PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Universidade Federal do Rio de Janeiro

DISSERTAÇÃO DE MESTRADO

RAPHAELA PEDREIRA NUNES

PROCESSAMENTO ANALÍTICO DISTRIBUÍDO DE GRANDES VOLUMES DE DADOS MULTIDIMENSIONAIS: UMA ABORDAGEM BASEADA EM GRAFOS RDF

Rio de Janeiro
2016

Instituto de Matemática

Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RAPHAELA PEDREIRA NUNES

PROCESSAMENTO ANALÍTICO DISTRIBUÍDO DE GRANDES VOLUMES DE DADOS MULTIDIMENSIONAIS: UMA ABORDAGEM BASEADA EM GRAFOS RDF

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como requisito parcial à obtenção do título de Mestre em Informática.

Orientadora: Profª. Adriana Santarosa Vivacqua, D.Sc.

Co-orientadora: Profª. Maria Luiza Machado Campos, Ph.D.

Rio de Janeiro
2016

CIP - Catalogação na Publicação

Pedreira Nunes, Raphaela
P972p Processamento analítico distribuído de grandes volumes de dados multidimensionais: uma abordagem baseada em grafos RDF / Raphaela Pedreira Nunes. - Rio de Janeiro, 2016.
154 f.

Orientadora: Adriana Santarosa Vivacqua.
Coorientadora: Maria Luiza Machado Campos.
Dissertação (mestrado) - Universidade Federal do Rio de Janeiro, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Programa de Pós-Graduação em informática, 2016.

1. Processamento Analítico. 2. Modelo Multidimensional. 3. Linked Open Data. 4. Distribuição. 5. Apache Hadoop. I. Vivacqua, Adriana Santarosa, orient. II. Campos, Maria Luiza Machado, coorient. III. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a).

Raphaela Pedreira Nunes

PROCESSAMENTO ANALÍTICO DISTRIBUÍDO DE GRANDES VOLUMES DE DADOS MULTIDIMENSIONAIS: UMA ABORDAGEM BASEADA EM GRAFOS RDF

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como requisito parcial à obtenção do título de Mestre em Informática.

Aprovada em: Rio de Janeiro, 27 de junho de 2016.

Profª. Adriana Santarosa Vivacqua, D.Sc. - PPGI/UFRJ

Profª. Maria Luiza Machado Campos, Ph.D. - PPGI/UFRJ

Profª. Jonice de Oliveira Sampaio, D.Sc. - PPGI/UFRJ

Profª. Fernanda Araujo Baião Amorim, D.Sc. - UNIRIO

Prof. Sérgio Manuel Serra da Cruz, D.Sc. - UFRRJ

AGRADECIMENTOS

A Deus, por renovar as minhas energias e me manter perseverante na caminhada.

Aos meus pais e irmã, que me acompanharam por toda a jornada, não me abandonando em nenhum momento. Agradeço pela paciência, pelas palavras de apoio e pelo precioso suporte nos momentos mais difíceis.

Aos meus avós, que continuam sendo exemplos para mim, mesmo não estando mais presentes em vida.

A minha orientadora, professora Adriana Santarosa Vivacqua, e co-orientadora, Maria Luiza Machado Campos, pelos ensinamentos, conselhos e incansável apoio oferecido desde os primeiros momentos do curso de mestrado.

Aos colegas do Centro de Referência em Big Data da UFRJ (Ana Carolina Almeida, Thiago Moreira, Cristiane Ceia, Mayara Morais e Gabriel Marques) e ao colega Renan Souza, com quem troquei e adquiri os primeiros conhecimentos sobre tecnologias de distribuição de processamento de dados. Agradeço pela ajuda de todos na montagem do *cluster* Hadoop que foi utilizado nos experimentos realizados nesta dissertação.

Aos professores Fernanda Baião, Jonice Sampaio e Sérgio Serra, meu agradecimento por aceitarem fazer parte da banca examinadora e pelos conhecimentos transmitidos.

Meus agradecimentos especiais à Universidade Federal do Rio de Janeiro (UFRJ) pelo importante suporte oferecido desde minha graduação, apoiando meu desenvolvimento intelectual e profissional e me ajudando a alcançar boas oportunidades profissionais.

RESUMO

NUNES, Raphaela Pedreira. **Processamento analítico distribuído de grandes volumes de dados multidimensionais**: uma abordagem baseada em grafos RDF. 2016. 154 f. Dissertação (Mestrado em Informática)- Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2016.

A atual difusão de técnicas de Web Semântica e ênfase em dados abertos resultaram em vários conjuntos de dados multidimensionais sendo disponibilizados na Web seguindo os princípios de Dados Aberto Interligados (LOD). Podem ser citados como exemplos de dados multidimensionais os dados oriundos de processos de medição, tais como dados capturados através de sensores, dados pluviométricos e indicadores socioeconômicos. Devido a sua estrutura subjacente de grafo, o processamento analítico de dados interligados ainda apresenta frequentes problemas de eficiência e escalabilidade, representando um grande desafio no contexto da Web Semântica. Diante deste cenário, apresentamos uma abordagem para apoiar a execução eficiente de consultas analíticas SPARQL sobre grandes volumes de dados multidimensionais no formato RDF. Nossa abordagem se baseia em tecnologias de distribuição oferecidas pelo *framewok* Apache Hadoop para permitir que consultas analíticas SPARQL sejam executados de forma distribuída em cima de dados RDF armazenados em um banco de dados distribuído.

Palavras-chave: Processamento Analítico. Modelo Multidimensional. Linked Open Data. Distribuição. Apache Hadoop.

ABSTRACT

NUNES, Raphaela Pedreira. **Distributed analytical processing of large volumes of multidimensional data**: an approach based in graph data. 2016. 154 f. Dissertation (Master's Degree in Computing Science)-Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2016.

The current spread of Semantic Web techniques and emphasis on open data have resulted in multiple multidimensional datasets being released online, following Linked Open Data (LOD) principles. Data generated from measurement processes can be cited as examples of multidimensional data, such as sensor data, rainfall data and socioeconomic indicators. Due to its underlying graph structure, linked data still has efficiency and scalability problems, and analytical processing of these data is a major challenge in the context of Semantic Web. We present an approach to support efficient execution of analytical SPARQL queries on large volumes of multidimensional data in RDF format. Our approach is based on distribution technologies provided by the Apache Hadoop framework to enable analytical SPARQL queries to be run in a distributed manner over RDF data stored in a distributed database.

Keywords: Analytical Processing. Multidimensional Model. Linked Open Data. Distribution. Apache Hadoop.

LISTA DE FIGURAS

Figura 1: Cubo de dados (Adaptado de CHAUDHURI, DAYAL, 1997) 24
Figura 2: Modelo conceitual do projeto PingER baseado em um esquema floco de neve 28
Figura 3: Estrutura do arquivo SLAC - Métrica Average RTT - Dia 10/09/2014 - Pacote de dados 100 bytes 29
Figura 4: Taxa de crescimento dos arquivos com granularidade hora entre 1998 e 2014 31
Figura 5: Quantidade de pontos ocorridos em arquivos entre 1998 e 2014 31
Figura 6: Quantidade de pontos em arquivos por métrica entre 1998 e 2014 32
Figura 7: Exemplo de árvore de álgebra SPARQL 42
Figura 8: Termos chaves do Vocabulário RDF Data Cube (CYGANIAK; REYNOLDS; TENNISON, 2013) 44
Figura 9: Ontologia PingER original (SOUZA et al., 2014) 48
Figura 10: Arquitetura do HDFS (Adaptado de BORTHAKUR, 2008) 55
Figura 11: Exemplo *job* MapReduce 57
Figura 12: Componentes da arquitetura do Cloudera Impala (Adaptado de CHAUHAN, 2013) 60
Figura 13: Exemplo de armazenamento orientado por linha e coluna 61
Figura 14: Nós do cluster CDH e seus componentes 65
Figura 15: Visão geral da arquitetura da aplicação *Impala MultiLOD* 67
Figura 16: Visão geral do processo de carga de dados RDF em tabelas no Impala 68
Figura 17: Exemplo de Tabela de Triplas 69
Figura 18: Exemplos de Tabelas de Particionamento Vertical 70
Figura 19: Exemplo de Tabela Proprietária Clusterizada 72
Figura 20: Exemplo de Tabela Proprietária de Classe para triplas cujo sujeito é da classe "Country" 72
Figura 21: Listas geradas para a Tabela Proprietária de Classe "Country" na fase de pré-processamento 76
Figura 22: Chamada ao método Reduce para gerar Tabela Proprietária da Classe "Country" 78
Figura 23: Esquema de funcionamento do Tradutor SPARQL para SQL 81
Figura 24: Tela onde é feita a tradução da consulta SPARQL 94
Figura 25: Tela com o resultado da execução da consulta SQL traduzida no Impala 94
Figura 26: Modelagem da ontologia do domínio PingER utilizando conceitos do vocabulário RDF Data Cube 97
Figura 27: Relacionamentos da classe qb:Observation na ontologia do domínio PingER 102

Figura 28: Parte do *workflow* ETC para triplificação de dados gerais 105

Figura 29: Fluxo para carga da dimensão Tempo 107

Figura 30: Visão panorâmica do *workflow* ETC que realiza a triplificação dos dados de medidas de rede 108

Figura 31: *Job* para carga de arquivos de dados no HDFS 110

Figura 32: Interface de configuração de conexão do Kettle com o HDFS 111

Figura 33: Comparação dos tempos de execução das consultas SPARQL traduzidas para SQL a partir do tradutor com os tempos de execução das mesmas consultas escritas diretamente em SQL executadas sobre 3 conjuntos de dados com 100M, 200M e 300M triplas, aproximadamente 124

Figura 34: Consulta Q1 escrita diretamente na linguagem SQL executada a partir do Impala-shell 126

Figura 35: Consulta Q1 escrita na linguagem SPARQL, traduzida para SQL e executada a partir do tradutor de linguagem SPARQL para SQL 126

Figura 36: Comparação dos tempos de execução das consultas SPARQL traduzidas e executadas a partir da aplicação *Impala MutiLOD* com os tempos de execução das mesmas consultas SPARQL executadas a partir do Virtuoso Triple Store utilizando três conjuntos de dados com 100M, 200M e 300M triplas, aproximadamente 131

LISTA DE TABELAS

Tabela 1: Quantidade total de arquivos referentes a 16 métricas de rede, granularidade hora, pacote de 100 bytes, período de tempo de 01/01/1998 até 10/09/2014.....30

Tabela 2: Quantidade total de arquivos compactados referentes a 16 métricas de rede, todas granularidades temporal e regional, pacotes de dados de 100 e 1000 bytes no período de tempo de 01/01/1998 até 20/10/2014 33

Tabela 3: Conjuntos de dados do PingER selecionados para testar a aplicação 104

Tabela 4: Estatísticas dos processos de triplificação dos conjuntos de dados de medidas.....109

Tabela 5: Estatísticas dos processos de carga dos arquivos N-Triples no HDFS.....111

Tabela 6: Quantidade de linhas e colunas em cada Tabela Proprietária de Classe criada 113

Tabela 7: Tamanho das tabelas externas e internas testadas e partições criadas.....115

Tabela 8: Tempo de criação, inserção de dados e cálculo de estatísticas nas tabelas testadas 117

Tabela 9: Tempos de execução de cada consulta SQL em cima de tabelas externas e internas utilizando conjuntos de dados de medidas com 100M, 200M e 300M de triplas, aproximadamente.....120

Tabela 10: Tempos de execução de cada consulta SPARQL traduzidas e executadas a partir do tradutor utilizando conjuntos de dados de medidas com 100M, 200M e 300M de triplas, aproximadamente.....123

Tabela 11: Estatísticas dos processos de carga de arquivos no formato N-Triples no Virtuoso Triple Store utilizando a estratégia de *bulk loading* 129

Tabela 12: Tempos de execução das consultas SPARQL executadas a partir do Virtuoso Triple Store carregado com conjuntos de dados de medidas com 100M, 200M e 300M de triplas, aproximadamente.....130

LISTA DE QUADROS

Quadro 1: Operadores da Álgebra e Sintaxe SPARQL 41
Quadro 2: Comparação das abordagens propostas nos trabalhos relacionados 45
Quadro 3: Vantagens e desvantagens dos esquemas de armazenamento levantados 74
Quadro 4: Conjunto de consultas SQL criado para testar o desempenho dos 3 tipos de tabelas no Impala 118

LISTA DE SIGLAS E ABREVIAÇÕES

BGP	Padrão Básico de Grafo
BSBM	*Berlin SPARQL Benchmark*
CDH	*Cloudera Distributed Hadoop*
CRBD-UFRJ	Centro de Referência em Big Data da UFRJ
CSV	*Character-Separated Values*
DML	*Data Manipulation Language*
ETC	Extração, Transformação e Carga
GRECO-UFRJ	Grupo de Engenharia do Conhecimento da UFRJ
HDFS	*Hadoop Distributed File System*
LOD	*Linked Open Data*
LUBM	*Lehigh University Benchmark*
MDX	*Multi-Dimensional Expressions*
MOS	*Mean Opinion Score*
MPP	Processamento Massivo Paralelo
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
OWL	*Web Ontology Language*
PingER	*Ping End-to-end Reporting*
RDF	*Resource Description Framework*
RDB2RDF	*Relational Database to RDF*
SGBD	Sistema de Gerenciamento de Banco de Dados
SLAC	*Stanford National Accelerator Laboratory*
SMDX	*Statistical Data and Metadata Exchange*
SP2Bench	*SPARQL Performance Benchmark*
SPARQL	*SPARQL Protocol and RDF Query Language*
SQL	*Structured Query Language*
TCL	*Transaction Control Language*
TPC	*Transaction Processing Performance Council*
TSV	*Tab-Separated Values*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO 15

1.1 MOTIVAÇÃO 15

1.2 QUALIFICAÇÃO DO PROBLEMA 17

1.3 OBJETIVO 19

1.4 RELEVÂNCIA DA DISSERTAÇÃO 21

1.5 ESTRUTURA DA DISSERTAÇÃO 21

2 DOMÍNIO DE APLICAÇÃO - DADOS MULTIDIMENSIONAIS 23

2.1 DADOS MULTIDIMENSIONAIS 23

2.2 PROCESSAMENTO ANALÍTICO DE DADOS MULTIDIMENSIONAIS 24

2.3 DOMÍNIO DO PROJETO PINGER 25

2.3.1 Característica multidimensional dos dados 27

2.3.2 Volume dos dados 28

3 FUNDAMENTOS DA WEB SEMÂNTICA 34

3.1 ONTOLOGIA 34

3.2 RDF 36

3.3 LINGUAGEM SPARQL 37

3.3.1 Padrão Básico de Grafo 38

3.3.2 Filtros 38

3.3.3 Mapeamentos opcionais 38

3.3.4 Combinando mapeamentos 39

3.3.5 Modificadores de solução 39

3.3.6 Agrupando mapeamentos 40

3.3.7 Álgebra SPARQL 40

3.4 VOCABULÁRIO RDF DATA CUBE 43

3.5 TRABALHOS RELACIONADOS - PROCESSAMENTO ANALÍTICO DE DADOS MULTIDIMENSIONAIS EM RDF 44

3.5.1 Abordagens para o processamento analítico de dados multidimensionais convertidos para RDF sob demanda 46

3.5.2 Abordagem para o processamento analítico de dados RDF convertidos para o modelo multidimensional 47

3.5.3 Abordagens para o processamento analítico direto de conjuntos de dados RDF 47

3.5.4 Abordagens para o processamento analítico de dados multidimensionais armazenados em sistemas de gestão de dados RDF que envolvem bancos relacionais 49
3.5.5 Abordagens para o processamento analítico de conjuntos de dados RDF armazenados em bancos de dados distribuídos 50
4 TECNOLOGIAS DE DISTRIBUIÇÃO 53
4.1 SISTEMAS DISTRIBUÍDOS 53
4.2 APACHE HADOOP 54
4.2.1 HDFS 54
4.2.2 Hadoop MapReduce 56
4.3 CLOUDERA IMPALA 58
4.3.1 Arquitetura do Cloudera Impala 59
4.3.2 Formatos de arquivos e algoritmos de compressão adotados pelo Cloudera Impala 60
4.3.3 Melhoria do desempenho das consultas executadas no Cloudera Impala . 62
4.4 *CLUSTER* MONTADO 63
5 PROPOSTA DE SOLUÇÃO 66
5.1 VISÃO GERAL DA ARQUITETURA DA APLICAÇÃO *IMPALA MULTILOD* 66
5.2 1º COMPONENTE (CARGA E PREPARAÇÃO DO BANCO DE DADOS) 67
5.2.1 Escolha do esquema de armazenamento de triplas RDF em tabelas 68
5.2.2 Mapeamento dos dados RDF em tabelas a partir de algoritmos MapReduce 75
5.2.3 Criação e carga de tabelas no Impala 79
5.3 2º COMPONENTE (TRADUTOR DE LINGUAGEM SPARQL PARA SQL) 81
5.3.1 Processo de tradução dos operadores da álgebra SPARQL para SQL 83
5.3.1.1 Operador BGP (Padrão Básico de Grafo) 83
5.3.1.2 Operador Join 84
5.3.1.3 Operador LeftJoin 85
5.3.1.4 Operador Union 85
5.3.1.5 Operador Group 86
5.3.1.6 Operador Filter 87
5.3.1.7 Operador Project 88

5.3.1.8 Operador Order 88
5.3.1.9 Operador Slice 89
5.3.1.10 Operador Distinct 89
5.3.2 Exemplo de tradução de consulta SPARQL para SQL 89
5.3.3 Interface gráfica criada para o tradutor SPARQL para SQL 93
6 EXPERIMENTO 95
6.1 ESTUDO DE CASO COM DADOS DO PROJETO PINGER 95
6.2 ETAPA DE PRÉ-PROCESSAMENTO 96
6.2.1 Nova versão da Ontologia PingER baseada no Vocabulário RDF Data Cube 96
6.2.2 Conjuntos de dados do PingER selecionados para testar a aplicação 102
6.2.3 Triplificação 104
6.2.3.1 Triplificação dos dados gerais 105
6.2.3.2 Triplificação dos dados de medidas de rede 107
6.2.4 Carga dos arquivos N-Triples no HDFS 110
6.3 TESTES DE DESEMPENHO EXECUTADOS SOBRE A APLICAÇÃO *IMPALA MULTILOD* 111
6.3.1 Teste do desempenho do 1º componente da aplicação (Carga e Preparação do Banco de Dados) 113
6.3.1.1 Tamanho em disco ocupado pelas tabelas 115
6.3.1.2 Tempos de criação, inserção de dados e cálculo de estatísticas 117
6.3.1.3 Tempo de execução do conjunto de consultas analíticas SQL 118
6.3.1.4 Considerações gerais 122
6.3.2 Teste do desempenho do 2º componente da aplicação (Tradutor de Linguagem SPARQL para SQL) 122
6.3.2.1 Tempo de execução do conjunto de consultas analíticas SPARQL 123
6.3.2.2 Corretude das respostas às consultas traduzidas 125
6.3.2.3 Considerações gerais 126
6.4 COMPARAÇÃO COM O VIRTUOSO TRIPLE STORE 127
6.4.1 Tempo para carga dos conjuntos de dados 128
6.4.2 Tamanho em disco ocupado pelo banco de dados 129
6.4.3 Tempo de execução do conjunto de consultas analíticas SPARQL 130

6.4.4 Considerações gerais.....132
7 CONCLUSÃO.....133
7.1 CONTRIBUIÇÕES.....135
7.2 LIMITAÇÕES.....136
7.3 TRABALHOS FUTUROS.....136
REFERÊNCIAS.....138
Apêndice A - Mapeamento dos prefixos utilizados na ontologia do domínio do Projeto PingER.....144
Apêndice B - Explicação das métricas de rede utilizados no estudo de caso do Projeto PingER.....145
Apêndice C - Conjunto de consultas SQL elaborado para testar os dados do PingER.146
Apêndice D - Conjunto de consultas SPARQL elaborado para testar os dados do PingER.....150
Apêndice E - Consulta Q3 do conjunto de consultas SPARQL traduzida para SQL a partir do tradutor.....154

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Finkelstein e Leaning (1984) definem medição como o processo de atribuição de números ou símbolos a propriedades de entidades do mundo real, de acordo com regras amplamente definidas, a fim de descrevê-las. Medição também pode ser entendida como um processo que envolve um conjunto de ações com a finalidade de caracterizar entidades a partir da atribuição de valores às suas propriedades (BARCELLOS, FALBO, FRAUCHES, 2014). Pode ser citado como exemplo de processo de medição a obtenção do índice de massa corporal de um ser humano, que é medido dividindo o peso (em quilogramas) pela altura (em metros) ao quadrado. O valor obtido a partir da medição pode ser utilizado para analisar se a pessoa está dentro da faixa de peso ideal ou não.

O produto de um processo de medição são medidas, que representam uma importante fonte de informação, desempenhando um papel fundamental no provimento de informações que embasam tomadas de decisões em diversas áreas. Dentro do domínio governamental, medidas permitem analisar a situação da população, economia e mercado, servindo como base para decisões políticas. Na área científica, a manipulação de grandes quantidades de medidas permite a observação de fenômenos e a verificação ou refutação de teorias científicas. Nos setores empresariais, medições são efetuadas para verificar o desempenho do negócio, influenciando a gestão estratégica e a tomada de decisão.

Medidas podem ser representadas a partir de um modelo multidimensional (KIMBALL, ROSS, 2002). Neste modelo o valor medido é comumente visualizado numa posição central, dentro de uma tabela fato, e os parâmetros que descrevem o valor medido em posições satélites, dentro de tabelas de dimensão (tempo, espaço, entre outras). Dentro das dimensões, os dados podem ser organizados em uma hierarquia que define diferentes níveis de detalhamento.

