and Boncz 2013] e [Neumann and Moerkotte 2011]. Ambos identificam entidades do esquema baseados no conceito de *Characteristic Sets*, onde a similaridade de elementos é calculada a partir de propriedades em comum. A existência destes trabalhos demonstra a necessidade da extração de esquemas RDF para otimização de consultas.

(a) *Triple Store* RDF

Sujeito	Propriedade	Objeto
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox,Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr,Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

(b) Fragmentação Vertical

Sujeito	type	title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL

Sujeito	Propriedade	Objeto
ID1	author	"Fox,Joe"
ID2	artist	"Orr,Tim"
ID2	language	"French"
ID3	language	"English"

(c) Fragmentação Híbrida

Classe: BookType

Sujeito	Title	Author	copyright
ID1	"XYZ"	"Fox,Joe"	"2001"
ID3	"MNP"	NULL	NULL

Classe: CDType

Sujeito	Title	Author	copyright
ID2	"ABC"	"Orr,Tim"	"1985"

Sujeito	Propriedade	Objeto
ID2	language	"French"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

Figura 3. Fragmentação RDF (Adaptado de [Abadi et al. 2009])

3. Métodos de Fragmentação

O termo fragmentação é definido em [Zilio et al. 2004] como um dos elementos do particionamento de banco de dados. Assim, o particionamento de um BD é definido em duas etapas complementares que compreendem a fragmentação e a alocação de dados. Na fragmentação espera-se obter agregações de dados cuja unidade corresponda a uma unidade de armazenamento do BD. Já na alocação, defini-se o destino de cada unidade para uma das máquinas que constitui o BD distribuído/paralelo.

Exemplos de fragmentos RDF são apresentados pela Figura 3. Nestes exemplos propostos por [Abadi et al. 2009], dados RDF são fragmentados conforme o modelo relacional. Assim, cada fragmento corresponde a uma tupla de uma tabela. Em repositórios conhecidos como *triple store*, todas as triplas são colocadas em uma única tabela que contém 3 campos: sujeito, predicado e objeto (Figura 3(a)). Na Figura 3(b), fragmentos verticais correspondem a um agrupamento baseado em valores de sujeitos ou objetos das triplas. Assim, propriedades consideradas relacionadas são agrupadas em uma única tabela juntamente com seus respectivos sujeitos e predicados. Na Fragmentação Híbrida da Figura 3(c), cada tabela corresponde a uma classe que agrupa triplas com os mesmos valores para o predicado *type*. Os campos destas tabelas de classe correspondem a propriedades utilizadas por triplas destas classes. Triplas não qualificadas pelos critérios da Fragmentação, são colocadas em uma *triple store* como mostra às últimas tabelas das Figuras 3 (b) e (c).

Apesar de visar o modelo relacional, os exemplos da Figura 3 ilustram algumas possibilidades de fragmentação de dados RDF baseadas nas estruturas de suas triplas. Quanto à fragmentação horizontal não contemplada por estes exemplos, aplica-se o princípio de conjuntos homogêneos de dados definidos em [Zilio et al. 2004]. Por exemplo, no modelo relacional fragmentos horizontais correspondem a sub-conjuntos de tuplas. No contexto RDF, fragmentos horizontais podem ser estabelecidos por valores de um sujeito. No exemplo anterior, todas as triplas associadas ao sujeito *ID1* poderiam estar em um único fragmento, e assim por diante.

Existem ainda métodos de fragmentação baseados em diferentes heurísticas. Um conjunto destes trabalhos foca em composições de grafos RDF para determinar seus fragmentos. Por exemplo, no trabalho de [Huang and A. 2011] um grafo RDF é dividido em fragmentos através do particionador de grafos METIS [Karypis and Kumar 1999]. A ideia deste método é considerar algumas propriedades do grafo, como a conectividade de nós, para determinar os cortes do grafo. Uma abordagem similar é implementa por

[Hose and Schenkel 2013]. Em geral, a fragmentação baseada na composição de grafos requer analisar todo o conjunto de dados para determinar a solução.