O tipo de processamento frequentemente adotado em conjuntos de dados multidimensionais é o Processamento Análitico On-line (também referenciado como OLAP - *On-line Analytical Process*) (THOMSEN, 1997). Consultas OLAP possuem o enfoque analítico e permitem a visualização dos dados através de múltiplos níveis de hierarquias e diferentes perspectivas a partir do uso de operadores dimensionais (BARBIERI, 2011). Possuem as características de serem complexas, de longa duração e, geralmente, apresentam alto custo de processamento porque envolvem o acesso aos dados em diferentes níveis de granularidade, realizam múltiplos agrupamentos, agregações e cruzamentos de dados.

Com a disseminação de técnicas de Web Semântica e a grande ênfase em abertura de dados (especialmente os governamentais), muitos conjuntos de dados com características multidimensionais estão sendo disponibilizados seguindo os princípios de Dados Abertos Interligados (*Linked Open Data*, na sigla em inglês: LOD). Exemplos de tal iniciativa incluem, no Brasil, o Portal Brasileiro de Dados Abertos[1] e, na Europa, o UK Open Government Data[2].

Dentro da Web Semântica, o termo Dados Abertos Ligados (BERNERS-LEE, 2009) é utilizado para descrever um conjunto de melhores práticas para publicar, compartilhar e interligar dados estruturados disponíveis na Web. A fim de permitir a publicação de dados com semântica na Web uma série de padrões e protocolos foram definidos, tais como: o padrão *Resource Description Framework*[3] (RDF), utilizado para descrever e codificar dados na forma de triplas, declarações do tipo (sujeito, predicado, objeto), que juntas formam um grafo RDF e o protocolo de comunicação *SPARQL Protocol and RDF Query Language*[4] (SPARQL), que pode ser utilizado para expressar consultas sobre diversas fontes de dados armazenadas nativamente como RDF ou vistas como RDF via *middleware*.

Conforme citado em (CYGANIAK, REYNOLDS, TENNISON, 2014), diversos benefícios podem ser obtidos a partir da publicação de dados multidimensionais seguindo os princípios do modelo LOD, tais como:

- Agregação de semântica que falta aos dados, se valendo para isso de ontologias e vocabulários.
- Medidas ou grupos de medidas se tornam endereçáveis na Web, através de *Uniform Resource Identifier* (URIs), cadeias de caracteres utilizadas para identificar recursos na Web. Isso permite aos publicadores e terceiros anotar e se conectar a estes dados.
- Simplificação do processo de ligação e agregação de fontes relacionadas na Web, criando um espaço de dados interligados que permite uma rica análise do conjunto.
- Favorecimento da reutilização dos dados.
- Dados passam a ser publicados em um formato não-proprietário, flexível e capaz de ser lido e processado por máquinas.

[1] http://dados.gov.br/

[2] http://data.gov.uk/data/search

[3] https://www.w3.org/standards/techs/rdf#w3c_all

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

Atualmente, no âmbito da Web Semântica, o vocabulário RDF Data Cube (CYGANIAK, REYNOLDS, TENNISON, 2014) vem sendo amplamente utilizado na publicação de dados com características multidimensionais na Web. Os dados são representados utilizando o padrão RDF e publicados seguindo os princípios de dados ligados. O vocabulário propõe a estruturação dos dados no formato RDF de maneira que fiquem explícitos os aspectos de multidimensionalidade e as perspectivas de análise do domínio tratado, facilitando a criação de consultas analíticas sobre os dados convertidos. O RDF Data Cube é baseado no padrão *Statistical Data and Metadata Exchange*[5] (SMDX), padrão para coleta, troca, processamento e divulgação de informações estatísticas.

1.2 QUALIFICAÇÃO DO PROBLEMA

Apesar dos diversos benefícios citados anteriormente, a publicação de dados seguindo os princípios do modelo LOD pode causar problemas de desempenho e escalabilidade. A representação dos dados na forma de triplas RDF gera uma estrutura subjacente de grafo para os dados, a qual é difícil de ser processada, consultada e agregada. Estas questões se tornam cada vez mais iminentes quando tecnologias tradicionais na Web Semântica são utilizadas em aplicações do mundo real, que geralmente lidam com conjuntos muito volumosos de dados RDF e realizam análises complexas sobre esses dados, que suportam tomadas de decisão. Este é normalmente o caso de aplicações que lidam com conjuntos de dados multidimensionais e realizam o seu processamento analítico, com o objetivo de atender necessidades analíticas do negócio.

Dados multidimensionais possuem a característica de serem muito volumosos, uma vez que tendem a ser produzidos em grandes quantidades em intervalos de tempo curtos. Podem ser citados como exemplo conjuntos de dados pluviométricos, que armazenam medidas da quantidade de precipitações de água (chuva, neve, granizo) em um determinado local em um dado período de tempo. Tais medidas, frequentemente, são capturadas automaticamente através de sensores em intervalos de tempo curtos, gerando grandes volumes de dados.

Além disso, dados multidimensionais possuem a peculiaridade de serem intrinsecamente estruturados (PALAZZI, TYGEL, 2014). Isso quer dizer que cada medida atende a um conjunto de dimensões e atributos, e é justamente isso que permite fazer observações agregadas e obter dados consolidados. Esta peculiaridade faz com que a

[5] https://sdmx.org/

representação de dados multidimensionais em triplas acarrete uma enorme redundância de informações, o que leva a geração de conjuntos de dados multidimensionais no formato RDF extremamente volumosos, com milhões ou até bilhões de triplas.

Embora já existam vocabulários e padrões, tais como o vocabulário RDF Data Cube e o padrão SMDX, que auxiliam a transformação e representação de bases de dados multidimensionais, de acordo com o paradigma OLAP (THOMSEN, 1997), no formato de triplas RDF, ainda existem poucos trabalhos na literatura que exploram o processamento analítico de grandes volumes de dados multidimensionais no formato RDF e aspectos ligados a medição e melhoria de seu desempenho.

O estudo realizado em (FUNDULAKI et al., 2013) avaliou *benchmarks* existentes que permitem realizar a medição do desempenho de sistemas de gestão de dados RDF, tais como o *Berlin SPARQL Benchmark* (BSBM) (BIZER, SCHULTZ, 2009), *SPARQL Performance Benchmark* (SP2Bench) (SCHMIDT et al., 2009) e *Lehigh University Benchmark* (LUBM) (GUO, PAN, HEFLIN, 2005). O resultado do estudo aponta que a maioria dos *benchmarks* RDF avaliados não cobrem os recursos encontrados em *benchmarks* analíticos relacionais, tais como os *benchmarks* de apoio à decisão TPC-H e TPC-DS (COUNCIL, 2005), que consistem em um conjunto de consultas OLAP de alto grau de complexidade executadas para medir o desempenho de sistemas de bancos de dados relacionais. A maioria dos *benchmarks* RDF avaliados são baseado em conjuntos de dados transacionais e executam consultas que realizam processamentos simples, envolvendo um número pequeno de triplas e junções, não empregando extensivamente operações SPARQL complexas, tais como agregação, agrupamento e combinações.

Como alternativa alguns estudos realizaram a tradução do *benchmark* TPC-H para o formato RDF com o objetivo de testar a capacidade de processamento analítico de sistemas de gestão de dados RDF. Dentre os estudos encontrados estão (ERLING; MIKHAILOV, 2009), no qual foi proposto um mapeamento do esquema do *benchmark* TPC-H para RDF e de suas consultas na linguagem SQL para SPARQL e (ERLING, 2012) que utilizou o mapeamento proposto em (ERLING, MIKHAILOV, 2009) para medir o desempenho do sistema Virtuoso Triple Store.

Diante do cenário exposto chega-se à conclusão que ainda há espaço para elaboração de estudos que apresentem abordagens adequadas para a gestão de conjuntos volumosos de dados no formato RDF. E a validação seja feita a partir da execução de consultas sobre conjuntos de dados multidimensionais volumosos, que testem a capacidade de processamento analítico dos dados oferecida a partir da abordagem.

1.3 OBJETIVO

O objetivo geral desta dissertação é propor uma abordagem para apoiar a execução de consultas analíticas SPARQL sobre grandes volumes de dados multidimensionais em formato de grafos RDF. Para isso, foi desenvolvida uma aplicação, chamada *Impala MultiLOD*, que se baseia em tecnologias de distribuição e de armazenamento oferecidas pelo *framework* Apache Hadoop[6] para permitir que consultas analíticas SPARQL sejam executadas de forma distribuída, buscando com isso melhorar o seu desempenho.

O *framework* Apache Hadoop permite o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores, utilizando modelos de programação simples. O Hadoop oferece como ferramentas principais o MapReduce[7], responsável pelo processamento distribuído, e o *Hadoop Distributed File System*[8] (HDFS), responsável pelo armazenamento de grandes conjuntos de dados, também de forma distribuída.

Existem projetos relacionados que utilizam o Hadoop com o objetivo de executar análise de dados em larga escala, tais como Apache Hive[9], Apache Pig[10] e Cloudera Impala[11]. Nesta dissertação a tecnologia utilizada foi o Cloudera Impala, SGBD NoSQL que utiliza armazenamento orientado a colunas e permite a execução de consultas SQL (*Structured Query Language*) interativas em cima de dados armazenados no HDFS.

Apesar das tecnologias escolhidas não serem elaboradas especificamente para trabalhar no contexto da Web Semântica, ainda foi possível adaptá-las, tornando-as capazes de armazenar e processar dados semânticos. Para isso, foram implementados dois componentes na aplicação *Impala MultiLOD*:

- **1º componente (Carga e Preparação do Banco de Dados)**: Responsável por carregar os dados RDF, armazenados no HDFS, em tabelas no Impala.
- **2º componente (Tradutor de Linguagem SPARQL para SQL)**: Responsável por realizar a tradução de consultas SPARQL, geradas pelos usuários finais da aplicação, em consultas SQL equivalentes, executadas diretamente no Impala a partir de conexão via *driver* JDBC. A versão do SPARQL com a qual o tradutor

[6] http://hadoop.apache.org/

[7] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[8] https://hadoop.apache.org/docs/r1.2.1/hdfs_user_guide.html

[9] https://hive.apache.org/

[10] http://hortonworks.com/hadoop/pig/

[11] http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html

trabalha é a 1.1, que oferece suporte a operadores de agregação, habilitando o processamento analítico dos dados.

Os experimentos realizados na aplicação foram feitos a partir de um estudo de caso que utilizou dados de medições reais originados do projeto *Ping End-to-end Reporting* (PingER) (COTTRELL, 2001), que monitora o desempenho de *links* de Internet ao redor do mundo. O estudo de caso teve por objetivo avaliar se a aplicação proposta nesta dissertação é adequada para a gestão dos conjuntos volumosos de dados no formato RDF do projeto e é capaz de melhorar o desempenho de consultas analíticas SPARQL tipicamente executadas sobre os dados, que dão suporte a tomadas de decisão e baseiam estudos que se relacionam à qualidade de *links* rede ao redor do mundo. Métricas como o tempo de resposta a consultas, tempo levado para carga de conjuntos de dados e quantidade em disco consumida por arquivos de dados foram utilizadas para medir o desempenho da aplicação.

Dentro do estudo de caso foi realizada uma comparação dos resultados de desempenho obtidos a partir da avaliação da arquitetura proposta nesta dissertação com os do sistema Virtuoso Triple Store[12], que apresentou os menores tempos de respostas na execução das consultas do BSBM dentre os repositórios de triplas avaliados no estudo realizado em (BIZER, SCHULTZ, 2009). O sistema foi avaliado também de forma distribuída no mesmo ambiente computacional em que foi testada a aplicação *Impala MultiLOD*, utilizando os mesmos conjuntos de dados, a fim de permitir uma comparação justa.

Além do objetivo geral, também foram estabelecidos objetivos específicos para esta dissertação:

- Descrição dos dados do projeto PingER como triplas RDF utilizando o vocabulário RDF Data Cube, explicitando, com isso, as perspectivas de análise do domínio, facilitando a construção de consultas analíticas sobre os dados publicados no formato RDF.
- Construção, gerência e manutenção de um *cluster* físico onde foram executados os serviços do *framework* Apache Hadoop e rodados os experimentos na aplicação Impala *MultiLOD*.
- Estudo da sintaxe e álgebra das linguagens SPARQL e SQL e implementação de uma lógica de tradução de consultas SPARQL para SQL.

[12] http://virtuoso.openlinksw.com/

1.4 RELEVÂNCIA DA DISSERTAÇÃO

A linguagem SPARQL só começou a dar suporte a operadores de agregação a partir da versão 1.1, recomendação W3C de 21 Março de 2013 (HARRIS, PRUD'HOMMEAUX, SEABORNE, 2013). Logo, esta dissertação é relevante uma vez que ainda são poucos os estudos que se propõem a investigar e melhorar o desempenho de consultas analíticas SPARQL, que seguem a versão 1.1 da linguagem, executadas sobre aplicações do mundo real, que geralmente lidam com conjuntos muito volumosos de dados RDF.

O RDF promete ser uma forma de representação de alto nível para dados extraídos a partir de qualquer fonte concebível. A principal promessa do RDF é no campo da integração de informações, análise e descoberta. Logo, a principal contribuição esperada para esta dissertação é cooperar para que a linguagem SPARQL alcance melhores resultados de desempenho ao responder consultas analíticas complexas, como as consultas que são efetuadas sobre *data warehouses*.

1.5 ESTRUTURA DA DISSERTAÇÃO

Quanto à estrutura desta dissertação, no **Capítulo 2** discutimos conceitos que envolvem o processamento analítico de dados multidimensionais. Além disso, é feita uma análise do domínio do projeto PingER, domínio de aplicação escolhido para verificação e experimentação da abordagem de gestão de dados multidimensionais no formato RDF proposta nesta dissertação. No **Capítulo 3** é feita uma breve revisão dos fundamentos da Web Semântica, focando na revisão dos conceitos que foram aplicados durante o desenvolvimento desta dissertação. No final do capítulo são discutidos trabalhos relacionados que propuseram abordagens para o processamento analítico de dados multidimensionais em RDF descritos utilizando o vocabulário RDF Data Cube ou vocabulários relacionados. No **Capítulo 4** são introduzidas as tecnologias de distribuição de processamento e armazenamento de dados que foram empregadas na aplicação *Impala MultiLOD*. No **Capítulo 5** é apresentada a proposta desta dissertação, sendo explicada a arquitetura da aplicação desenvolvida e o funcionamento de cada um dos seus componentes. No **Capítulo 6** são detalhados os experimentos realizados dentro do estudo de caso que testou o desempenho da aplicação utilizando conjuntos de dados de tamanhos diferentes provenientes do projeto PingER. Os resultados de desempenho obtidos foram comparados com os do sistema Virtuoso Triple Store, carregado e testado com os mesmos conjuntos de dados. Finalmente, no **Capítulo 7** é apresentada a conclusão do trabalho, sintetizando os principais pontos tratados nos capítulos anteriores e resumindo as

contribuições proporcionadas. Também são listados trabalhos futuros relacionados tanto ao aperfeiçoamento da aplicação *Impala MultiLOD* quanto a novas possibilidades de experimentos que poderiam ser realizados.

2 DOMÍNIO DE APLICAÇÃO - DADOS MULTIDIMENSIONAIS

Neste capítulo, nas Seções 2.1 e 2.2 são revisados os conceitos de dados multidimensionais e processamento analítico que foram aplicados durante o desenvolvimento desta dissertação. Na Seção 2.3 é feita uma análise do domínio do projeto PingER, domínio de aplicação escolhido para verificação e experimentação da abordagem de gestão de conjuntos de dados multidimensionais volumosos no formato RDF proposta nesta dissertação.

2.1 DADOS MULTIDIMENSIONAIS

De acordo com (CYGANIAK, REYNOLDS, TENNISON, 2014), dados com características multidimensionais, tais como dados oriundos de processos de medições e dados estatísticos, compreendem uma coleção de observações (medidas) feitas em alguns pontos através de um espaço lógico. Esta coleção pode ser caracterizada por um conjunto de dimensões (por ex., tempo, espaço, população) que identificam a observação, juntamente com metadados que descrevem o que foi medido (por ex., atividade), como foi medido e como as observações são expressas (por ex., unidade de medida).

Conjuntos de dados com características multidimensionais são muitas vezes representados através de uma estrutura multidimensional, conhecida como cubo de dados. Tal estrutura é organizada de acordo com um conjunto de componentes: dimensões, atributos e medidas (KIMBALL, ROSS, 2002).

Os componentes de dimensões servem para identificar as observações. Um conjunto de valores para todos os componentes de dimensão é suficiente para identificar uma única observação. Cada dimensão é descrita por um conjunto de atributos e, frequentemente, estes atributos são relacionados a partir de hierarquias de níveis, que permitem a análise dos dados em níveis diferentes de detalhes. No exemplo da Figura 1, a dimensão Cidade pode ser organizada em uma hierarquia Cidade-Estado-País (CHAUDHURI, DAYAL, 1997).

Os componentes da medida representam o fenômeno a ser observado (representam fatos), tais como o Valor da Venda no exemplo da Figura 1. E os componentes de atributos permitem qualificar e interpretar o valor observado ou valores observados. Dessa forma, permitem a especificação das unidades de medidas, quaisquer fatores de escala e metadados, como o estado de observação.

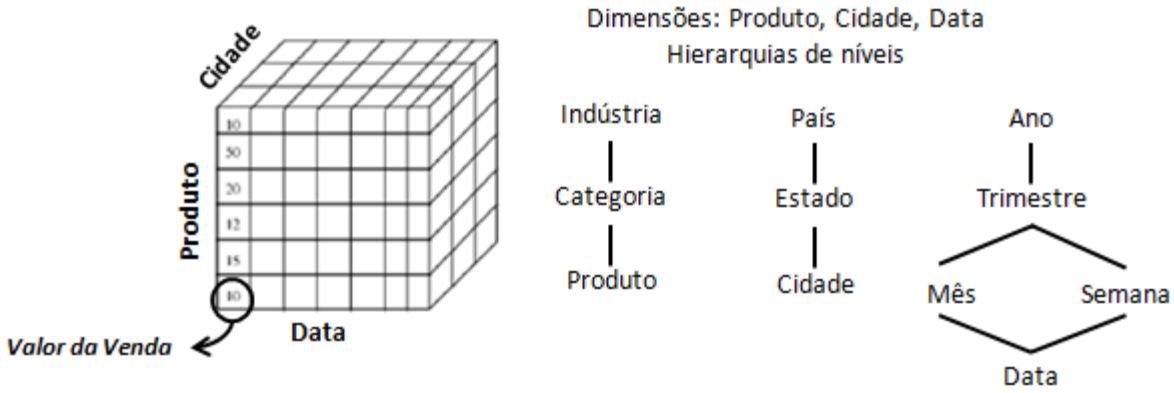

Figura 1: Cubo de dados (Adaptado de CHAUDHURI, DAYAL, 1997)

Um cubo é representado em um banco de dados relacional como um conjunto de tabelas organizadas na forma de um esquema estrela ou floco de neve. O esquema estrela, proposto por Kimball (KIMBALL, ROSS, 2002), consiste de uma única tabela fato central ligada a um certo número de tabelas de dimensão não normalizadas. O esquema floco de neve aplica um refinamento do esquema estrela, onde a hierarquia dimensional é explicitamente representada através da normalização das tabelas de dimensão.

2.2 PROCESSAMENTO ANALÍTICO DE DADOS MULTIDIMENSIONAIS

A estruturação adequada de dados com características multidimensionais na forma de cubos de dados indexados por dimensões viabiliza a sua análise e manipulação. O Processamento OLAP refere-se à capacidade de manipular e analisar grandes volumes de dados multidimensionais através de múltiplos níveis de hierarquias e diferentes perspectivas, onde, para isso, os dados devem estar organizados a partir de uma estrutura multidimensional (CHAUDHURI, DAYAL, 1997).

O Processamento OLAP se caracteriza por operações que permitem a agregação e combinação de medidas de diferentes cubos de dados de acordo com valores comuns de suas dimensões. O resultado de tais operações geram novas informações que podem apoiar a tomada de decisão em diferentes domínios. Exemplos de operações OLAP que são tipicamente efetuadas sobre cubos de dados são (KIMBALL, ROSS, 2002):

- *Drill Down*: Navegação ao longo dos níveis de uma hierarquia de uma dimensão em busca de informações mais detalhadas, em níveis menores.
- *Roll-Up*: Navegação ao longo dos níveis de uma hierarquia de uma dimensão em busca de informações mais consolidadas, em níveis maiores.
- *Slice*: Seleção que produz um subcubo com uma dimensão a menos, escolhendo um único valor para uma de suas dimensões.

- *Dice*: Seleção que produz um subcubo, permitindo um filtro para valores específicos de múltiplas dimensões. A operação *dice* é uma operação *slice* em mais de duas dimensões de um cubo de dados.

Aplicações OLAP classificam-se de acordo com a forma que armazenam os dados, em:

- MOLAP: Armazena os dados em formato multidimensional.
- ROLAP: Armazena os dados em formato relacional.
- HOLAP: Forma híbrida.

A arquitetura ROLAP é uma simulação da tecnologia OLAP feita em banco de dados relacionais que, por utilizar a estrutura relacional, possui a vantagem de não restringir o volume de armazenamento de dados. Esta ferramenta não utiliza cubos pré-calculados como a MOLAP. À medida que o usuário monta sua consulta em uma interface gráfica, a ferramenta acessa metadados ou quaisquer outros recursos que possua, para gerar uma consulta SQL sobre tabelas fato e de dimensão organizadas na forma de um esquema estrela ou floco de neve (THOMSEN, 1997).