Um outro conjunto de soluções baseadas em heurísticas são àquelas que utilizam a análise da carga de trabalho do BD. Em geral, padrões de consultas são analisados para definir os fragmentos de forma a agrupar dados que são frequentemente acessados em conjunto [Yang et al. 2012] [Aluç et al. 2014]. Uma abordagem neste conjunto são aquelas que especificam os padrões de consulta a partir de esquemas RDF [Papadomanolakis and Ailamaki 2004] [Schroeder and Hara 2015]. Nestes trabalhos são aplicadas abordagens de extração de esquemas como descrito na Seção 2, e então definidos os padrões de consulta e fragmentos sobre o esquema.

Neste trabalho, o método de fragmentação proposto por [Schroeder and Hara 2015] é implementado. Nele, um esquema RDF é previamente fragmentado conforme os dados acessados em conjunto dos padrões de consulta de uma carga de trabalho. Em seguida, um conjunto de dados RDF é processado de forma que cada dado é associado a um elemento do esquema e enviado ao fragmento respectivo a esse elemento. Um exemplo deste processo é apresentado pela Figura 4. Esta abordagem de fragmentação difere dos demais métodos heurísticos por poder ser aplicada e re-aplicada a qualquer dado que esteja conforme ao esquema RDF previamente definido. Esta vantagem é relevante para o contexto RDF onde a inserção de dados ao conjunto pode ocorrer a qualquer tempo. Nas outras soluções, estas inserções oneram o processo por precisar reavaliar todo o conjunto de dados, e não somente os novos dados inseridos.

Uma constatação importante na revisão de todos estes trabalhos foi a inexistência de ferramentas disponíveis e que implementam os respectivos processos. A exceção se dá para a ferramenta METIS [Karypis and Kumar 1999]. Entretanto, trata-se de uma ferramenta genérica para particionar grafos, além de trabalhar sobre heurísticas que necessitam avaliar todo o grafo para definir a fragmentação. Este trabalho tem por foco desenvolver e tornar disponível uma ferramenta que implementa um destes métodos, neste caso o proposto em [Schroeder and Hara 2015]. A seção seguinte descreve esta ferramenta, denominada *FragRDF*.

4. *FragRDF*

FragRDF é uma ferramenta desenvolvida para fragmentar um conjunto de dados RDF de acordo com um esquema de fragmentação. A Figura 4 exemplifica o processo de fragmentação em termos de sua entrada e saída. Além de um conjunto de dados, *FragRDF* espera como entrada um esquema de fragmentação, que nada mais é do que o esquema dos dados previamente fragmentado. O conjunto de dados pode envolver um ou mais arquivos nos formatos NT ou RDF/XML. A ferramenta analisa cada dado do conjunto para determinar a qual elemento do esquema ele se refere. Uma vez identificado, o dado é colocado no fragmento respectivo.

O esquema de fragmentação é dado por um arquivo XML que define a qual fragmento cada elemento do esquema deve ser colocado. A Figura 5 apresenta o esquema de fragmentação respectivo ao esquema de entrada da Figura 4. Cada elemento do esquema é tratado pela tag *item* que possui os atributos *name*, *entity* e *frag*. O atributo *name* refere-se a uma propriedade, ou aresta do grafo do esquema. Já o atributo *entity* especifica