Consultas OLAP podem ser formuladas em SQL ou SPARQL, utilizando a versão 1.1 da linguagem, através do uso de operadores de agregação, agrupamentos, filtros e modificadores de solução (tais como *order by*, *limit* e *offset*) (ORACLE, 2003). O tradutor de consultas SPARQL para SQL construído nesta dissertação realiza a tradução de consultas OLAP formuladas, por usuários finais, na linguagem SPARQL em consultas equivalentes na linguagem SQL, que após serem geradas são executadas no Impala.

2.3 DOMÍNIO DO PROJETO PINGER

Dados resultantes de processos de medição são exemplos típicos de dados multidimensionais, pois os valores de suas medidas são, normalmente, descritos a partir de diferentes dimensões e em diferentes níveis de detalhe. Como são dados que possuem diferentes perspectivas de análise, são passíveis de processamento analítico.

O projeto PingER (COTTRELL, 2001) armazena conjuntos de dados reais oriundos de processos de medição de qualidade de *links* de Internet, desde 1998, utilizando uma granularidade horária e diária e 16 métricas de rede diferentes. A listagem e a definição de todas estas métricas de rede podem ser encontradas no trabalho de (SOUZA et al., 2014). O projeto descreve medições de cerca de 80 nós monitores para mais de 800 nós monitorados

(mais de 8000 pares de nós - nem todos os nós monitores monitoram todos os nós monitorados), em mais de 160 países.

O projeto foi desenvolvido por uma equipe de colaboradores de universidades e laboratórios da América do Norte, Europa, Paquistão e Malásia. Mas, é concentrado principalmente no *Stanford National Accelerator Laboratory*[13] (SLAC), que é operado pela Universidade de Stanford.

Todos os dados do SLAC estão disponíveis e podem ser acessados através da ferramenta *Pingtable*[14], por onde é possível consultar e baixar arquivos em formato CSV selecionando apenas alguns parâmetros a serem utilizados na recuperação dos dados desejados. Uma consulta no *Pingtable* passa pela seleção dos seguintes parâmetros:

- *Metric*: Seleção de uma dentre 16 métricas de rede.
- *Tick-Type*: Granularidades de tempo:
  - *hourly*: as 24 horas (0 - 23) de um dia específico
  - *daily*: os dias de um determinado mês
  - *last60days*: os últimos 60 dias a contar da data de ontem
  - *last120days*: os últimos 120 dias a contar da data de ontem
  - *last360days*: os últimos 360 dias a contar da data de ontem
  - *allmonthy*: todos os meses, do mês atual até fevereiro/98
  - *monthly*: o último ano
  - *allyearly*: todos os anos (1998-atual)
- *Packet-Size* (bytes): Seleciona pacotes de 100 ou 1000 bytes.
- *From*: Exibe uma lista de todos os nomes dos nós de cada nó monitor.
- *To*: Exibe uma lista de todos os nomes dos nós de cada nó monitorado. Sendo que cada nó monitor é também um nó monitorado.
- *By* (node ou site): Forma de agregar regionalmente os resultados.
- *Excluding*: Exclusão dos resultados referentes a um determinado nó.

Como cada arquivo gerado a partir da ferramenta *Pingtable* corresponde a medidas relacionadas aos parâmetros que foram selecionados, o acesso a um conjunto muito volumoso de dados se torna extremamente lento. Logo, o laboratório SLAC passou a disponibilizar todos os dados de forma compactada a partir de um endereço de FTP[15]. Neste endereço o

[13] https://www6.slac.stanford.edu/

[14] http://www-wanmon.slac.stanford.edu/cgi-wrap/pingtable.pl

[15] FTP público de Cottrell no SLAC: ftp://ftp.slac.stanford.edu/users/cottrell/

volume total de arquivos CSV do SLAC ficam organizados a partir de uma série de arquivos compactados com extensão .TAR, separados por métrica e por tamanho de pacote. Cada arquivo .TAR inclui todos os arquivos relativos a uma métrica, em todas as agregações temporais (*Tick-Type*), dentro do período de 1998 ao ano corrente.

2.3.1 Característica multidimensional dos dados

Os dados do projeto PingER podem ser caracterizados como dados multidimensionais devido as seguintes características:

- Dados oriundos de processos de medição.
- Diferentes parâmetros (tempo, região, tipo de métrica de rede, etc.) descrevem as medições realizadas.
- Dados descritos em diferentes níveis de agregação.
- Dados históricos acumulados ao longo de anos.
- Dados passíveis de manipulação OLAP.
- Baseiam estudos que se relacionam à qualidade de rede e dão suporte à tomada de decisão.

Na Figura 2, o modelo conceitual do projeto PingER é representado a partir de um esquema de floco de neve, com o objetivo de representar explicitamente as hierarquias dimensionais através da normalização das tabelas de dimensão. No caso do PingER, as dimensões centrais (retratadas na cor verde na Figura 2) que identificam o valor medido são Hora (representa o momento em que a medição foi realizada), Nó de rede (representa onde a medição foi realizada, assumindo dois papéis diferentes - Nó monitor e Nó monitorado) e Métrica de Rede (representa o que foi medido). As dimensões auxiliares (retratadas na cor azul na Figura 2), que representam explicitamente as hierarquias e permitem a análise dos dados em níveis diferentes de agregação são Data, Mês e Ano (compõem a hierarquia de tempo); Cidade, Estado, País e Continente (compõem a hierarquia de espaço e identificam a localização regional de um nó de rede) e Instituição de Ensino (identifica a Instituição de Ensino em que o nó de rede está localizado).

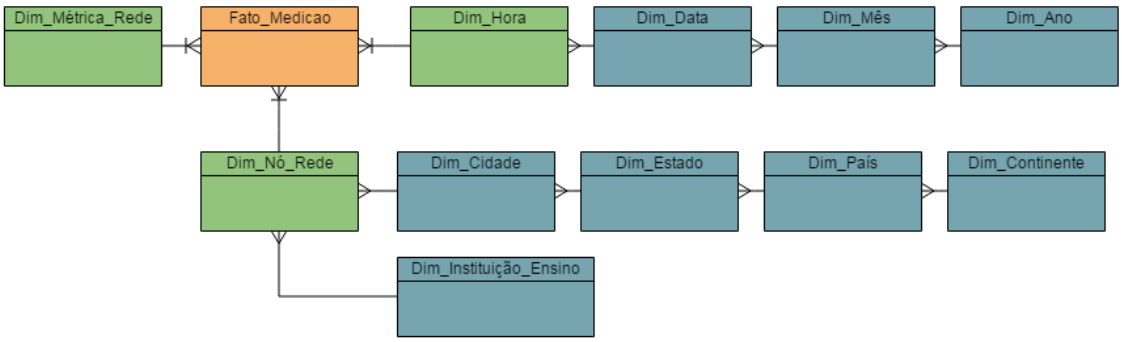

Figura 2: Modelo conceitual do projeto PingER baseado em um esquema floco de neve

2.3.2 Volume dos dados

No projeto MultiLOD (2015) desenvolvido pelo Centro de Referência em Big Data da UFRJ (CRBD/UFRJ) foi realizada uma estimativa do volume dos dados do projeto PingER. Para isso foram considerados arquivos de dados do dia 01/01/1998 até o dia 10/09/2014, de todas as 16 métricas de rede, com agregação por nó, tamanho de pacote de 100 bytes e granularidade temporal de hora. Todos os arquivos foram obtidos a partir do FTP público do SLAC.

A Figura 3 ilustra parte da estrutura de um arquivo CSV que armazena as medidas coletadas para a métrica de rede Average RTT no dia 10/09/2014 utilizando granularidade de hora e pacotes de 100 bytes de dados. O cabeçalho rotula as colunas (com a hora do dia) que começam após os nomes dos nós monitores e monitorados (por exemplo, na linha 2 os nós são pingerkhi.pern.edu.pk e pingerpwr.pern.edu.pk). Seguindo os dois nomes são ilustrados os valores da métrica para as horas correspondentes. Um ponto (.) significa que foi coletada uma medida nula. O nome do arquivo identifica a métrica sendo medida, o tipo de pacote de dados e a granularidade regional e temporal adotada, no caso o nome do arquivo ilustrado na Figura 3 é "average_rtt-100-by-node-2014-09-10".

		0	1	2	3	4	5	6	7	8	9	10	11
pingerkhi.pern.edu.pk	pingerpwr.pern.edu.pk	.	.	.	.	.	.	.	.	.	.	.	.
pinger.sprace.org.br	www.kist.ac.rw	.	.	.	.	.	.	.	.	.	.	.	.
airuniversity.seecs.edu.pk	mail.gnet.tn	141.522	141.515	141.500	141.720	144.745	141.918	144.594	142.411	142.123	142.355	142.495	141.863
pingerlhr-gcu.pern.edu.pk	www.univ-koudougou.bf	291.699	287.438	2.114.004	1.236.587	291.677	308.649	782.475	1.231.689	1.713.549	1.406.652	1.165.659	430.299
pinger.stanford.edu	ping.rmki.kfki.hu	215.995	287.775	263.791	208.113	200.410	202.197	198.406	199.815	199.686	201.898	201.225	200.255
netmon.physics.carleton.ca	www.puc.cl	157.500	158.000	158.000	162.000	161.500	158.500	157.500	158.500	157.500	158.500	158.000	159.500
pingerjms.pern.edu.pk	www.lbr.gov.lr	.	.	.	.	.	.	.	849.364	935.599	903.839	958.453	907.845
pinger2.unimas.my	www.unila.ac.id	104.765	105.904	136.722	121.724	106.672	108.928	108.358	108.531	109.131	107.391	123.543	109.126
pinger.uettaxila.edu.pk	www.zcuniversity.edu.zm	374.269	374.269	446.753	413.952	413.952	390.123	375.007	428.469	365.491	387.341	372.923	372.923
pinger.uettaxila.edu.pk	www.college.edu.sn	.	.	.	.	.	.	.	.	.	.	.	.
pingerkhi-cpsp.pern.edu.pk	magic.mn	326.374	326.435	326.010	326.967	326.637	376.625	427.080	427.922	438.685	343.901	351.638	343.111
pinger.uettaxila.edu.pk	pinger.comsats.edu.pk	.	.	.	.	.	.	.	.	.	.	.	.
pinger.nwfpuet.edu.pk	duhs.seecs.edu.pk	.	.	.	.	.	.	.	.	.	.	.	.

Figura 3: Estrutura do arquivo SLAC - Métrica Average RTT - Dia 10/09/2014 - Pacote de dados 100 bytes

Um código JAVA foi gerado para percorrer cada um dos arquivos obtidos recolhendo informações relevantes como: a métrica considerada, a data e o tamanho do arquivo. Estas informações foram armazenadas em um arquivo CSV. Para algumas combinações de dia, mês e ano, nenhum arquivo foi gerado; sendo atribuído um valor nulo para o campo correspondente ao tamanho do arquivo. Para outras, o arquivo gerado tem apenas o cabeçalho com as horas daquele dia, sem nenhuma medida. Este tipo de arquivo possui tamanho de 63 bytes.

A partir do arquivo CSV gerado, foi feita uma análise que permitiu concluir que, no período considerado, foram gerados 96688 arquivos, somando um volume de aproximadamente 64 GB. Caso existissem arquivos de medidas com granularidade hora para todas as datas do intervalo considerado haveriam 6209 arquivos CSV para cada métrica. Ou seja, haveria um total de 99344 arquivos.

Entretanto, 2656 desses arquivos não foram gerados por motivos como os dados brutos não ofereceram granularidade suficiente para fornecer medidas para certas métricas. Não é possível especificar ao certo, para cada arquivo inexistente, o motivo que levou a sua inexistência. Sabemos apenas que o arquivo para a métrica e a data correspondentes não existe. Para os casos em que o arquivo existe mas contém apenas o cabeçalho, sem os dados de medidas, pode ter ocorrido que o nó destino não estivesse acessível no dia determinado. Isto é, seu endereço IP pode ter mudado, *pings* podem ter sido bloqueados, etc.

Na Tabela 1 é possível visualizar a quantidade total de arquivos e seu volume ocupado em disco para cada métrica de rede considerada.

Tabela 1: Quantidade total de arquivos referentes a 16 métricas de rede, granularidade hora, pacote de 100 bytes, período de tempo de 01/01/1998 até 10/09/2014

Métrica	**Quantidade Total de Arquivos**	**Volume, em bytes, do Total de Arquivos**	**Volume aproximado em *GigaBytes***
Alpha	6028	4793721046	4,5
Average_rtt	6048	4972484812	4,6
Conditional_loss_probability	6047	2860339536	2,6
Duplicate_packets	6048	4023795715	3,7
Ipdv	6048	4097921651	3,8
Iqr	6048	4066288735	3,8
Maximum_rtt	6028	5751898949	5,3
Minimum_packet_loss	6047	4280883632	4
Minimum_rtt	6049	4965122633	4,6
MOS	6028	3683225405	3,4
Out_of_order_packets	6048	3574443521	3,3
Packet_loss	6049	4231619932	3,9
Throughput	6028	5918099435	5,5
Unpredictability	6048	3669110965	3,4
Unreachability	6048	3838587658	3,6
Zero_packet_loss_frequency	6048	3700839289	3,4
TOTAL	**96688**	**68428382914**	**63,7**

No gráfico ilustrado na Figura 4 é possível observar a taxa de crescimento dos dados de granularidade hora entre 1998 e 2014.

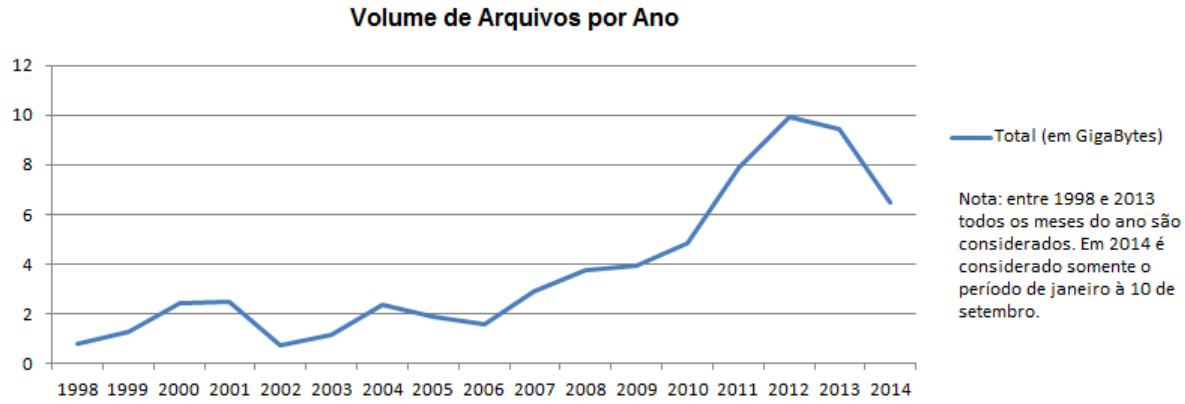

Figura 4: Taxa de crescimento dos arquivos com granularidade hora entre 1998 e 2014

Há casos em que, para uma determinada combinação de parâmetros, é gerado um arquivo com pontos (.) no lugar de uma ou mais medidas, indicando que não há o dado correspondente. Pontos são inesperados, mas podem acontecer pelos mesmos motivos que um arquivo é gerado somente com o cabeçalho. No gráfico ilustrado na Figura 5, é possível visualizar a quantidade de pontos ocorridos por ano. E no gráfico ilustrado na Figura 6 é possível observar para qual métrica temos a maior quantidade de pontos.

Figura 5: Quantidade de pontos ocorridos em arquivos entre 1998 e 2014

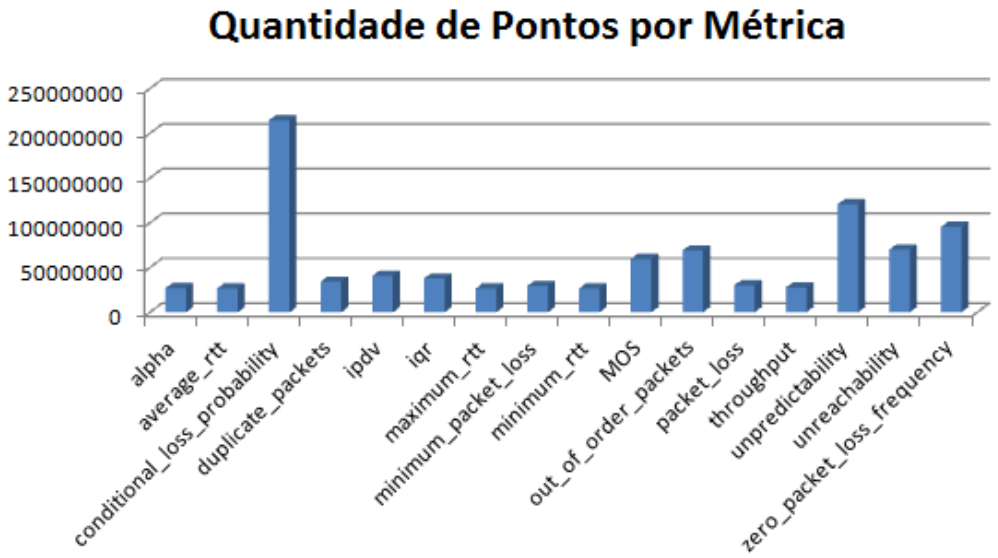

Figura 6: Quantidade de pontos em arquivos por métrica entre 1998 e 2014

Ao considerar o conjunto de todos os dados compactados do PingER gerados entre os dias 01/01/1998 a 20/10/2014, podemos ver que o volume de dados compactados é de, aproximadamente, 39 GB. Neste conjunto foram considerados todas as possíveis combinações de pacote de dados de tamanho de 100 bytes ou 1000 bytes, agrupamento por nó e por região, mais todas as granularidades de tempo (hora, dia, mês, ano, últimos 60 dias, etc.). Na Tabela 2 é possível observar a quantidade e o volume de arquivos compactados por métrica.

A partir da análise das características do conjunto de dados do projeto PingER é possível enquadrá-lo como um problema de *Big Data* por atender as três características típicas deste tipo de problema - Volume, velocidade e variedade (ZIKOPOULOS, EATON, 2011). Volume, pois a análise feita nesta seção mostrou que até outubro de 2014 a sua quantidade de dados compactados já estava na faixa de 39GB, sendo possível que a quantidade atual de dados descompactados já tenha ultrapassado a faixa de muitos *gigabytes* e chegado na faixa de muitos *terabytes*. Velocidade, pois os servidores do PingER recebem dados com uma taxa de *bytes* por intervalo de tempo alta, os *pings* são realizados a cada 30 minutos e as medições são armazenadas em arquivos a cada hora. Variedade, pois os dados do PingER são dados semi-estruturados, que não estão armazenados em um SGBD tradicional e sim através de uma estrutura de arquivos CSV.

Tabela 2: Quantidade total de arquivos compactados referentes a 16 métricas de rede, todas granularidades temporal e regional, pacotes de dados de 100 e 1000 bytes no período de tempo de 01/01/1998 até 20/10/2014

Métrica	**Quantidade de Arquivos Compactados**	**Volume, em bytes, de Arquivos Compactados**	**Volume, em gigabytes, de Arquivos Compactados**
Alpha	18302	2014067586	1,88
average_rtt	25473	5934539571	5,53
conditional_loss_probability	25455	1104514719	1,03
duplicate_packets	25455	941305650	0,88
Ipdv	28707	3849609610	3,59
Iqr	23761	2669757092	2,49
maximum_rtt	18309	5171201738	4,82
minimum_packet_loss	25465	1085177479	1,01
minimum_rtt	25480	5260948517	4,90
MOS	18296	1388697374	1,29
out_of_order_packets	25452	884536677	0,82
packet_loss	25404	1608191173	1,50
Throughput	16850	5277999763	4,92
Unpredictability	25464	2175013357	2,03
Unreachability	25455	1001740993	0,93
zero_packet_loss_frequency	25451	1043947454	0,97
TOTAL	**378779**	**41411248753**	**38,57**

3 FUNDAMENTOS DA WEB SEMÂNTICA

A Web Semântica (BERNERS-LEE, 1998) vem abordar o problema de que o conteúdo da Web consiste, em grande parte, de documentos não estruturados, apenas compreensível por humanos e não processável por máquinas. Logo, a sua proposta inicial foi estender os paradigmas da Web de documentos para uma Web de Dados, na qual os documentos apresentam anotações legíveis por máquinas, permitindo-as entender o significado dos dados.

Para isso, foram criados padrões de formatos e protocolos, assim como os existentes para HTML, CSS, etc., para disponibilização e ligação de dados e metadados. Muitos destes padrões são recomendados pela *World Wide Web Consortium* (W3C), organização fundada em 1994 para fomentar padrões e tecnologias interoperáveis, através da publicação de guias, softwares e ferramentas para a Web.

Dentro da Web Semântica, o projeto LOD (BERNERS-LEE, 2009), tem como objetivo promover a interligação de dados que estão gratuitamente disponíveis. O diagrama da nuvem de LOD[16] é uma aplicação que desenha um grafo com os bancos de dados em RDF mais expressivos existentes. A versão mais recente disponível é do ano de 2014 e segundo estatísticas divulgadas possui um total de 1014 conjuntos de dados que tratam diferentes domínios.