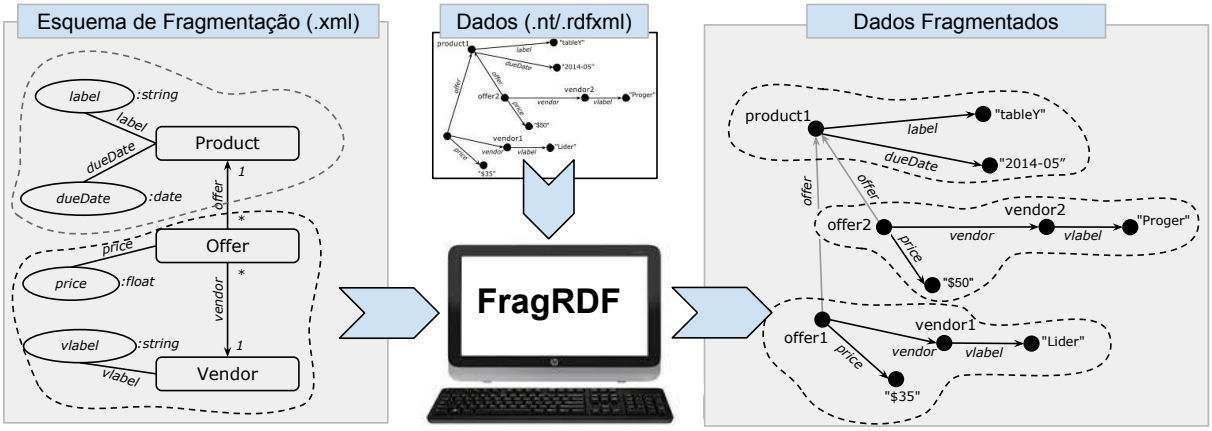

Figura 4. Entrada e Saída do Processo de Fragmentação

a entidade do esquema que constitui a origem da aresta respectiva ao *name*. Por fim, o atributo *frag* define o fragmento no qual o dado deve ser colocado.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schemafrag> <!-- definitions for fragmentation schema -->
3      <items> <!-- map elements/attributes to fragments -->
4          <item name="label" entity="Product" frag="1"/>
5          <item name="dueDate" entity="Product" frag="1"/>
6          <item name="offer" entity="Offer" frag="2"/>
7          <item name="price" entity="Offer" frag="2"/>
8          <item name="vendor" entity="Offer" frag="2"/>
9          <item name="vlabel" entity="Vendor" frag="2"/>
10     </items>
11 </schemafrag>
```

Figura 5. Definição do Esquema de Fragmentação

No exemplo da Figura 4, o processo gerou 3 fragmentos de saída, 1 do tipo *frag=1* e 2 do tipo *frag=2*. Os fragmentos de saída podem ser colocados em um arquivo por tipo de fragmento (valor de *frag*), ou um arquivo por fragmento. O formato dos arquivos segue o mesmo formato do aquivo de dados (.nt ou .rdfxml). Observe que por haver duas instâncias da entidade *Offer* dois fragmentos do tipo 2 são gerados. Uma outra observação diz respeito às arestas entre fragmentos. No exemplo, verifica-se que as arestas *offer* encontram-se no corte dos fragmentos. No entanto, conforme especificado no arquivo .xml, esta propriedade foi enquadrada para *frag=2*. Desta forma, nos fragmentos de *Offer*, uma aresta do tipo *offer* é mantida e aponta para o fragmento que contém *product1*.

O fragmento tipo *frag=2* envolve uma aresta que relaciona duas entidades (*Offer* e *Vendor*). Neste caso, a ferramenta gera um fragmento por cada instância da entidade que dá origem às arestas de relacionamento, que neste exemplo é *Offer*. Havendo mais de um relacionamento por tipo de fragmento, a ferramenta define os fragmentos a partir da primeira entidade que aparecer no arquivo do esquema de fragmentação para o respectivo tipo (valor de *frag*). Uma observação importante é a possibilidade da geração de elementos redundantes. Observe que uma instância de *Vendor* pode estar associada a várias instâncias de *Offer*. Isto faz com que seja possível a replicação de uma mesma instância de *Vendor* em diferentes instâncias de *Offer*. Na implementação atual de *FragRDF* a geração

Tabela 1. Estatísticas das Bases Avaliadas

#Produtos	#Triplas	Tamanho (MB)	#Arquivos
50	22.729	5,71	1
100	40.377	10,22	1
200	75.550	19,21	1
1.000	374.911	95,72	1
2.000	725.305	186,04	1
5.000	1.809.874	465,19	1
10.000	3.564.773	919,09	1
10.000	3.564.773	897,62	5
20.000	7.073.571	1.790,02	10
30.000	10.628.484	2.693,76	15
40.000	14.172.772	3.594,61	20
60.000	21.198.599	5.384,19	30
100.000	35.272.182	8.973,50	50

destas redundâncias são permitidas. No entanto, em uma versão futura pretende-se criar mecanismos para controle de redundâncias, de forma que o usuário possa definir se aceita a geração de dados redundantes, ou ainda quantas réplicas são permitidas por elemento.