Nas próximas seções deste capítulo, os principais conceitos da Web Semântica que foram aplicados durante o desenvolvimento desta dissertação são revisados. Na Seção 3.1 é feita uma introdução sobre ontologias e na Seção 3.2 sobre RDF. Na Seção 3.3 é feita uma breve explicação sobre a sintaxe e álgebra da linguagem SPARQL. Na Seção 3.4 é feita uma breve revisão do vocabulário RDF Data Cube, que auxilia a descrição e publicação de dados com características multidimensionais e seus metadados em triplas RDF. Por final, na Seção 3.5 são discutidos trabalhos relacionados que propuseram abordagens para o processamento analítico de dados com características multidimensionais em RDF descritos utilizando o vocabulário RDF Data Cube ou vocabulários relacionados.

3.1 ONTOLOGIA

Para a Web Semântica, a ontologia funciona como um modelo que define a estrutura de um domínio, incluindo a sua taxonomia, caracterização dos seus conceitos, relação entre

[16] http://lod-cloud.net/

eles e suas restrições específicas. É a maneira adotada para modelar conceitos no mundo da Web Semântica.

Possui a característica de ser entendida não somente por humanos, mas também processável por máquinas. Devido a essa característica, acrescenta uma camada de inferência sobre objetos da Web (GRUBER et al., 1993). Além disso, ontologias estabelecem níveis de interoperabilidade entre as aplicações de Web Semântica e acrescentam significado, representação e expressividade sobre as camadas atuais da Web (DJURIC, GASEVIC, DEVEDZIC, 2006).

Os principais elementos de uma ontologia são:

- Vocabulário controlado: Lista de termos do domínio enumerados de forma explícita. Todos os termos devem ser definidos de forma não ambígua e redundante.
- Taxonomia: Coleção de vocabulários controlados organizados seguindo uma estrutura hierárquica. Por exemplo, o termo `Livro` pode se especializar em `Livro de Ficção` ou `Livro de Não-Ficção`.
- Conjunto de relações: Especifica as relações entre os termos do vocabulário controlado. Além das relações de hierarquia providas pela taxonomia, as relações presentes dentro da ontologia adicionam semântica, estabelecendo ligações entre os conceitos do domínio.
- Axiomas: Regras e restrições pertinentes ao domínio tratado. Viabilizam a criação de inferências na ontologia, que permitem a descoberta de novas relações entre os conceitos.

Uma ontologia pode ser modelada definindo-se as classes do domínio, as propriedades e restrições que as classes podem ter e as relações entre as classes. As classes são utilizadas para definir conjuntos de objetos (também chamados de indivíduos) que compartilham certa característica. Utilizando os conceitos de taxonomia, é possível ainda introduzir o conceito de herança de classes ou, mais comumente, subclasse de classes.

Para definir as características das classes, ou seja, aspectos que diferenciam uma classe da outra, são utilizadas propriedades. Existem dois tipos de propriedades que podem ser criadas em uma ontologia: Propriedades de dados, que tem como papel definir os atributos das classes, determinando os dados relacionados aos objetos e quais são os tipos destes dados. Propriedade de objeto, definem as relações entre dois objetos em uma ontologia. As propriedades também podem ser hierarquizadas em taxonomias.

Ontologias são, normalmente, escritas através da linguagem OWL (*Web Ontology Language*), declarada como uma recomendação W3C de 10 de fevereiro de 2004 (SMITH, WELTY, MCGUINNESS, 2004). Linguagem capaz de formalizar conceitos de um domínio, definindo as classes e suas propriedades; definir os indivíduos; e inferir sobre essas classes e indivíduos utilizando as semânticas providas pela OWL.

3.2 RDF

A ontologia define o esquema para o modelo conceitual do domínio. O RDF é um padrão utilizado para declarar os recursos instanciados, de acordo com a ontologia, com suas propriedades e as ligações entre eles. Descreve as informações em um formato de dados legível por máquina. Os elementos essenciais da linguagem RDF são:

- *Uniform Resource Identifiers*[17] (URIs): Identificadores únicos que referenciam recursos. Exemplos de URIs são URLs que são utilizadas para procurar recursos na Web. *Blank nodes* podem ser utilizados em um documento RDF para representar recursos para os quais uma URI não foi definida.
- Tripla RDF: Um documento RDF é composto por um conjunto de declarações sobre recursos que consistem de três elementos: sujeito, predicado e objeto. Cada declaração descreve a propriedade (predicado) de um recurso (sujeito) atribuindo um valor para essa propriedade (objeto). O sujeito e o objeto podem ser tanto uma URI quanto um *blank node*. O objeto também pode ser um literal, caso a propriedade de um sujeito for expressa a partir de um valor específico em vez de uma relação com um outro recurso. Este literal é representado por uma *string* que pode ser, adicionalmente, identificada por um tipo de dados, tais como no exemplo `"Brazil"^^xsd:string`.
- Grafos RDF: Um documento RDF pode ser visualizado a partir de um grafo direcionado, formado pelas triplas RDF declaradas no documento, no qual o sujeito e o objeto de uma tripla são representados como os nós do grafo e o predicado como um arco que liga o sujeito e objeto da tripla.
- Notação RDF - As declarações RDF podem ser representadas de diferentes formas que possam ser facilmente processadas por máquinas. Existem diferentes notações (sintaxes) que podem ser adotadas, tais como os formatos N3[18], Turtle[19] e N-

[17] http://www.ietf.org/rfc/rfc2396.txt

[18] https://www.w3.org/DesignIssues/Notation3

Triples[20]. O formato N-Triples utiliza a sintaxe mais simples, representando dados RDF basicamente a partir de uma coleção de triplas linearmente listadas, cada uma limitada por um ponto ("."). Existe redundância nas declarações e não se declara prefixos ("*namespaces*") no começo do documento, para simplificar as URI.

3.3 LINGUAGEM SPARQL

SPARQL é uma linguagem de consulta e um protocolo para acessar dados RDF, recomendado pela W3C. A primeira versão da linguagem, SPARQL 1.0, foi declarada uma recomendação oficial da W3C em 15 de janeiro de 2008 (PRUD'HOMMEAUX, SEABORNE, 2008). A linguagem foi estendida por recursos tais como agregações resultando na versão 1.1, recomendação W3C de 21 de Março 2013 (HARRIS, PRUD'HOMMEAUX, SEABORNE, 2013).

O tradutor de consultas SPARQL para SQL construído nesta dissertação oferece suporte à versão 1.1 da linguagem. Nesta seção são introduzidos os recursos da linguagem que são suportados pelo tradutor. No final, é feita uma explicação sobre a álgebra da linguagem.

Considere a consulta SPARQL seguinte:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x ns:price ?price .
          ?x dc:title ?title . }
```

Esta consulta é constituída de três partes, separadas pelas palavras PREFIX, SELECT e WHERE.

- PREFIX: Também conhecido como *namespaces*. Os prefixos tem como objetivo simplificar as URIs, são definidos no começo da consulta, sendo associados a URLs.
- SELECT: Identifica as variáveis que devem aparecer no resultado da consulta.
- WHERE: Define um Padrão Básico de Grafo (*Basic Graph Pattern*, na sigla em Inglês: BGP) que deve ser comparado ao grafo RDF em busca dos nós e arestas que são válidos para a consulta.

---

[19] https://www.w3.org/TeamSubmission/turtle/

[20] https://www.w3.org/2001/sw/RDFCore/ntriples/

3.3.1 Padrão Básico de Grafo

Um Padrão Básico de Grafo contém pelo menos uma tripla. Nesta tripla, o sujeito, predicado ou objeto podem ser uma variável, que é representada por um ponto de interrogação ("?") na frente. Uma variável é um espaço reservado para qualquer valor no documento RDF que combine com o resto da declaração da tripla. Cada consulta SPARQL resulta em um conjunto de todos os possíveis mapeamentos das variáveis, conhecidos como *solution mappings*. Se uma variável dentro de um Padrão Básico de Grafo aparecer em uma outra tripla, ela deve ser mapeada com o mesmo valor em ambas as triplas, para que esta seja uma solução válida para a consulta. No exemplo de consulta SPARQL apresentado na seção anterior, a variável `?x` deve ser mapeada com o mesmo valor em ambas as triplas que formam o Padrão Básico de Grafo na consulta.

3.3.2 Filtros

A cláusula FILTER restringe como soluções para a consulta somente aquelas para as quais a expressão do filtro é avaliada como verdadeira. A linguagem SPARQL 1.1 dá suporte a diversos operadores lógicos e funções que podem ser utilizadas para criar filtros sobre as consultas. Na consulta SPARQL seguinte, somente serão retornados como resultado os livros que possuem um valor associado ao seu título.

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
SELECT  ?book
WHERE   { ?book dc:title ?title
          FILTER bound(?title)
        }
```

3.3.3 Mapeamentos opcionais

A cláusula OPTIONAL permite que sejam feitos mapeamentos opcionais ao resultado da consulta. Ou seja, caso um mapeamento corresponda ao Padrão Básico de Grafo definido dentro da cláusula OPTIONAL, este mapeamento será adicionado ao resultado da consulta. Entretanto, caso haja outros mapeamentos que não correspondem ao Padrão Básico de Grafo definido na cláusula OPTIONAL, estes também irão fazer parte do resultado da consulta, não os excluindo. Na consulta SPARQL abaixo são localizados os nomes das pessoas no grafo RDF. Se houver um tripla com o predicado `foaf:mbox` presente e o mesmo valor mapeado

para a variável `?x` em ambas as triplas, a solução irá conter o objeto da tripla na variável `?mbox`.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox  ?mbox }
       }
```

3.3.4 Combinando mapeamentos

Dois conjuntos de mapeamentos podem ser combinados com o operador UNION, que age a partir da lógica do ou (disjunção lógica). Assim, cada mapeamento que faz parte da solução da consulta ou está presente em um dos conjuntos de mapeamento ou está presente em ambos. A consulta SPARQL abaixo busca por todos os livros cujo autor seja Alice ou Bob ou ambos.

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0>
SELECT ?book
WHERE  { { ?book dc10:creator "Alice" }
         UNION
         { ?book dc10:creator "Bob" }
       }
```

3.3.5 Modificadores de solução

Uma consulta gera como resultado uma sequência de soluções, que é tratada, inicialmente, de forma desordenada. Modificadores de solução podem ser aplicados em cima desta sequência para criar outra sequência de soluções. Os modificadores de solução que são suportados pelo tradutor construído nesta dissertação são ORDER (ordena a sequência), PROJECTION (escolhe as variáveis presentes na sequência), DISTINCT (garante que as soluções dentro da sequência são únicas), OFFSET (controla a partir de que ponto as soluções começam a ser apresentadas na sequência global de soluções) e LIMIT (restringe o número de soluções).

3.3.6 Agrupando mapeamentos

A versão 1.1 da linguagem SPARQL passou a oferecer suporte a operadores de agregações e agrupamentos. A cláusula GROUP BY transforma um conjunto de mapeamentos, de modo que apenas uma linha é exibida para cada variável de agrupamento. Uma consulta pode especificar uma cláusula HAVING, que aplica um filtro sobre o conjunto de mapeamentos após o agrupamento. O filtro pode envolver as variáveis da cláusula GROUP BY ou operadores de agregação. A cláusula HAVING é avaliada antes do SELECT, por isso não pode conter *alias*.

O tradutor criado dá suporte a todos os operadores de agregação do SQL (COUNT, SUM, AVG, MAX, MIN). Estes operadores ainda podem ser combinadas com a cláusula DISTINCT para que sejam agregados somente valores únicos. Na consulta SPARQL a seguir somente são contados os amigos distintos de cada pessoa.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(distinct ?friend) AS ?count)
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows ?friend .
} GROUP BY ?name
```

3.3.7 Álgebra SPARQL

A álgebra SPARQL define a semântica de execução de uma consulta SPARQL. A execução de uma consulta SPARQL ocorre da seguinte maneira: Um *parser* converte a consulta em uma árvore de sintaxe, a qual é traduzida posteriormente para uma árvore de álgebra SPARQL, conforme descrito em (SEABORNE, 2007), que é processada para que seja calculado o resultado da consulta. Para gerar a árvore de álgebra SPARQL a partir de uma consulta SPARQL é muito utilizada a biblioteca Jena ARQ (SEABORNE, 2010). O Quadro 1 ilustra os operadores da álgebra SPARQL e as expressões de sintaxe correspondentes.

Quadro 1: Operadores da Álgebra e Sintaxe SPARQL

Operador Álgebra	**Sintaxe SPARQL**	**Significado**
BGP(T)	Conjunto de triplas separadas por um (.)	Seleciona conjunto de triplas no grafo RDF
LeftJoin(M1,M2,F)	OPTIONAL	Os mapeamento M1 e M2 serão combinados, se a condição F for atendida
Filter(M,F)	FILTER, HAVING	Os mapeamentos de M que atendem à condição F serão válidos
Union(M1,M2)	UNION	Os mapeamentos M1 e M2 serão unidos
Group(M,Vars)	GROUP BY	Agrupa os mapeamentos de M de acordo com as variáveis de agrupamento de Vars
Slice(M,Início,Fim)	LIMIT, OFFSET	Modificadores de solução que influenciam como o mapeamento M será retornado
Order(M,Vars)	ORDER BY	Ordena os mapeamentos de M de acordo com as variáveis de ordenação Vars
Distinct(M)	DISTINCT	Somente mapeamentos distintos em M serão retornados
Project(M,Vars)	SELECT	Apenas os mapeamentos com variáveis de Vars são selecionados

A consulta SPARQL abaixo é um exemplo de uma consulta que utiliza alguns dos operadores da álgebra SPARQL. A consulta retorna como resultado a quantidade de amigos que cada pessoas tem, sendo que esta quantidade tem que ser maior que 10. Opcionalmente o amigo pode ser autor de algum artigo.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?name (count(?friend) AS ?count)
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows ?friend .
    OPTIONAL{ ?article dc:author ?friend .}
} GROUP BY ?name
HAVING(COUNT(?friend) > 10)
```

A tradução da consulta em formato de álgebra SPARQL, realizada a partir da biblioteca Jena ARQ, é ilustrada abaixo:

```
(prefix ((dc: <http://purl.org/dc/elements/1.1/>)
         (foaf: <http://xmlns.com/foaf/0.1/>))
  (project (?name ?count)
    (filter (> ?.0 10)
      (extend ((?count ?.0))
        (group (?name) ((?.0 (count ?friend)))
          (leftjoin
            (bgp
              (triple ?person foaf:name ?name)
              (triple ?person2 foaf:knows ?friend)
            )
            (bgp (triple ?article dc:author ?friend)))))))
```

A representação em formato de árvore de álgebra pode ser visualizada na Figura 7. Os nós da árvore podem representar uma tripla RDF, um operador da álgebra ou o nome de uma variável. As triplas RDF formam as folhas da árvore e representam as entradas para os nós com o operador BGP, das quais as triplas fazem parte, que estão um nível acima das folhas. Estes nós de operadores BGP produzem conjuntos de mapeamento temporários, que são repassados para os nós que se situam acima deles, produzindo novos conjuntos de mapeamento. Este processo é executado recursivamente até que o nó raiz da árvore seja atingido.

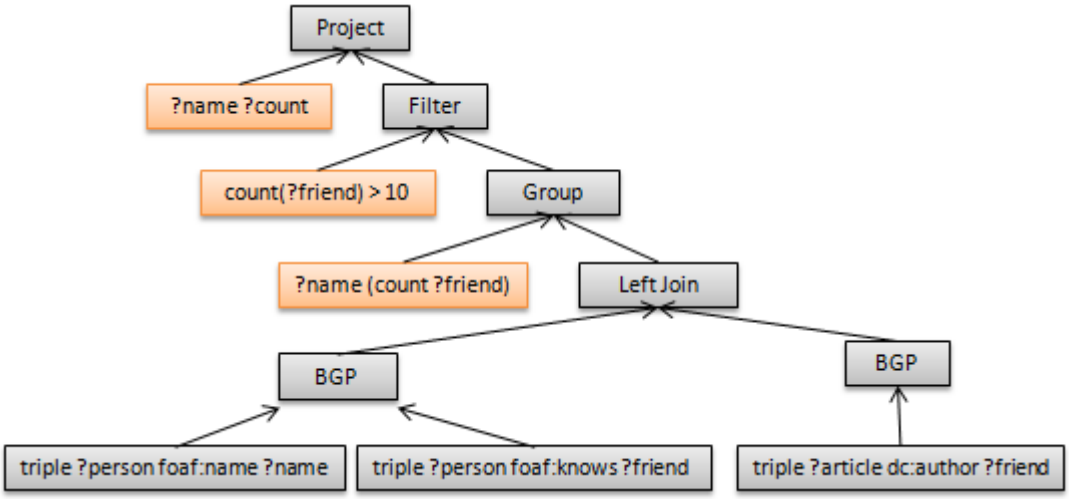

Figura 7: Exemplo de árvore de álgebra SPARQL

3.4 VOCABULÁRIO RDF DATA CUBE

O vocabulário RDF Data Cube (CYGANIAK, REYNOLDS, TENNISON, 2014) foi especificamente concebido para publicar cubos de dados na Web, de tal forma que eles possam ser ligados a conjuntos de dados relacionados em RDF. Foi declarado uma recomendação oficial da W3C em 16 de janeiro de 2014. O modelo por trás do vocabulário RDF Data Cube é compatível com o modelo SDMX, padrão ISO para intercâmbio e compartilhamento de dados e metadados estatísticos entre organizações (INTERNATIONAL STANDARD, 2005).

Segundo o vocabulário, dados oriundos de processos de medição podem ser definidos como um espaço multidimensional, ou cubo. Um cubo é organizado de acordo com um conjunto de dimensões, atributos e medidas (observações), chamados coletivamente de componentes.

A Figura 8 ilustra os termos chave do vocabulário e seus relacionamentos. O vocabulário representa as dimensões, atributos e medidas como classes `qb:DimensionProperty`, `qb:AttributeProperty` e `qb:MeasureProperty`, respectivamente, que por sua vez são subclasses de `qb:ComponentProperty`. Um `qb:DataStructureDefinition` define a estrutura de um ou mais conjuntos de dados (`qb:DataSet`). Ele poderá ser reutilizável em todos os conjuntos de dados que possuam a mesma estrutura. O `qb:DataStructureDefinition` faz referência a um conjunto de recursos `qb:ComponentSpecification` através da propriedade `qb:component`. No caso mais simples o `qb:ComponentSpecification` simplesmente referencia o `qb:ComponentProperty` correspondente (geralmente usando uma das sub propriedades `qb:dimension`, `qb:measure` ou `qb:attribute`).

Cada observação é representada como uma instância da classe `qb:Observation`. No caso básico, os valores para cada um dos atributos, dimensões e medições são diretamente ligados à observação. A observação é ligada ao conjunto de dados correspondente utilizando-se a propriedade `qb:dataSet`.

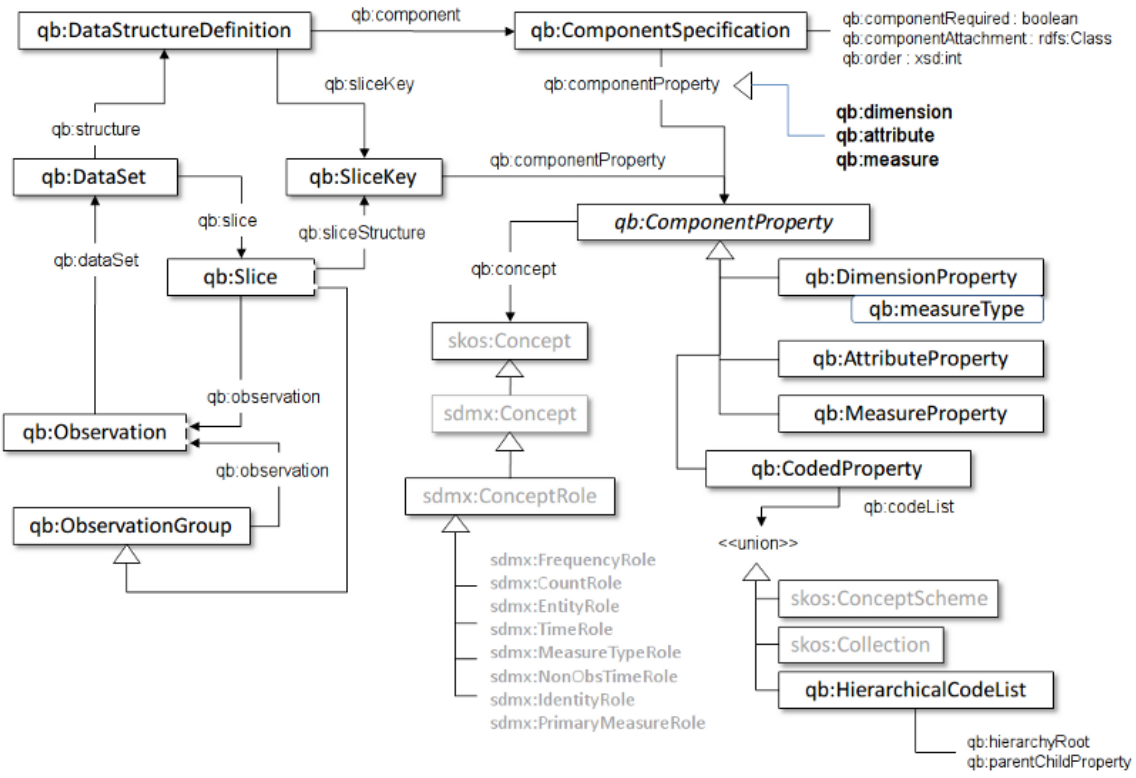

Figura 8: Termos chaves do Vocabulário RDF Data Cube (CYGANIAK; REYNOLDS; TENNISON, 2013)

3.5 TRABALHOS RELACIONADOS - PROCESSAMENTO ANALÍTICO DE DADOS MULTIDIMENSIONAIS EM RDF

Agrupamos os trabalhos relacionados em cinco grupos de acordo com o tipo de abordagem proposta para o processamento analítico de dados multidimensionais em RDF:

- 1ª abordagem: Abordagens para o processamento analítico de dados multidimensionais convertidos para RDF sob demanda. A conversão é realizada a partir da aplicação de operações OLAP em cima de dados multidimensionais, no formato relacional, que geram cubos de dados, mapeados para o formato de triplas RDF utilizando vocabulários padrões (SALAS, 2015; PESCE, 2012).
- 2ª abordagem: Abordagem para o processamento analítico de dados RDF convertidos para o modelo multidimensional, antes de serem efetuadas operações OLAP padrões (KÄMPGEN, HARTH, 2011).
- 3ª abordagem: Abordagens para o processamento analítico direto de conjuntos de dados RDF, sem a necessidade de efetuar processos de conversão (KÄMPGEN, HARTH, 2014; GHASEMI et al., 2014; SOUZA et al., 2014).
- 4ª abordagem: Abordagens para o processamento analítico de dados multidimensionais armazenados em sistemas de gerenciamento de dados RDF que

envolvem bancos de dados relacionais e esquemas de tradução de consultas SPARQL em SQL (SEQUEDA, MIRANKER, 2013; ELLIOTT et al., 2009; PRUD'HOMMEAUX, BERTAILS, 2008; BIZER, SEABORNE, 2004; AUER et al., 2015).