FragRDF foi desenvolvida em Python, com o suporte das bibliotecas *ElementTree* e *RDF Lib*. *ElementTree* é uma biblioteca *built-in* do Python que trabalha de forma a armazenar estruturas hierarquizadas de dados em objetos tipo *containers*. Esta biblioteca foi utilizada para processar o arquivo de entrada definido em XML. A biblioteca *RDFLib*[1] foi utilizada para processar os dados de entrada contendo os arquivos de dados. *RDFLib* é uma biblioteca externa ao Python e é capaz de interpretar dados RDF como um grafo não ordenado. Trabalhando com operadores comuns à linguagem Python, a biblioteca opera buscando por parâmetros e tipos previamente declarados. A seção a seguir relata alguns experimentos que demonstram o desempenho da versão atual da ferramenta.

5. Experimentos

Em virtude do elevado volume que bases RDF estão sujeitas, um estudo experimental foi realizado para avaliar o desempenho de *FragRDF*. A avaliação foi executada em uma única máquina, a partir da qual ocorreu a execução da ferramenta avaliada. O ambiente de execução conta com um processador AMD Phenom(tm) II X4 B93 de 4 núcleos e 1,6 gHz, 4Gb de RAM e sistema operacional ubuntu 16.04 LTS. O conjunto de dados a ser fragmentado foi obtido a partir do gerador de bases RDF do Berlin SPARQL Benchmark (BSBM) [Bizer and Schultz 2009].

O BSBM é baseado em um caso de uso de um sistema de *e-commerce*, onde uma lista de produtos é oferecida por vendedores e posteriormente avaliada por clientes através de revisões. Parte do esquema de dados do BSBM foi utilizado no exemplo da Figura 2. Para a geração das bases, o *benchmark* utiliza um fator de escala baseado no número de produtos a gerar. A Tabela 1 relaciona as bases geradas e utilizadas nesta avaliação. O esquema RDF do BSBM foi fragmentado de forma que seus elementos foram distribuídos em 6 tipos de fragmentos.

Em uma primeira avaliação, as bases foram geradas no formato NT em um único

[1]https://github.com/RDFLib/rdflib

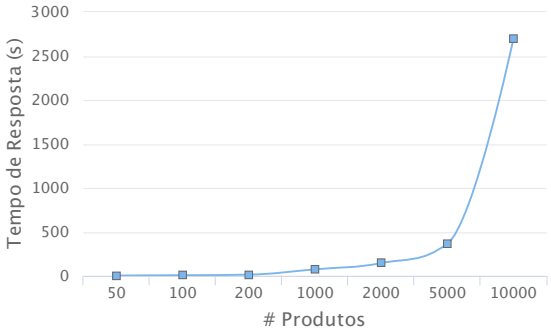

Figura 6. #Arquivos=1

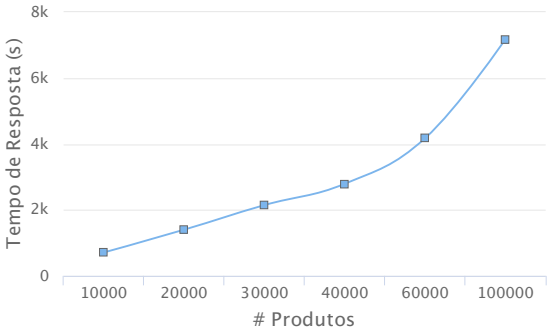

Figura 7. Arquivos de 180 MB

arquivo. Os resultados apresentados pela Figura 6 correspondem ao tempo de resposta em segundos para a completa execução do processo *FragRDF*. Como pode ser verificado, o tempo de resposta apresentou um crescimento considerável para a fragmentação da base de 10.000 produtos, atingindo a casa dos 2.700 segundos. Uma das razões para esta elevação se deu em virtude do tamanho do arquivo único da base que foi de 919,09 MB. Esta constatação foi comprovada pelo segundo experimento, em que optou-se por gerar a base em mais arquivos de no máximo 180 MB cada. Como mostra a segunda metade da Tabela 1, foram geradas bases de 10.000 a 100.000 produtos com o número de arquivos conforme apresentado na última coluna. Os resultados deste segundo experimento estão apresentados na Figura 7, onde atingiu-se a marca de 7.182,5 segundos para a base de 100.000 produtos.

Os resultados deste estudo experimental mostram que *FragRDF* consegue gerar bases de até 35.272.182 triplas em uma instalação em uma única máquina. Não foi possível o teste com bases de tamanhos superiores em virtude do limite de memória da máquina utilizada. Entretanto, como algumas bases RDF podem atingir a casa de trilhões de triplas [W3C 2017a], uma implementação paralela da ferramenta encontra-se em desenvolvimento. Esta nova implementação fará uso do *framework* de processamento paralelo, onde cada nó de um cluster de máquinas terá a responsabilidade de fragmentar parte dos arquivos contendo o conjunto de dados. Espera-se que deste modo, o processo *FragRDF* se torne escalável em relação ao tamanho das bases de dados a fragmentar e o conjunto de máquinas que podem cooperar na fragmentação de porções destas bases.

6. Conclusão