- 5ª abordagem: Abordagens para o processamento analítico de conjuntos de dados RDF armazenados em bancos de dados distribuídos (SCHÄTZLE, PRZYJACIEL-ZABLOCKI, LAUSEN, 2011; NEU, 2014; HAQUE, PERKINS, 2012).

No Quadro 2 é feita uma comparação entre as abordagens dos trabalhos relacionados levantados.

Quadro 2: Comparação das abordagens propostas nos trabalhos relacionados

Tipo de Abordagem	**Trabalhos Encontrados**
Processamento analítico de dados armazenados em bancos multidimensionais convertidos para RDF sob demanda	**OLAP to RDF(*DataCube)*** (SALAS, 2015; PESCE, 2012)
Processamento analítico de dados RDF convertidos para o modelo multidimensional, antes de serem efetuadas operações OLAP padrões	**RDF(*DataCube*) to Multidimensional Model (MDM)** (KÄMPGEN, HARTH, 2011)
Processamento analítico direto de conjuntos de dados RDF, sem a necessidade de efetuar processos de conversão	**MDX to SPARQL** (KÄMPGEN, HARTH, 2014; GHASEMI et al., 2014)
	Consultas OLAP formuladas diretamente em SPARQL (SOUZA et al., 2014)
Processamento analítico de conjuntos de dados armazenados em sistemas de gestão de dados RDF que envolvem bancos relacionais	**RDF apoiado por SGBDR e RDB2RDF** (SEQUEDA, MIRANKER, 2013; ELLIOTT et al., 2009; PRUD'HOMMEAUX, BERTAILS, 2008; BIZER, SEABORNE, 2004; AUER et al., 2015)
Processamento analítico de conjuntos de dados RDF armazenados em bancos de dados distribuídos	**SPARQL 1.0 to Pig Latin** (SCHÄTZLE, PRZYJACIEL-ZABLOCKI, LAUSEN, 2011)
	SPARQL 1.0 to Impala SQL (NEU, 2014)
	SPARQL 1.0 to HiveQL (HAQUE, PERKINS, 2012)

3.5.1 Abordagens para o processamento analítico de dados multidimensionais convertidos para RDF sob demanda

Em (SALAS, 2015) foi proposto o *framework OLAP2DataCube* que permite que grandes bases de dados analíticos, representadas de acordo com o paradigma OLAP, possam ser eficientemente transformadas para o formato RDF. No *framework* proposto, operações OLAP são executadas diretamente sobre as bases de dados subjacentes a partir de consultas escritas na linguagem SQL. Após isso, o resultado da consulta SQL é mapeado em tempo de execução em triplas RDF utilizando os vocabulários RDF Data Cube e QB4OLAP (ETCHEVERRY, VAISMAN, 2012).

Em (PESCE, 2012) foi proposto o *framework OLAP2DataCube Catalog On Demand* que permite que dados multidimensionais estatísticos armazenados em um catálogo sejam filtrados e visualizados no formato RDF, através de transformações OLAP, do tipo *slice* e *dice*, em cubos de dados. O *framework* é composto por quatro módulos: camada de aplicação, camada de mediação, catálogo e camada wrapper.

A camada de aplicação permite que os usuários interajam com o *framework* informando que dados desejam buscar. Esta informação é repassada para a camada de mediação que é responsável por chamar o catálogo que retorna os cubos de dados que atendem à busca feita pelo usuário. A aplicação, então, exibe uma lista contendo todos os cubos retornados, para que o usuário faça sua escolha. Após escolhido um dos cubos, o usuário seleciona dimensões e métricas do seu interesse. Dessa forma, a camada de aplicação monta um arquivo no formato RDF, codificando as escolhas feitas pelo usuário, e por sua vez chama a camada de mediação requisitando os dados correspondentes. A camada wrapper tem como responsabilidade acessar o banco de dados e gerar triplas RDF, utilizando o vocabulário RDF Data Cube. A camada de mediação repassa o arquivo RDF gerado pelo wrapper para a aplicação, que exibe estes dados para o usuário.

O principal inconveniente das duas abordagem descritas é que os dados triplificados são gerados por demanda, a partir da execução de operações OLAP, não sendo armazenados fisicamente em um repositório de triplas. Logo, apesar de poderem ser consultados, não podem ser publicados fisicamente na Web.

3.5.2 Abordagem para o processamento analítico de dados RDF convertidos para o modelo multidimensional

Em (KÄMPGEN, HARTH, 2011) foi proposta uma abordagem para a conversão de dados no formato RDF (descritos utilizando o vocabulário RDF Data Cube) em um modelo multidimensional. A estrutura resultante é manipulada utilizando o servidor OLAP Mondrian e consultada utilizando a linguagem MDX (*Multi-Dimensional Expressions*), linguagem para execução de consultas em bancos de dados multidimensionais. Assim, as manipulações OLAP são realizadas em fontes de dados multidimensionais e não diretamente em conjuntos de dados RDF. O inconveniente desta abordagem é que o processo de conversão deve ser repetido toda vez que alterações sejam feitas nos dados originais.

3.5.3 Abordagens para o processamento analítico direto de conjuntos de dados RDF

Em (KÄMPGEN, HARTH, 2014) foi proposto o *framework OLAP4LD*, um *framework* para a construção de aplicações de análise em dados ligados reutilizando o vocabulário RDF Data Cube. *OLAP4LD* fornece acesso a conjuntos de dados multidimensionais publicados utilizando o vocabulário RDF Data Cube. O *framework* traduz consultas MDX em consultas SPARQL executadas sobre fontes de dados ligados, e traduz os resultados em representações compreensíveis pelo cliente. Em um trabalho semelhante (GHASEMI et al., 2014) fornece acesso e manipulação de conjuntos de dados multidimensionais por meio de consultas MDX e retorna como resposta um conjunto de dados RDF descritos utilizando o vocabulário RDF Data Cube.

Em (SOUZA et al., 2014) foi proposta uma primeira abordagem de conversão dos dados do projeto PingER, detalhado na Seção 2.3, para o formato RDF. Uma ontologia de domínio foi desenvolvida para servir como um vocabulário de referência e estrutura para representar e anotar os dados do projeto como triplas RDF e publicá-los seguindo o modelo de dados ligados, podendo ser acessados a partir de um terminal SPARQL. A ontologia do domínio PingER criada é uma adaptação da ontologia MOMENT (*Monitoring and Measurement in the Next Generation Technologies*) (SALVADOR et al., 2010; RAO, 2010), uma ontologia central (*core ontology*) que conceitua o domínio de medições de desempenho de rede.

A Figura 9 mostra uma visão geral do modelo da ontologia. No centro fica a superclasse principal, classe `Measurement`, que representa o processo de aquisição das medidas. `Measurement` relaciona-se com as seguintes classes, a fim de qualificar a medição:

`Metric`, através da relação `measuresMetric`, para especificar qual métrica de rede está sendo medida; `MeasurementParameters`, através da relação `hasMeasurementParameters`, para especificar os atributos de medição; `DateTime`, através da relação `hasDateTime`, para especificar o intervalo de tempo em que a medição foi feita; `SourceDestinationNodes`, através da relação `hasSourceDestinationNode`, para especificar os *links* de Internet medidos. A classe `SourceDestinationNodes` se relaciona com a classe `NetworkNode` para definir os dois nós de rede que compõem o link de Internet, sendo que um nó desempenha o papel de nó monitor, enviando o sinal de *ping*, e outro desempenha o papel do nó monitorado, recebendo o *ping*. A ligação é feita através das relações `PingER-ont:hasSourceNode` e `PingER-ont:hasDestinationNode`, respectivamente.

Para definir os parâmetros de tempo (quando a medição foi realizada) e espaço (onde os nós de rede - `NetworkNode` - estão localizados), foram usados conceitos extraídos das ontologias Time (HOBBS, PAN, 2006) e Geonames (VATANT, WICK, 2012).

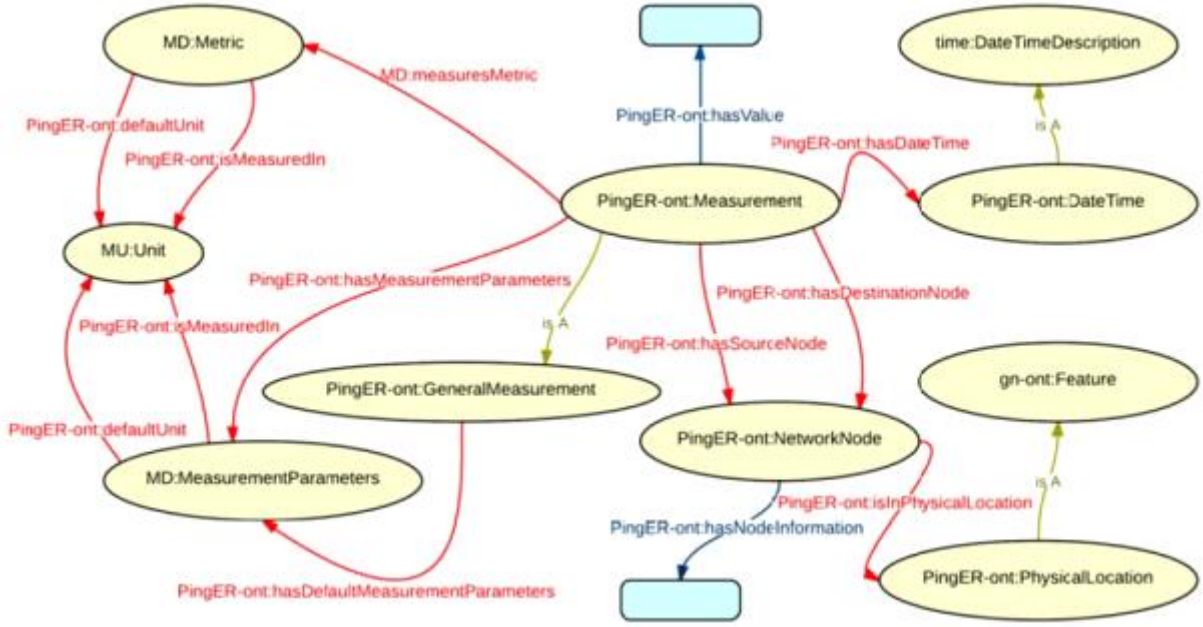

Figura 9: Ontologia PingER original (SOUZA et al., 2014)

Para permitir o processamento analítico dos dados do projeto PingER no formato RDF foi disponibilizado um terminal SPARQL[21], que recebe consultas escritas na linguagem SPARQL, consulta a base de dados armazenada no repositório Open RDF Sesame Native 2.7.2 e envia as respostas nos formatos JSON, XML ou CSV.

[21] http://pingerlod.slac.stanford.edu/sparql

Como os dados do projeto PingER não foram descritos como triplas RDF utilizando o vocabulário Data Cube ou um vocabulário relacionado que permita modelar os dados a partir de uma estrutura de cubo, a modelagem e execução de operações de OLAP sobre os dados foi limitada. Além disso, a escolha de armazenar os dados convertidos no repositório Open RDF Sesame Native 2.7.2 se mostrou inadequada na medida em que uma quantidade superior a 50 milhões de triplas passou a ser manipulada a partir de consultas complexas (consultas do tipo *mashup*, que incluíam integrações com conjuntos de dados externos) no repositório, gerando resultados de desempenho insatisfatórios.

3.5.4 Abordagens para o processamento analítico de dados multidimensionais armazenados em sistemas de gestão de dados RDF que envolvem bancos relacionais

Dentro da gestão de dados RDF existem tecnologias que se encaixam na categoria de Repositórios de Triplas e na categoria de Bancos Relacionais Legados mapeados para RDF (*Relational Database to RDF*, na sigla em inglês: RDB2RDF) (SEQUEDA, MIRANKER, 2013). Dentre as categorias de Repositórios de Triplas, existe a categoria de Repositórios de Triplas apoiados por Sistemas de Gerenciamento de Bancos de Dados Relacionais (SGBDR).

Os Repositórios de Triplas apoiados por SGBDR adicionam uma camada específica RDF a um SGBDR que utiliza um esquema de dados que modela e armazena triplas RDF. O modelo de dados adotado por este tipo de repositório é o RDF. Já os sistemas RDB2RDF, apresentam uma visão RDF lógica dos dados relacionais que estão fisicamente armazenados no SGBDR, de forma que nenhuma cópia dos dados relacionais seja feita. No caso destes tipos de sistemas, o modelo relacional é o modelo de dados primário adotado.

Apesar de adotarem modelos de dados primários diferentes, tanto os Respositórios de Triplas apoiados por SGBDR quantos os sistemas RDB2RDF envolvem banco de dados relacionais e esquemas de tradução de consultas SPARQL para SQL. Alguns trabalhos na literatura propuseram esquemas para o mapeamento de consultas elaboradas na linguagem SPARQL para a linguagem SQL. Em (SEQUEDA, MIRANKER, 2013), (ELLIOTT et al., 2009) e (PRUD'HOMMEAUX, BERTAILS, 2008) foram propostas semânticas para representar dados relacionais como grafos RDF e esquemas para o mapeamento de consultas SPARQL em consultas SQL a serem executados sobre os dados relacionais originais. Foi verificado que todos os esquemas de tradução propostos não dão suporte a tradução de funções de agregação.

Sistemas RDB2RDF como D2R (BIZER, SEABORNE, 2004) e Sparqlify (AUER et al., 2015) implementam algoritmos que permitem a criação de visões RDF em cima de dados

relacionais e elaboração de consultas SPARQL sobre estas visões. Com exceção da função de agregação COUNT(*), o sistema Sparqlify na sua versão Alpha ainda não oferecia suporte a outras funções de agregação. Já o sistema D2R oferece suporte a todos os operadores da linguagem SPARQL 1.1, incluindo operadores de agregação.

3.5.5 Abordagens para o processamento analítico de conjuntos de dados RDF armazenados em bancos de dados distribuídos

Conforme relatado em (STEWART, TRINDER, LOIDL, 2011), uma série de linguagens de consulta de alto nível estão sendo construídas sobre o ambiente Hadoop com objetivo de abstrair alguns recursos de programação MapReduce. Este alto nível de abstração é criado para que o desenvolvedor não precise entender a fundo o funcionamento de sistemas distribuídos para realizar a análise dos dados. Dentre as linguagens mais importantes estão Pig Latin[22] e HiveQL[23]. Programas escritos nestas linguagens são compilados em uma sequência de tarefas MapReduce, que são executados em um ambiente Hadoop.

Assim como a abordagem proposta nesta dissertação, alguns estudos já propuseram abordagens para a execução de consultas SPARQL sobre um *cluster* Hadoop. Estes estudos implementaram mecanismos de tradução distintos, realizando o mapeamento de consultas SPARQL em consultas equivalentes escritas em diferentes linguagens de alto nível.

Ao contrário dos estudos apresentados nas seções anteriores, cujo objetivo principal foi gerar abordagens que possibilitem o processamento analítico de conjuntos de dados multidimensionais em RDF sem focar nos aspectos de desempenho das abordagens propostas, os estudos apresentados nesta seção tem como objetivo principal prover a melhoria do desempenho de consultas SPARQL executadas sobre grandes volumes de dados, empregando para isso tecnologias de distribuição de processamento.

Em (SCHÄTZLE, PRZYJACIEL-ZABLOCKI, LAUSEN, 2011) propõe-se um *framework* para a tradução de consultas na linguagem SPARQL em um programa equivalente em Pig Latin, dando suporte a todos os operadores da recomendação SPARQL 1.0. Pig Latin é a linguagem usada pela plataforma Apache Pig, a qual permite escrever transformações MapReduce complexas usando uma linguagem de script simples. O mecanismo de tradução foi avaliado a partir do *benchmark* SP2Bench (SCHMIDT et al., 2009), que é específico para medir o desempenho de consultas SPARQL, que seguem o estilo OLTP. O conjunto de dados

[22] http://hortonworks.com/hadoop/pig/

[23] https://hive.apache.org/

RDF utilizado foi armazenado no HDFS seguindo o esquema de armazenamento de particionamento vertical (ABADI et al., 2007), um dos três tipos de esquemas que modelam e armazenam triplas RDF em tabelas (os outros são tabela de triplas e tabela proprietária) que são investigados e comparados na Seção 5.2.1.

Neu (2014) investiga se o Cloudera Impala é adequado para o armazenamento de grandes volumes de dados RDF, fazendo a comparação de seu desempenho em relação a do Apache Hive, outra tecnologia de banco de dados baseada em MapReduce. Para este propósito, o Impala foi adaptado através da adição de uma camada superior que realiza a tradução de consultas SPARQL, seguindo a recomendação SPARQL 1.0, para o dialeto SQL utilizado pelo Impala. O desempenho do sistema foi testado em um *cluster* de computadores usando três conhecidos *benchmarks* OLTP da Web Semântica: LUBM (GUO, PAN, HEFLIN, 2005), BSBM (BIZER, SCHULTZ, 2009) e SP2Bench (SCHMIDT et al., 2009).

Os conjuntos de dados RDF utilizados foram armazenado no HDFS utilizando uma única e larga tabela proprietária (WILKINSON, 2006), contendo uma coluna para cada predicado presente no documento RDF trabalhado. Além disso, (NEU, 2014) optou por armazenar atributos multivalorados separadamente em uma tabela de triplas (CHONG et al., 2005), evitando com isso um aumento no espaço ocupado pelo arquivo de dados, caso os atributos multivalorados fossem expressos no formato de tabela proprietária.

Em (HAQUE, PERKINS, 2012) também é implementada a mesma abordagem dos estudos anteriores. Entretanto, foi utilizado o Apache Hive como gerador de consulta sobre o *cluster* Hadoop e o Hbase[24] como banco de dados distribuído. Foi adicionada uma camada superior que realiza a tradução de consultas SPARQL, seguindo a recomendação 1.0, para o dialeto HiveQL, que é a linguagem de consulta do Hive. O mecanismo de tradução foi avaliado a partir do *benchmark* OLTP BSBM (BIZER, SCHULTZ, 2009). Como o Hbase é orientado à coluna e utiliza o mecanismo de famílias de colunas (*column families*) para agrupar dados em um mesmo disco que possuam padrões de acesso similares, o esquema de dados adotado foi a abordagem de tabela proprietária, proposta em (WILKINSON, 2006), que sugere a criação tabelas proprietárias que agrupam sujeitos cujas propriedades tendem a ser definidas em conjunto.

Todos os esquemas de tradução implementados nos trabalhos citados oferecem suporte à versão 1.0 do SPARQL, sendo testados utilizando *benchmarks* OLTP. Logo, o diferencial

[24] http://hbase.apache.org/

da proposta desta dissertação é construir um mecanismo de tradução que ofereça suporte a consultas analíticas sobre dados multidimensionais, seguindo a recomendação SPARQL 1.1.

4 TECNOLOGIAS DE DISTRIBUIÇÃO

Nas próximas seções são introduzidas as tecnologias de distribuição de processamento e armazenamento de dados que foram utilizadas na aplicação criada nesta dissertação. Primeiramente, na Seção 4.1 é feita uma breve categorização de sistemas distribuídos. Na Seção 4.2 é feita uma introdução sobre o *framework* Apache Hadoop, que possui um conjunto de subprojetos que oferecem infraestrutura para sistemas distribuídos. Na Seção 4.3 é feita uma revisão sobre o Cloudera Impala, que utiliza o Hadoop para executar análise de dados em larga escala. Na Seção 4.4 é introduzido o *cluster* que foi montado para executar a aplicação distribuída desenvolvida nesta dissertação.

4.1 SISTEMAS DISTRIBUÍDOS

Um sistema de processamento distribuído é uma coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema: hardware, software e dados (COULOURIS, DOLLIMORE, KINDBERG, 2005). Os computadores que estão conectados pela rede podem estar espacialmente separados por qualquer distância, podendo estar separados por continentes ou dividindo a mesma sala.

As principais características dos sistemas distribuídos são: concorrência, uma vez que os nós do sistema compartilham recursos durante a execução de programas. Não existir uma noção global de tempo compartilhada entre os nós do sistema, já que a única forma de comunicação entre os nós é feita a partir de troca de mensagens. Falhas independentes, as quais resultam no isolamento do nó que falhou, mas não necessariamente na paralisação de todo o sistema.

A motivação principal para construção e utilização de sistemas distribuídos é a possibilidade de compartilhar recursos (COULOURIS, DOLLIMORE, KINDBERG, 2005). O termo recurso pode abranger uma variedade de coisas que podem ser proveitosamente compartilhadas entre computadores, desde componentes de hardware (como discos e impressoras) até componentes de software (arquivos, banco de dados e objetos de dados).

Sistemas distribuídos devem operar com eficácia e eficiência diante de muitas escalas diferentes, que vão desde de uma pequena intranet até a Internet. Um sistema é descrito como escalável se ele permanece eficaz quando há um aumento significante no número de recursos compartilhados e no número de usuários (COULOURIS, DOLLIMORE, KINDBERG, 2005).

4.2 APACHE HADOOP

O Apache Hadoop é um *framework* aberto escrito em Java para armazenamento e processamento distribuído de grandes conjuntos de dados em *clusters* de computadores (WHITE, 2012). Além de rodar em *clusters* de computadores tradicionais, ele também pode rodar na nuvem. A nuvem permite que as organizações implementem o Hadoop sem adquirir hardware ou conhecimentos de configuração específicos. Dentre os vendedores que atualmente oferecem este tipo de oferta está a Amazon com a Amazon Elastic Compute Cloud[25] (Amazon EC2).

O núcleo do Apache Hadoop é composto por dois módulos principais: *Hadoop Distributed File System* (HDFS), um sistema de arquivo distribuído que armazena dados ao longo de máquinas de um *cluster* e Hadoop MapReduce, um modelo de processamento de dados em larga escala e ambiente de execução que roda em *clusters* de computadores. Além destes módulos, o projeto Hadoop também é composto por outros dois módulos de suporte: Hadoop Common, contém bibliotecas e utilitários que suportam outros módulos e Hadoop Yarn[26], uma plataforma de gestão de recursos, responsável por fornecer operações consistentes, segurança e ferramentas de governança de dados em *clusters* Hadoop.

Além dos módulos principais citados, o *framework* Hadoop também possui outros subprojetos que oferecem uma série de serviços complementares ou adicionam uma camada de abstração sobre os módulos principais, facilitando o desenvolvimento. Exemplos de subprojetos são Apache Hive, Apache HBase, Cloudera Impala, Apache Hue, entre outros.

4.2.1 HDFS

O HDFS permite armazenar dados em blocos que são distribuídos nos computadores de um *cluster*. O tamanho padrão de um bloco de dados no HDFS é 64MB. Este tamanho é maior do que o tamanho de blocos de dados que costumam ser utilizados por discos, pois o HDFS costuma trabalhar com arquivos de dados bem maiores dos que são armazenados em discos comuns.

Por padrão, os blocos de dados são replicados três vezes no *cluster*. A replicação garante a segurança dos dados em caso de falha de hardware. Além disso, ela também garante o princípio da localidade de dados (BORTHAKUR, 2008).

[25] https://aws.amazon.com/pt/ec2/

[26] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Quando ocorre um processamento de dados em larga escala em um *cluster* de computadores, a largura de banda da rede pode se tornar um gargalo, caso grandes quantidades de dados devam ser copiados para as máquinas. O Hadoop MapReduce resolve este problema inicialmente permitindo que cada máquina processe apenas os dados disponíveis no seu dispositivo de armazenamento local. Através da replicação de dados no HDFS, os dados estão disponíveis para mais máquinas no *cluster*, permitindo o processamento paralelo.

O HDFS possui uma arquitetura do tipo cliente/servidor (BORTHAKUR, 2008), ilustrada na Figura 10. Um *cluster* HDFS é composto por um único NameNode, um servidor mestre que administra o sistema de arquivos e regula o acesso a arquivos por clientes. Além disso, há uma série de DataNodes, geralmente um por nó no *cluster*, que gerencia o armazenamento relacionado aos nós em que são executados.

O HDFS dispõe de um sistema de arquivos e permite que os dados do usuário sejam armazenados em arquivos. Internamente, um arquivo é dividido em um ou mais blocos e estes blocos são armazenados em um conjunto de DataNodes. O NameNode executa operações no sistema de arquivos como abrir, fechar e renomear arquivos e diretórios. Ele também determina o mapeamento de blocos para os DataNodes. Os DataNodes são responsáveis por executar os pedidos de leitura e escrita dos clientes do sistema de arquivos. Os DataNodes também realizam a criação, exclusão e replicação dos blocos sob comando do NameNode.

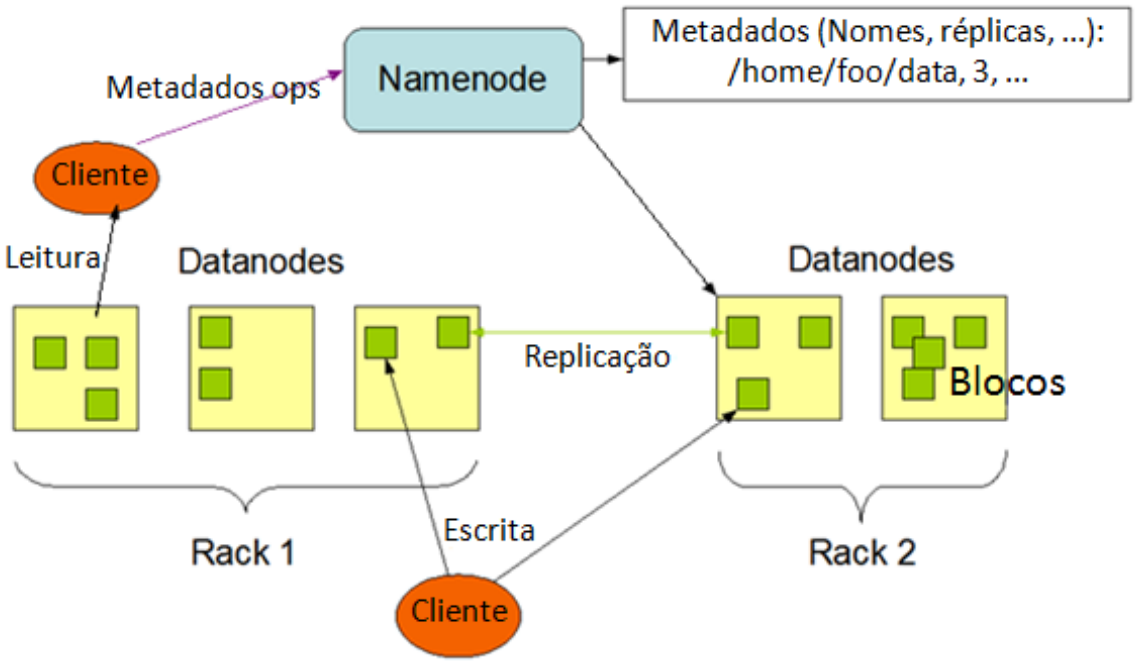

Figura 10: Arquitetura do HDFS (Adaptado de BORTHAKUR, 2008)

O *cluster* HDFS também pode conter um Secondary NameNode. Ele ocasionalmente se conecta ao NameNode (por padrão, a cada hora) e faz uma cópia dos metadados e arquivos usados para armazenar metadados (ambos os quais podem estar fora de sincronia) da memória do NameNode. As informações recolhidas são combinadas em um novo conjunto de arquivos pelo Secondary NameNode e oferecidas de volta para o NameNode, mantendo uma cópia para si. Caso o NameNode falhe, os arquivos mantidos pelo Secondary NameNode podem ser utilizados para recuperá-lo.

4.2.2 Hadoop MapReduce

Programas MapReduce, que também são chamados de *jobs*, seguem um paradigma especial e devem essencialmente implementar dois métodos, um método Map e um método de Reduce (WHITE, 2012; YAHOO DEVELOPER NETWORK, 2016). No início de um *job*, cada nó do *cluster* executa tarefas Map e lê os blocos de dados de entrada que estão localmente disponível para o nó. Os valores de entrada são representados a partir de pares chave-valor (MapInputKey e MaputInputValue). As chaves identificam os valores associados. Estes pares são processados pelo método Map, gerando também como resultado um conjunto de pares chave-valor de saída (MapOutputKey e MapOutputValue).

A fase Map é seguida pela fase Shuffle, em que os pares de saída da fase Map são classificados e agrupados de acordo com a sua chave (MapOutputKey). Eles são em seguida, enviados como entrada para a fase Reduce, que pode ser executada paralelamente em nós diferentes. Uma tarefa Reduce recebe todos os valores que têm uma mesma chave (MapOutputKey) e foram agrupados na fase Shuffle. Cada tarefa Reduce recebe uma chave (ReduceInputKey) e um conjunto de valores (ReduceInputValues) como entrada. Seu resultado é um conjunto de pares de chave-valor de saída (ReduceOuputKey, ReduceOutputValue). No final do *job* MapReduce, todos os valores são escritos no HDFS e resultados intermediários são eliminados. É possível definir o número de tarefas Reduce a zero. Neste caso, tanto a fase Shuffle quanto a fase Reduce são ignoradas, o que significa que os resultados da fase Map são escritos de forma não ordenada no HDFS.

A Figura 11 ilustra um exemplo clássico de programa MapReduce que realiza a contagem de palavras em um arquivo (YAHOO DEVELOPER NETWORK, 2016). O arquivo de entrada fica armazenado no HDFS. Diversas instâncias da fase Map são criadas em cada nó do *cluster*. Cada instância processa uma parte do arquivo original. Para cada palavra encontrada, a tarefa Map gera um par cuja a chave é a palavra e o inteiro "1" é o valor, gerando como saída o par (palavra, 1). Após as tarefas Map serem executadas, os pares

emitidos por elas serão ordenados e agrupados pela sua chave, na fase Shuffle. Instâncias da fase Reduce são criadas em diferentes nós. Cada tarefa Reduce é responsável por processar a lista de valores associados a uma palavra específica, isso garante que cada palavra seja processada por uma única tarefa Reduce. A lista de valores será uma lista de valores "1"s. A tarefa Reduce soma estes valores gerando uma contagem final associada a uma palavra, emitindo como saída final o par (palavra, contagem), que é gravado em um arquivo de saída armazenado no HDFS.

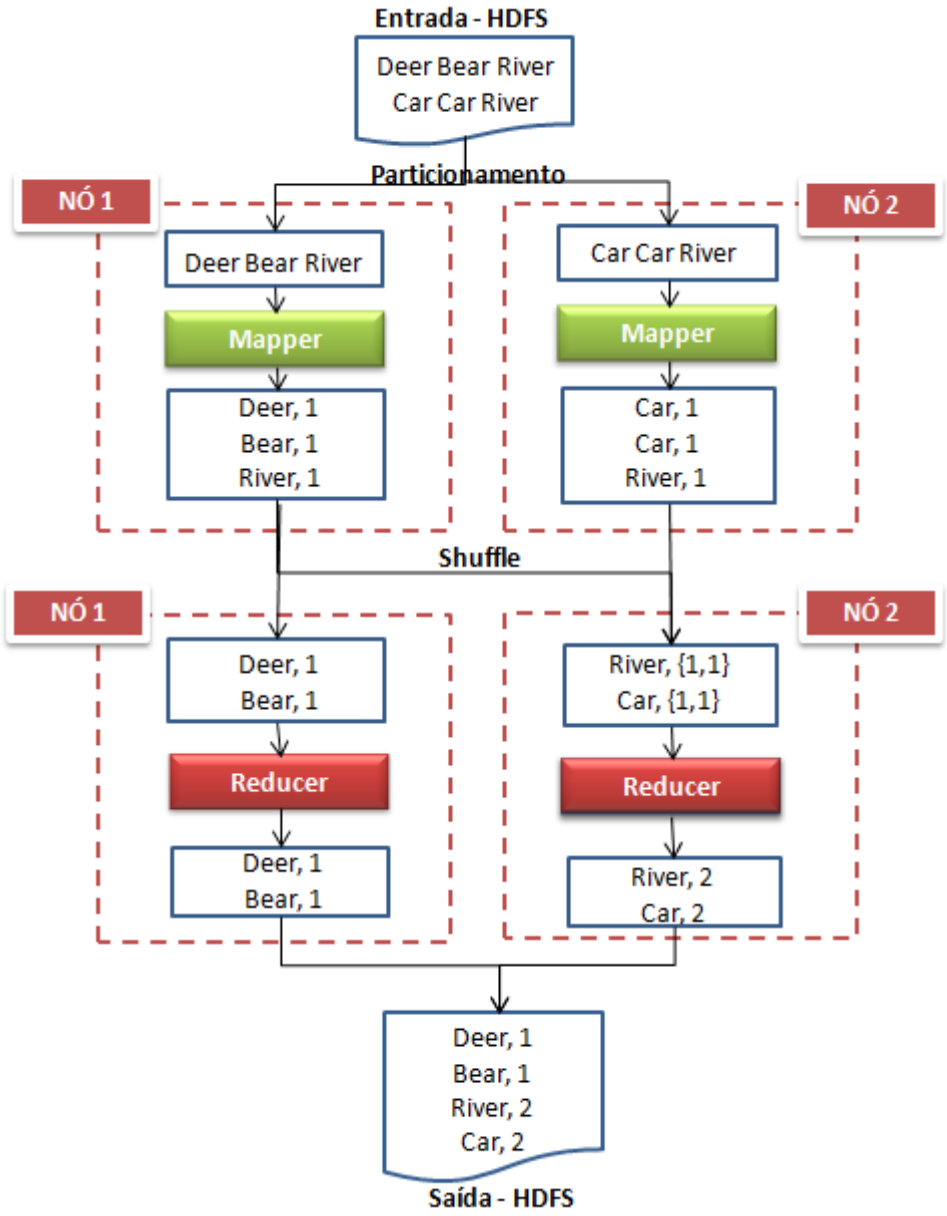

Figura 11: Exemplo *job* MapReduce

4.3 CLOUDERA IMPALA

O Cloudera Impala é um software aberto que permite a execução de consultas interativas sobre conjuntos de dados volumosos armazenados no HDFS em um *cluster* Hadoop. O Impala utiliza a linguagem de consulta Impala SQL, que segue a sintaxe do SQL-92, terceira revisão do padrão SQL. Por utilizar a linguagem SQL permite que os analistas de dados que estão familiarizados com a linguagem trabalhem em cima de grandes conjuntos de dados sem ter que aprender sobre MapReduce ou outras tecnologias Hadoop.

O Impala utiliza Processamento Massivo Paralelo (MPP) para processar as consultas SQL, garantindo um alto desempenho na execução das consultas, retornando resultados tipicamente dentro de segundos ou alguns minutos. Em vez dos diversos minutos ou horas que são muitas vezes necessários para obter resultados na execução de consultas sobre o Apache Hive, que é outra tecnologia de banco de dados baseada em MapReduce, que permite a execução de consultas interativas utilizando a linguagem HiveQL sobre o Hadoop (CLOUDERA, 2016). O Impala integra-se com o banco de metadados do Apache Hive, Hive Metastore (que armazena metadados de suas tabelas e partições), o que permite compartilhar bancos de dados e tabelas entre os dois componentes. O alto nível de integração com o Hive e compatibilidade com a sintaxe HiveQL, permite a utilização tanto do Impala quanto do Hive para criar tabelas, consultas e emitir dados de carga.

Apesar das vantagens oferecidas pelo Impala, especialmente no âmbito da obtenção de melhores resultados de desempenho na execução de consultas, ele ainda apresenta algumas limitações se comparado a outros sistemas de bancos de dados relacionais. Algumas limitações apresentadas pelas versões 2.1.x do Impala são (CLOUDERA, 2016):

- As cláusulas *Data Manipulation Language* (DML) DELETE e UPDATE não são aceitas. As únicas cláusulas DML aceitas são INSERT, LOAD DATA e SELECT.
- Cláusulas *Transaction Control Language* (TCL), tais como COMMIT e ROLLBACK não são aceitas.
- Restrições de chave primária e estrangeira não estão disponíveis, devido ao *overhead* causado por elas em aplicações que lidam com grandes volumes de dados.
- Índices em colunas também não estão disponíveis. Em compensação, estatísticas sobre colunas e tabelas são criadas, para melhorar o desempenho das consultas.
- Não oferece suporte a tipos de dados não-escalares, tais como *lists* ou *maps*.

4.3.1 Arquitetura do Cloudera Impala

A arquitetura do Impala é composta por diferentes componentes que são executados em máquinas específicas dentro de um *cluster* Hadoop (CHAUHAN, 2013; CLOUDERA, 2016). A Figura 12 ilustra a arquitetura do Impala com os seus principais componentes (CHAUHAN, 2013). O Impala Daemon é um componente local em cada nó do *cluster*, fisicamente representado pelo processo *impalad*. Ele é utilizado para processar os dados diretamente onde eles residem dentro do *cluster*, evitando a transferência de dados intensiva. Consultas SQL podem ser submetidos ao Impala Daemon via JDBC, ODBC, Apache Hue ou Impala-shell. Cada Impala Daemon pode aceitar uma consulta SQL e distribuir o trabalho de processamento para outros Impala Daemons no *cluster*. Cada nó retorna seus resultados parciais de volta ao nó coordenador, nó que tenha inicialmente admitido a consulta. Com base nos resultados parciais, o nó coordenador calcula o resultado final e o retorna de volta para o usuário.

O componente Impala Statestore controla o estado dos Impala Daemons em cada nó do *cluster* e transmite continuamente informações para cada um destes Daemons. Ele é executado somente em um nó do *cluster* e é representado fisicamente pelo processo *statestored*. Se um Impala Daemon falha ficando *offline*, o Impala Statestore informa todos os outros Daemons para que consultas futuras evitem fazer pedidos para o nó inacessível. Se o Statestore se tornar inacessível, os Daemons continuam executando e distribuindo o trabalho entre si, como de costume, porém sem saber se um específico Daemon está disponível. Quando o Statestore volta a ficar *online*, ele restabelece a comunicação com os Daemons e retoma a sua função de controle.

O componente Impala Catalog teve um grande impacto sobre o desempenho das avaliações das consultas. Foi introduzido na versão 1.2.2 do Impala e é fisicamente representado por um único processo *catalogd* que roda em somente um nó do *cluster*. O serviço de catálogo distribui automaticamente metadados sobre as tabelas armazenadas no Impala para todos os nós do *cluster*. Sempre que uma tabela é criada, excluída, alterada, atualizada ou tiver dados inseridos, o Impala Catalog transmite estas informações a todos os outros nós. Consequentemente, atualizações manuais de metadados, como REFRESH ou INVALIDATE METADATA - necessárias em versões anteriores - não são mais necessárias no Impala. Os metadados desempenham um importante papel para a otimização de consultas que usam muita memória, como consultas que utilizam *joins*.

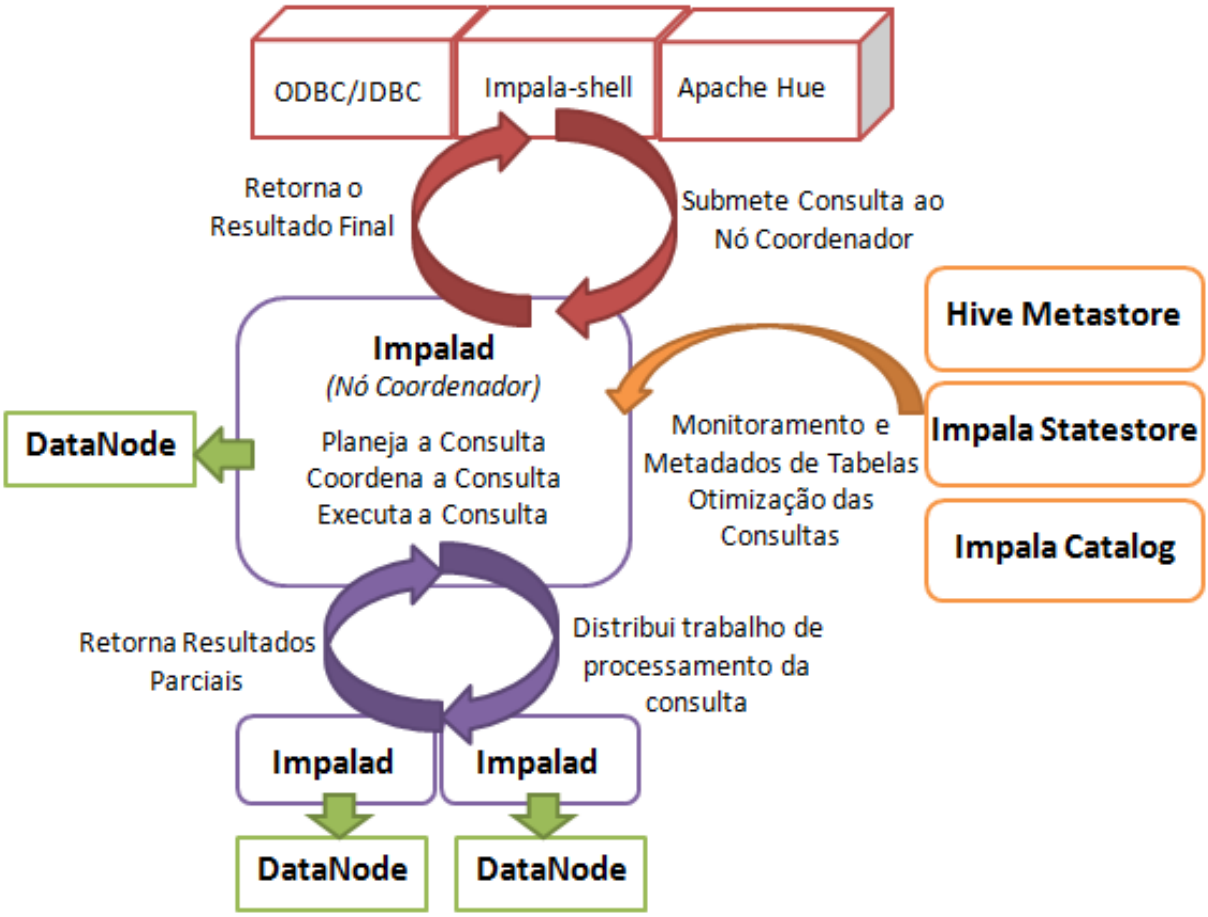

Figura 12: Componentes da arquitetura do Cloudera Impala (Adaptado de CHAUHAN, 2013)

4.3.2 Formatos de arquivos e algoritmos de compressão adotados pelo Cloudera Impala

Cada banco de dados criado no Impala é armazenado em um diretório no HDFS, normalmente dentro do diretório `/user/hive/warehouse`. Arquivos já existentes no HDFS podem ser carregados em tabelas externas no Impala, estes arquivos após serem carregados continuam a residir no mesmo local no HDFS. Também é possível criar tabelas internas no Impala, que dão origem a novos arquivos que são armazenadas em diferentes diretórios no HDFS.