Este artigo apresentou a ferramenta *FragRDF* que é capaz de fragmentar conjuntos de dados RDF a partir de um esquema de fragmentação pré-estabelecido. O estudo experimental apresentado neste trabalho demonstrou que a ferramenta é capaz de gerar bases fragmentadas de ate 35.272.182 triplas em uma instalação de uma única maquina. Como trabalhos futuros pretende-se prosseguir com a versão paralela da ferramenta, de forma que sua capacidade de processamento se torne escalável. Além disto, em uma versão futura espera-se disponibilizar mecanismos para controlar a redundância de dados gerados nos fragmentos. Por fim, experimentos mais exaustivos e envolvendo conjuntos de dados RDF reais são esperados.

Referências

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

[Aluç et al. 2014] Aluç, G., Özsu, M. T., and Daudjee, K. (2014). Workload matters: Why rdf databases need a new design. *Proc. VLDB Endow.*, 7(10):837–840.

[Bizer and Schultz 2009] Bizer, C. and Schultz, A. (2009). The Berlin SPARQL Benchmark. In *International Journal on Semantic Web & Information Systems*.

[Brickley and Guha 2014] Brickley, D. and Guha, R. V. (2014). Rdf schema 1.1. http://www.w3.org/TR/rdf-schema/.

[Curino et al. 2010] Curino, C., Jones, E., Zhang, Y., and Madden, S. (2010). Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57.

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

[Huang and A. 2011] Huang, J. and A., D. J. (2011). Scalable sparql querying of large rdf graphs. *The VLDB Journal*, 4(1):1123–1134.

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

[Lima and Carvalho 2005] Lima, J. C. and Carvalho, C. L. (2005). Resource description framework (rdf). Relatório técnico, Instituto de Informática - Universidade Federal de Goiás.

[Maia 2005] Maia, A. S. (2005). Uma Abordagem para Extração de Esquemas RDF . Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação), UDESC.

[Miller 1998] Miller, E. (1998). *An Introduction to the Resource Description Framework*. D-Lib Magazine.

[Neumann and Moerkotte 2011] Neumann, T. and Moerkotte, G. (2011). Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *International Conference on Data Engineering*, pages 984–994.

[Papadomanolakis and Ailamaki 2004] Papadomanolakis, S. and Ailamaki, A. (2004). Autopart: automating schema design for large scientific databases using data partitioning. In *International Conference on Scientific and Statistical Database Management*, pages 383–392.

[Pham and Boncz 2013] Pham, M. and Boncz, P. (2013). Ieee 29th international conference on data engineering workshops. In *Self-organizing structured RDF in MonetDB*, pages 310–313.

[Schroeder and Hara 2015] Schroeder, R. and Hara, C. S. (2015). Partitioning templates for rdf. In *Advances in Databases and Information Systems*, pages 305–319. Springer International.

[W3C 2017a] W3C (2017a). Large Triple Stores. https://www.w3.org/wiki/LargeTripleStores.

[W3C 2017b] W3C (2017b). Semantic Web:Resource Description Framework. https://www.w3.org/RDF/.

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

[Zeng et al. 2013] Zeng, K., Yang, J., Wang, H., Shao, B., and Wang, Z. (2013). A Distributed Graph Engine for Web Scale RDF data. *Proceedings of the VLDB Endowment*, 6(4):265–276.

[Zilio et al. 2004] Zilio, D. C., Rao, J., Lightstone, S., Lohman, G., Storm, A., Garcia-Arellano, C., and Fadden, S. (2004). Db2 design advisor: Integrated automatic physical database design. In *VLDB*, pages 1087–1097.