O Impala oferece suporte a diferentes formatos de arquivos de dados utilizados no Apache Hadoop, como Text, Parquet, RCFile e SequenceFile. Também é possível escolher diferentes algoritmos de compressão de arquivos como Snappy, Gzip, Deflate, Bzip2 e LZO (CLOUDERA, 2016). O formato de arquivo e o tipo de compressão escolhidos para uma tabela interna têm consequências significativas no seu desempenho. A compressão afeta o tamanho dos dados no disco e, consequentemente, a quantidade de recursos de entrada/saída e

CPU necessários, podendo ser um fator limitante no desempenho das consultas. Ao comprimir os dados, um menor número total de bytes são transferidos a partir do disco para a memória. Isso reduz a quantidade de tempo necessário para a transferência de dados.

O formato de arquivo Parquet é especificamente recomendado pelo Cloudera (CLOUDERA, 2016) para melhorar o desempenho de consultas analíticas. Este tipo de formato de arquivo utiliza uma abordagem de armazenamento orientada a coluna, que permite acesso direto a dados de uma coluna, sem precisar ler a linha inteira da tabela. Isto diminui a quantidade de recursos de entrada/saída necessários quando precisa-se selecionar apenas algumas colunas particulares de uma tabela. Com isso, a abordagem colunar se mostra vantajosa para consultas que executam operações de agregação, como SUM e AVG, que precisam processar todos ou quase todos os valores de uma coluna específica.

A Figura 13 ilustra como exemplo uma tabela que é armazenada utilizando duas abordagens; uma orientada por linha e outra por coluna. Bancos de dados relacionais tradicionais armazenam dados utilizando a abordagem orientada por linha, armazenando cada linha uma após a outra. Na abordagem colunar, os valores para cada coluna são armazenados em conjunto, como um grupo. Desta forma, quando uma consulta seleciona somente colunas específicas, estas colunas podem ser lidas diretamente ao contrário da outra abordagem, onde toda a linha tem de ser lida para acessar o valor da coluna. No entanto, bancos de dados orientados a linha atingem um melhor desempenho se a consulta seleciona a maior parte das colunas da linha, na qual bancos de dados orientados a coluna tem uma sobrecarga maior ao reconstruir uma linha inteira. Logo, a escolha da abordagem a ser utilizada no banco de dados depende das consultas que serão executadas com frequência e do tamanho geral da tabela.

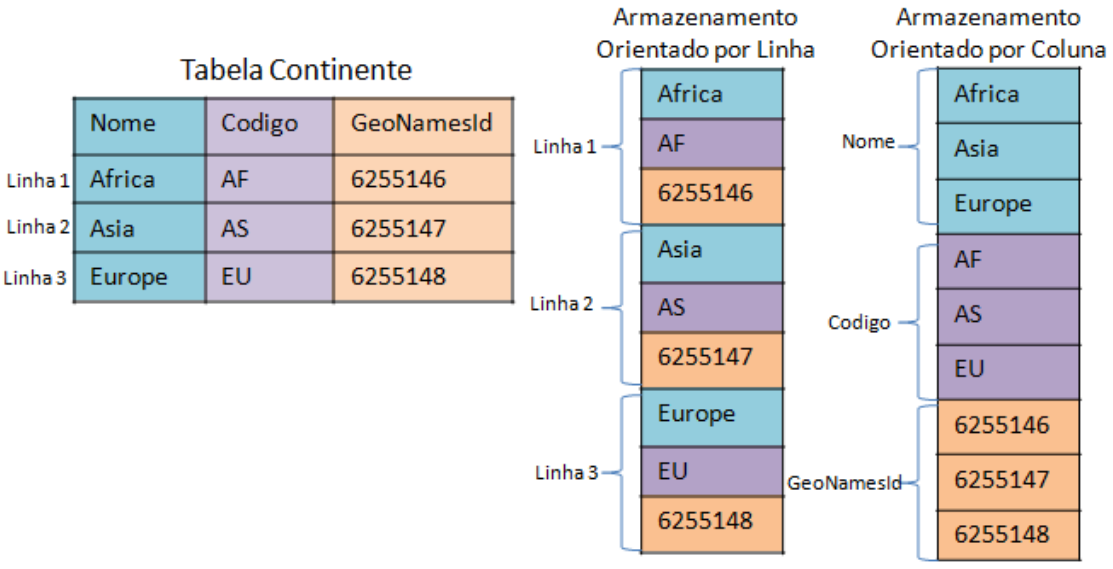

Figura 13: Exemplo de armazenamento orientado por linha e coluna

O Impala armazena os dados de uma tabela em arquivos de tamanho fixo, seguindo a abordagem orientada a coluna. Quando é necessário recuperar valores de uma coluna específica, todos os arquivos são abertos, mas somente as partes relacionada à coluna são lidas.

4.3.3 Melhoria do desempenho das consultas executadas no Cloudera Impala

(CLOUDERA, 2015) cita como principais causas que levam a exceder o limite de memória quando são executadas consultas muito complexas no Impala: 1) Falta de estatísticas (metadados); 2) Grande quantidade de *joins* dentro de uma única consulta; 3) *Joins* entre tabelas muito grandes; 4) *Group by* gigante. Diante destes casos, o Impala oferece diversas possibilidades para otimizar consultas que consomem bastante memória, tais como (CLOUDERA, 2016):

- *Computação de Estatísticas*: É possível computar estatísticas tanto para a tabela quanto para as colunas da tabela através da instrução COMPUTE STATS. As estatísticas da tabela computam o seu número de linhas, arquivos de dados, tamanho e formato dos arquivos. Caso a tabela possua partições, as estatísticas são mostradas por partição. As estatísticas de coluna computam o tipo de coluna, o número de valores distintos em cada coluna, número de valores nulos, seu tamanho máximo e médio. Os metadados recolhidos permitem que o Impala otimize automaticamente as consultas.
- *Automatização da ordenação dos Joins*: Após recolher as estatísticas sobre cada tabela envolvida em um *Join* dentro de uma consulta, o Impala, a partir da versão 1.2.2, é capaz de otimizar a consulta com base no tamanho da tabela, número de valores distintos em cada coluna, e assim por diante. Se o usuário optar por ordenar as tabelas no *Join* de forma manual, substituindo a otimização automática, basta utilizar a palavra STRAIGHT_JOIN logo após a cláusula SELECT. Durante a ordenação manual das tabelas, o Impala aconselha carregar a maior tabela primeiro e depois juntá-la aos restantes das tabelas, em ordem ascendente de tamanho (exemplo: TABELA GRANDE JOIN TABELA MINÚSCULA, PEQUENA, MÉDIA).
- *Diferentes técnicas para realização de Joins*: O planejador de consultas do Impala escolhe entre diferentes técnicas para a realização junções em consultas, dependendo dos tamanhos absolutos e relativos das tabelas. A técnica *broadcast* é

o padrão, onde a tabela do lado direito é considerada menor do que a tabela do lado esquerdo, e os seus conteúdos são enviados para todos os nós envolvidos na consulta. A técnica alternativa é conhecida como *shuffle*, e é mais apropriada para junção de tabelas grandes de tamanho aproximadamente igual. Com esta técnica, porções de cada tabela são enviados para cada nó do *cluster* para que estes subconjuntos de linhas sejam processados em paralelo. Caso as estatísticas das tabelas estejam disponíveis, o Impala escolhe a técnica mais apropriada para cada *join* presente na consulta. Caso as estatísticas não estejam disponíveis, o Impala escolhe a técnica *broadcast* por padrão. É possível saber qual estratégia de associação é usado para uma consulta particular digitando a palavra EXPLAIN antes da cláusula SELECT. Se o próprio usuário preferir escolher qual técnica quer utilizar em cada junção, basta adicionar as palavras [BROADCAST] ou [SHUFFLE] após as cláusulas JOIN.

- *Partições*: No Impala, por padrão, todos os arquivos de dados para uma tabela são armazenados em um único diretório. O particionamento é uma técnica de fragmentação horizontal utilizada para dividir fisicamente os dados durante a carga, com base em valores de uma ou mais colunas, para acelerar as consultas que testam estas colunas. Por exemplo, em uma tabela de tempo particionada por ano, há um diretório de dados separado para cada valor de ano diferente. Uma consulta que inclui uma condição como ANO = 2013 irá examinar apenas os arquivos de dados do diretório correspondente ao ano 2013, reduzindo a quantidade de dados para ler. As partições devem ser identificadas a partir de casos de uso e consultas SQL a serem executadas em cima das tabelas.

4.4 *CLUSTER* MONTADO

Para rodar a aplicação distribuída desenvolvida nesta dissertação foi montado um *cluster*, utilizando o Cloudera Distributed Hadoop[27] (CDH) versão 5.3.2, versão 2.5.0 do Apache Hadoop. O CDH é uma distribuição aberta do Apache Hadoop provida pelo Cloudera Inc.

O *cluster* é composto por quatro máquinas com configuração de 7.6GB RAM, Intel Core I7-4770 (4 *cores*), 1TB Hard Disk e sistema operacional CentOS 6.5. As máquinas se

[27] http://www.cloudera.com/content/www/en-us/products.html

conectam através da rede a partir do protocolo SSH. Os serviços do Apache Hadoop utilizados no *cluster* CDH foram:

- HDFS: Sistema de arquivos distribuído.
- Cloudera Impala: Ferramenta para consultas SQL analíticas sobre dados armazenados no Hadoop.
- Apache Hive: Para o Impala funcionar é necessário instalar o Hive Metastore, que tem seu uso compartilhado entre o Hive e o Impala, e cuja função é armazenar metadados sobre os objetos do esquema de dados, tais como tabelas e partições.
- Apache Yarn (Map Reduce 2.0): Evolução do MapReduce onde as funções do JobTracker são repartidas em processos independentes. Uma das funções principais do Yarn é a de partilhar os dados utilizados nas tarefas de Map e Reduce entre os nós específicos do *cluster*, outra função importante é gerenciar as falhas e procurar nós disponíveis para executar a tarefa onde houve falha. Os componentes do Yarn são ResourceManager, rodado em um único nó, que faz a gerência dos recursos que são concedidos na forma de *containers*, JobHistory, rodado em um único nó, que registra informações sobre os *jobs* MapReduce executados e NodeManager, rodados em todos os nós do *cluster*, responsável pelos *containers* e por monitorar a utilização dos recursos a reportando ao ResourceManger.
- Apache Hue[28]: Conjunto de aplicações web que permitem a interação com o *cluster* CDH. Seus aplicativos permitem a navegação pelo HDFS, execução de consultas no Hive e no Impala, monitoramento de *jobs* MapReduce, entre outras funções.
- Cloudera Manager[29]: Ferramenta para gestão e monitoramento do *cluster* CDH.

A Figura 14 ilustra como os componentes da arquitetura Hadoop foram organizados entre os nós do *cluster* CDH.

[28] http://gethue.com/

[29] https://www.cloudera.com/content/www/en-us/products/cloudera-manager.html

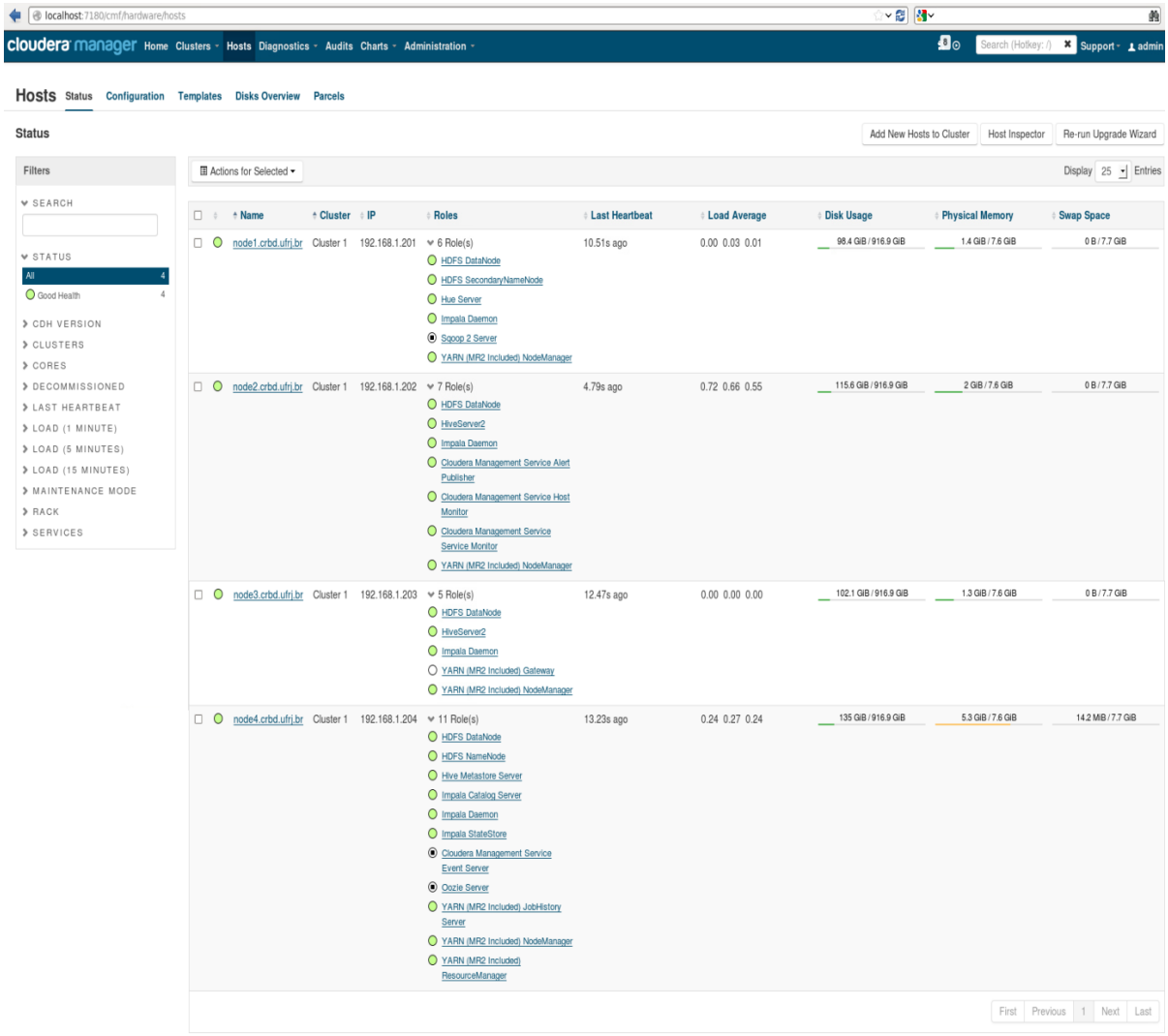

Figura 14: Nós do cluster CDH e seus componentes

5 PROPOSTA DE SOLUÇÃO

Neste capítulo apresentamos uma proposta de solução para o armazenamento de grandes volumes de dados multidimensionais no formato de grafo RDF e avaliação eficiente de consultas analíticas SPARQL sobre estes dados. Na Seção 5.1 é apresentada uma visão geral da arquitetura da aplicação *Impala MultiLOD*, desenvolvida nesta dissertação. Nas Seções 5.2 e 5.3 é explicado detalhadamente o funcionamento dos dois componente da arquitetura da aplicação, o primeiro responsável pela carga e preparação do banco de dados e o segundo responsável pelo acesso aos dados armazenados no banco.

5.1 VISÃO GERAL DA ARQUITETURA DA APLICAÇÃO *IMPALA MULTILOD*

A Figura 15 ilustra uma visão da arquitetura da aplicação *Impala MultiLOD*, desenvolvida nesta dissertação, que é composta por dois componentes. O primeiro componente é responsável por preparar dados na forma de triplas RDF para serem carregados em tabelas no Cloudera Impala. Para isso, foi utilizado um esquema de armazenamento que permite modelar e armazenar triplas RDF em tabelas. Os dados na forma de triplas RDF, que servem de entrada para o primeiro componente, são extraídos de suas fontes de origem, triplificados e armazenados em um arquivo RDF em uma etapa de pré-processamento, que é explicada com mais detalhes no Capítulo 6 - capítulo em que são detalhados os experimentos realizados sobre a aplicação.

O segundo componente é responsável pelo acesso dinâmico aos dados armazenados no Impala. O acesso é feito a partir de consultas na linguagem SPARQL, geradas pelos usuários finais da aplicação a partir de uma interface gráfica, que são traduzidas em consultas SQL equivalentes, executadas diretamente sobre as tabelas no Impala. A conexão com o Impala é feita a partir de um *driver* JDBC. O resultado das consultas é exibido na forma de triplas RDF para os usuários finais, já que os dados são armazenados utilizando o modelo RDF.

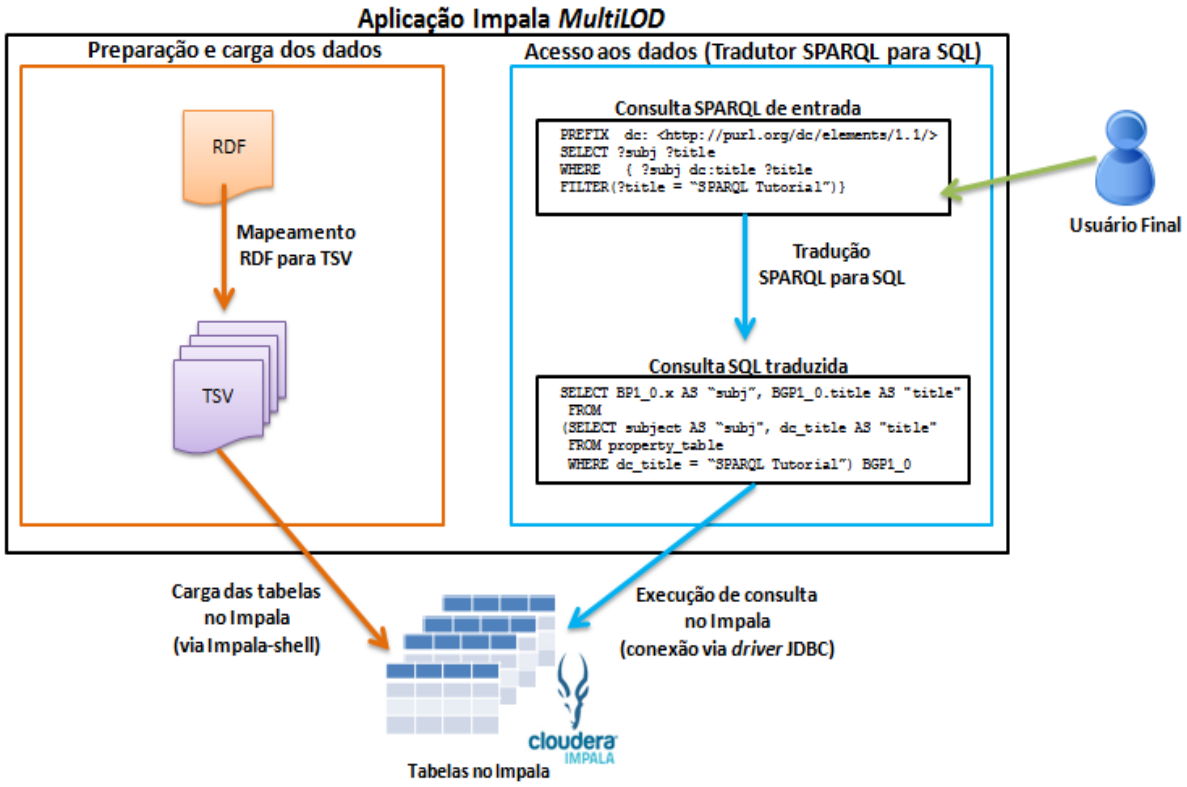

Figura 15: Visão geral da arquitetura da aplicação *Impala MultiLOD*

Nas Seções 5.2 e 5.3 é feita a explicação da implementação de cada componente da arquitetura da aplicação, sendo detalhadas as ferramentas e plataformas utilizadas.

5.2 1º COMPONENTE (CARGA E PREPARAÇÃO DO BANCO DE DADOS)

Nesta seção é apresentado o primeiro componente da aplicação Impala *MultiLOD*, responsável pelo processo de carga dos dados RDF em tabelas no Impala, banco de dados analítico, baseado no *framework* Apache Hadoop. A Figura 16 ilustra uma visão geral de como é realizado o processo de carga dos dados.

O primeiro passo do processo de carga (detalhado na Seção 5.2.1) consiste em escolher o esquema de armazenamento mais adequado para armazenar dados RDF em tabelas para um dado domínio. O segundo passo (detalhado na Seção 5.2.2) envolve o desenvolvimento de algoritmos MapReduce para mapear o arquivo RDF de entrada, localizado no HDFS, em tabelas no formato de arquivos TSV (*tab-separated values*), seguindo o esquema de armazenamento escolhido no passo anterior. Os arquivos TSV gerados são armazenados diretamente no HDFS. Uma vez que os arquivos TSV tenham sido criados, no passo final (detalhado na Seção 5.2.3), tabelas correspondentes podem ser criadas no Impala e os dados carregados.

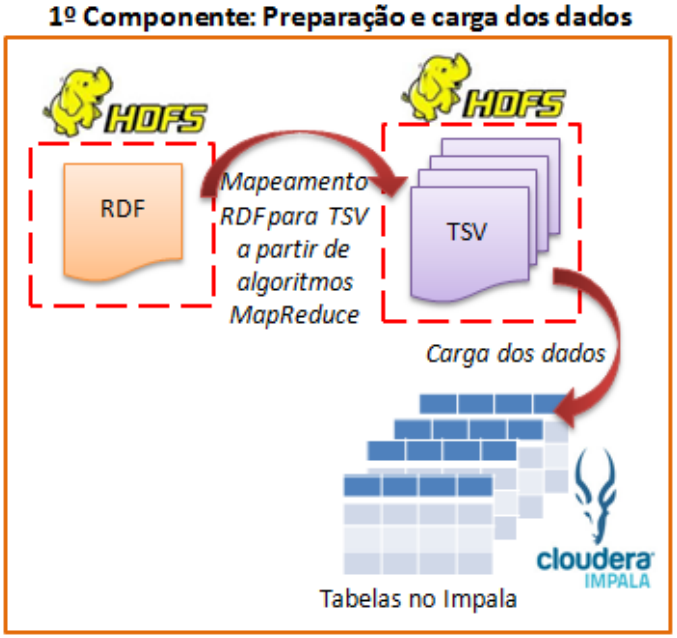

Figura 16: Visão geral do processo de carga de dados RDF em tabelas no Impala

5.2.1 Escolha do esquema de armazenamento de triplas RDF em tabelas

Antes de realizar a implementação dos algoritmos MapReduce que convertem um arquivo RDF em tabelas no formato de arquivo TSV, foi necessário escolher qual esquema de armazenamento seria adotado nas tabelas.

A escolha do melhor esquema de armazenamento teve que ser baseada não só na natureza dos dados a serem trabalhados na aplicação, no caso os dados do projeto PingER, que possuem características multidimensionais, mas também no fato que o Impala trabalha de forma distribuída e utiliza a abordagem orientada por coluna para armazenar os dados de uma tabela interna em arquivos no formato Parquet. Estes fatores divergem bastante das características dos repositórios de triplas apoiados por SGBDR tradicionais, que costumam ser utilizados de forma centralizada e utilizam a abordagem orientada por linha para armazenar os dados de uma tabela.

Assim, foram considerados três esquemas que modelam e armazenam triplas RDF em tabelas, já propostos na literatura e utilizados por alguns repositórios de triplas apoiados por SGBDR:

(a) Tabela de Triplas

A abordagem de Tabela de Triplas (CHONG et al., 2005) é adotada por alguns repositórios de triplas apoiados por SGBDR tais como 3Store (HARRIS, GIBBINS, 2003). Utiliza como esquema uma única tabela com três atributos (Sujeito, Propriedade, Objeto),

contendo uma linha para cada tripla. A Figura 17 ilustra um exemplo de Tabela de Triplas. Este esquema de dados, apesar de ser flexível e ter um *design* simples, tem o potencial para gerar graves problemas de desempenho na execução de consultas, já que há apenas uma única tabela e uma consulta é capaz de envolver vários *self-joins* sobre a tabela. O *self-join* é utilizado para unir uma tabela a ela mesma, para isso é necessário renomear temporariamente pelo menos uma tabela na consulta SQL. Por exemplo, a consulta SPARQL abaixo procura o nome da cidade cuja área mede 1798 km^2 e sua população é igual a 4205961:

```
SELECT ?name
WHERE { ?town gn-ont:population “4205961” .
?town MGC:areaInSqKm “1798” .
?town PingER-ont:name ?name }
```

Tal consulta seria traduzida para a consulta SQL abaixo e seria rodada sobre a tabela ilustrada na Figura 17.

```
SELECT C.objeto
FROM TAB_TRIPLAS AS A,
TAB_TRIPLAS AS B,
TAB_TRIPLAS AS C
WHERE A.sujeito = B.sujeito
AND B.sujeito = C.sujeito
AND A.propriedade = "gn-ont:population"
AND A.objeto = "4205961"
AND B.propriedade = "MGC:areaInSqKm"
AND B.objeto = "1798"
AND C.propriedade = "PingER-ont:name"
```

Sujeito	Propriedade	Objeto
URI1	rdf:type	Country
URI1	MGC:countryCode	“SA”
URI1	gn-ont:population	“25731776”
URI1	MGC:areaInSqKm	“1960582”
URI2	rdf:type	Town
URI2	PingER-ont:name	“Riyadh”
URI2	gn-ont:population	“4205961”
URI2	MGC:areaInSqKm	“1798”
URI3	rdf:type	State
URI3	PingER-ont:name	“Najran”
URI3	gn-ont:population	“505652”

Figura 17: Exemplo de Tabela de Triplas

É possível notar que a simples consulta SPARQL ilustrada no exemplo resultaria na execução de três *self-joins* sobre a tabela de triplas. Consultas do mundo real, especialmente consultas OLAP, utilizam muito mais junções, além de outra operações como agregações,

uniões e filtros, logo as repostas a esse tipo de consulta provavelmente apresentariam alta latência.

Para o experimento com os dados do PingER esse tipo de esquema de armazenamento se mostraria bastante ineficiente, na medida que se pretende utilizar amostras de dados muito volumosas, com mais de 100 milhões de triplas, o que geraria tabelas com mais de 100 milhões de linhas. Qualquer *self-join* executado em uma consulta sobre a tabela, representaria uma operação de junção entre tabelas muito grandes, o que facilmente excederia o limite de memória RAM disponível em cada nó do *cluster*.

Além disso, consultas realizadas sobre uma Tabela de Triplas necessitam selecionar a maior parte das colunas de uma linha para identificar as triplas na tabela, atingindo um melhor desempenho em bancos de dados orientados a linha. O que não é o caso do Impala, que utiliza a abordagem orientada a coluna e geraria uma sobrecarga maior no desempenho para recuperar e reconstruir os dados de uma linha inteira.

Outra desvantagem da Tabela de Triplas é que obrigatoriamente o tipo do atributo Objeto na tabela deve ser o mesmo para todos os valores de objetos das triplas, normalmente um Varchar, sendo adotado o tamanho do maior Varchar existente para o campo. Este fato pode levar a um mau desempenho nas consultas e consumo excessivo de espaço.

(b) Particionamento Vertical

A abordagem de Particionamento Vertical foi proposta em (ABADI et al., 2007). Nela são criadas tabelas para cada propriedade presente no documento RDF. Em cada uma, a primeira coluna contém o sujeito que define a propriedade e a segunda coluna contém o objeto, valor para a propriedade. Cada tabela é ordenada por sujeito, de modo que sujeitos específicos possam ser localizados rapidamente. A Figura 18 ilustra como a tabela da Figura 17 poderia ser representada utilizando o Particionamento Vertical.

rdf:type

Sujeito	Objeto
URI1	Country
URI2	Town
URI3	State

gn-ont:population

Sujeito	Objeto
URI1	“25731776”
URI2	“4205961”
URI3	“505652”

PingER-ont:name

Sujeito	Objeto
URI2	“Riyadh”
URI3	“Najran”

MGC:arealnSqKm

Sujeito	Objeto
URI1	“1960582”
URI2	“1798”

MGC:countryCode

Sujeito	Objeto
URI1	“SA”

Figura 18: Exemplos de Tabelas de Particionamento Vertical

As vantagens da abordagem de Particionamento Vertical são: i) Suporte a atributos multi-valorados. Se um sujeito tem mais do que um valor de objeto para uma propriedade particular, cada valor distinto é listado em uma linha sucessiva na tabela para essa propriedade. ii) Sujeitos que não definem uma propriedade particular são simplesmente omitidos da tabela para essa propriedade, ou seja, não é necessário representar explicitamente valores nulos. iii) Somente as propriedades acessadas pela consulta precisam ser lidas.

Apesar das vantagens, também existem algumas desvantagens. Dependendo do número de propriedades distintas num documento RDF, podem ser criadas muitas tabelas de tamanho pequeno. Além disso, o processo de carga dos dados pode ser mais lento, na medida que várias tabelas precisam ser acessados para declarações sobre o mesmo sujeito. Estas desvantagens já tornam o seu uso inapropriado para conjuntos de dados multidimensionais onde múltiplas propriedades estejam presentes. Particularmente, o conjunto de dados do PingER possui 68 propriedades distintas definidas, o que levaria à criação de 68 tabelas, muitas com um tamanho bastante reduzido. Logo, as consultas a serem construídas teriam que ter muitas junções entre tabelas, o que provavelmente causaria danos ao seu desempenho.

(c) Tabela Proprietária

A abordagem de Tabela Proprietária foi proposta em (WILKINSON, 2006), onde foi utilizado o *framework* Jena2, que considera o uso de tabelas proprietárias para melhorar o desempenho de consultas sobre repositórios de triplas. Nele, foram propostos dois tipos de tabelas proprietárias. O primeiro tipo, nomeado como tabela proprietária clusterizada, propõe a criação de tabelas com aglomerados de propriedades que tendem a ser definidas em conjunto. Por exemplo, para a tabela da Figura 17 `PingER-ont:name`, `gn-ont:population` e `MGC:areaInSqKm` tendem a ser definidos como propriedades para sujeitos semelhantes. Assim, uma tabela contendo estas três propriedades como atributos, ilustrada na Figura 19, juntamente com o sujeito como a chave da tabela, pode ser criada, armazenando as triplas da tabela original cuja propriedade é um destes três atributos. Várias tabelas proprietárias com diferentes conjuntos de propriedades podem ser criados; no entanto, um requisito fundamental para este tipo de tabela é que uma propriedade particular só pode aparecer em no máximo uma tabela proprietária.

Sujeito	PingER-ont:name	gn-ont:population	MGC:arealnSqKm
URI1	NULL	"25731776"	"1960582"
URI2	"Riyadh"	"4205961"	"1798"
URI3	"Najran"	"505652"	NULL

Figura 19: Exemplo de Tabela Proprietária Clusterizada

O segundo tipo de tabela proprietária, nomeado como tabela proprietária de classe (*property-class table*), explora os tipos dos sujeitos, agrupando sujeitos de uma mesma classe juntos na mesma tabela. Os tipos dos sujeitos são dados pela propriedade `rdf:type`. Como efeito, todas as tabelas armazenam o atributo `rdf:type` para indicar a classe que está sendo representada pela tabela. Além do `rdf:type` e do sujeito como chave, a tabela proprietária também pode armazenar outras propriedades ligadas aos sujeitos da classe como atributos. Por exemplo, para a tabela da Figura 17, sujeitos com o tipo "Country" seriam armazenados em uma primeira tabela, conforme ilustrada na Figura 20, sujeitos com o tipo "Town" em uma segunda tabela e sujeitos com o tipo "State" em uma terceira tabela.

Classe: Country

Sujeito	rdf:type	MGC:countryCode	gn-ont:population	MGC:arealnSqKm
URI1	Country	"SA"	"25731776"	"1960582"
URI2	Country	"LK"	"21513990"	"65610"
URI3	Country	"IN"	"1173108018"	"3287590"

Figura 20: Exemplo de Tabela Proprietária de Classe para triplas cujo sujeito é da classe "Country"

Ao contrário do primeiro tipo de tabela proprietária proposto, uma propriedade pode existir como atributo em várias tabelas proprietárias de classe. Uma limitação atual é que uma tabela proprietária de classe armazena apenas instâncias de uma única classe. No entanto, isso poderia ser relaxado no futuro para permitir que uma única tabela armazene subclasses de uma classe comum.

A abordagem de tabela proprietária de classe é especialmente eficaz para consultas que buscam por múltiplas propriedades de um único sujeito específico, as quais são chamadas *star-queries*. Por exemplo, para a consulta SPARQL expostas anteriormente nesta seção que procura o nome da cidade cuja área mede 1798 km^2 e sua população é igual a 4205961, que quando convertida para a linguagem SQL resultava na execução de três *self-joins* sobre a

tabela de triplas, utilizando a abordagem de tabela proprietária de classe, o seu resultado seria facilmente obtido através de uma simples seleção sobre a tabela que guarda as propriedades dos sujeitos da classe "Town", sem necessitar realizar nenhuma operação de junção.

Além disso, esta abordagem também é eficaz para tipos de armazenamento que utilizam a abordagem colunar, como é feito pelo Impala. Uma vez que a abordagem colunar é mais adequada para armazenar tabelas largas, com muitas colunas e menos linhas, permitindo acesso direto ao valor de uma coluna, sem precisar ler a linha inteira, características das tabelas proprietárias.

Esta abordagem também possui algumas desvantagens. A primeira é que como nem todas as propriedades serão definidas para todas os sujeitos, tabelas largas terão (possivelmente) muito valores nulos. Para tabelas muito largas com muitos atributos esparsos, a sobrecarga de espaço destes valores nulos pode potencialmente dominar o espaço dos dados. Particularmente, no caso dos dados do PingER e de outros conjuntos de dados multidimensionais, a ocorrência de valores nulos nas tabelas praticamente não existiria, pois os dados multidimensionais possuem a característica de serem muito bem estruturados, fazendo com que todos os sujeitos pertencentes a uma classe específica possuam definidas as propriedades relacionadas a esta classe.

A segunda desvantagem seria diante a existência de atributos multivalorados (como um livro com múltiplos autores) que são mais difíceis de serem expressos nesta representação. Uma forma de tratá-los seria para cada valor de um determinado atributo multi-valorado, uma linha duplicada é adicionada, contendo todos os mesmos valores das outras colunas, o que levaria a uma perda maior de espaço. No caso dos dados do PingER existem poucos casos de atributos multivalorados, que não ocasionariam um grande aumento no espaço ocupado pelos dados.

A terceira desvantagem seria a pouca flexibilidade deste tipo de esquema, já que antes da tabela ser criada é necessário conhecer todos as possíveis propriedades presentes no documento RDF. Entretanto, para domínios que têm uma ontologia de domínio definida, geralmente é possível identificar claramente as classes de indivíduos e suas propriedades associadas, o que torna a criação de tabelas muito mais fácil.

O Quadro 3 apresenta uma visão resumida dos pontos apresentados anteriormente para cada alternativa de armazenamento estudada.

Quadro 3: Vantagens e desvantagens dos esquemas de armazenamento levantados

Tipo de esquema de armazenamento	Vantagens	Desvantagens
Tabela de Triplas	• Esquema flexível • *Design* simples • Não representa explicitamente valores nulos na tabela	• Única tabela com muitas linhas • Consultas podem envolver a execução de vários *self-joins* • Ideal para banco de dados orientados a linha • Tipo do atributo Objeto deve ser o mesmo para todos os valores de objetos das triplas
Particionamento Vertical	• Não representa explicitamente valores nulos nas tabelas • Somente as propriedades acessadas pela consulta precisam ser lidas	• Podem ser geradas muitas tabelas de tamanho pequeno • Processo de carga dos dados mais lento • Consultas podem envolver muitas junções entre tabelas
Tabela Proprietária	• Eficaz para consultas do tipo *star-queries* • Ideal para banco de dados orientados a coluna, caso do Impala • Número de junções entre tabelas dentro de uma consulta é bem menor	• Modelagem mais complexa • Esquema pouco flexível • Necessidade de representar explicitamente valores nulos nas tabelas • Atributos multi-valorados representados a partir de linhas duplicadas na tabela

Diante do exposto, chega-se à conclusão que o esquema de armazenamento mais adequado para modelar dados RDF com características multidimensionais para processamento analítico e armazenamento no Impala seria o de Tabela Proprietária, adotando a segunda abordagem proposta em (WILKINSON, 2006) de Tabela Proprietária de Classe. Pois, apesar de possuir uma modelagem mais complexa e menos flexível em relação aos outros esquemas estudados, é capaz de melhorar significativamente o desempenho de consultas. Ao adotar tabelas largas, agrupando sujeitos de uma mesma classe em uma mesma tabela, o número de junções entre tabelas dentro de uma consulta é menor, sendo possível responder consultas muitas vezes de forma direta. Além de ser o esquema mais apropriado para tipos de armazenamento que utilizam a abordagem colunar, a qual é adotada pelo Impala.

5.2.2 Mapeamento dos dados RDF em tabelas a partir de algoritmos MapReduce

Após ser escolhido o esquema de armazenamento a ser adotado nas tabelas, foi possível desenvolver os algoritmos (*jobs*) MapReduce na linguagem JAVA que realizam o mapeamento de arquivos RDF, armazenados no HDFS, em tabelas proprietárias de classe representadas no formato de arquivos TSV, que após serem gerados são armazenadas diretamente no HDFS.

Foi necessário desenvolver dois tipos de *jobs* MapReduce para realizar o mapeamento de um documento RDF em tabelas proprietárias de classe. Antes de criar as tabelas, uma para cada classe de indivíduo presente na ontologia, é necessário conhecer todas propriedades (predicados) relacionados a esta classe e os tipos dessas propriedades, caso o objeto seja expresso na forma de um literal. Para isso, em uma fase de pré-processamento, para cada tabela de classe foi criado um *job* MapReduce que realiza uma leitura de todo o documento RDF e escreve, como saída, uma lista, armazenada no HDFS, com todos os predicados distintos ligados à classe tratada na tabela e seus tipos (caso fossem definidos).

Na fase Map é implementado um *parser* em cada linha do documento RDF, retornando a tripla armazenada na linha. É verificado se o predicado da tripla é um dos predicados associados à classe da tabela a ser criada, caso seja, o nome do predicado e o tipo do literal na posição do objeto (se disponível) são emitidos como chave de saída (MapOutputKey) e valor de saída (MapOutputValue), respectivamente. Por exemplo, o tipo do literal "Canada"^^xsd:string é string. Na fase Shuffle, os pares gerados na fase Map são ordenados e agrupados de acordo com a chave (predicado). Na fase Reduce, o predicado associado à chave de entrada e o primeiro valor associado ao predicado são emitidos como chave de saída (ReduceOutputKey) e valor de saída (ReduceOutputValue), respectivamente, assumindo com isso que todos os objetos associados ao predicado tratado são do mesmo tipo. Caso o tipo dos literais não tenha sido definido é retornado como valor de saída a string "UNDEFINED".

A Figura 21 ilustra as listas geradas no HDFS para a tabela que trata a classe País (*Country*) na ontologia do PingER. No caso foram geradas quatro listas, armazenadas em um mesmo diretório no HDFS, devido ao *job* MapReduce ter sido rodado com quatro instâncias da fase Reduce trabalhando de forma distribuída e cada uma gerou um arquivo diferente (nomeados como part-r-00000 até part-r-00003), contendo uma parte do resultado final do *job*. É possível notar que os tipos dos literais no arquivo RDF dos dados do PingER não foram definidos, sendo retornando a palavra "UNDEFINED".

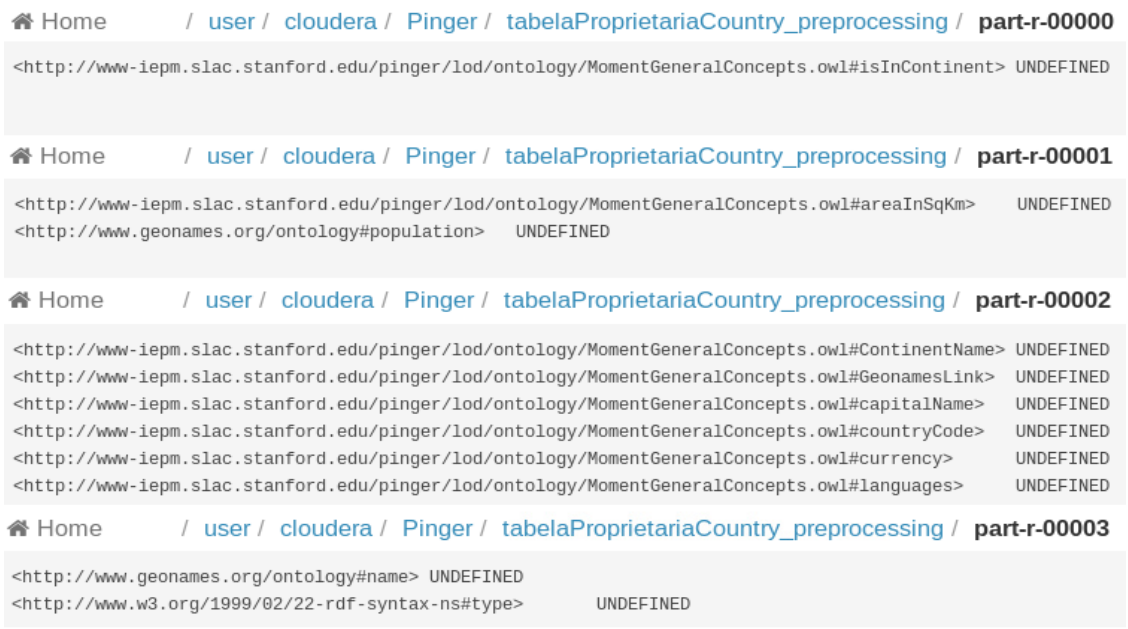

Figura 21: Listas geradas para a Tabela Proprietária de Classe "Country" na fase de pré-processamento

Depois de ser implementada a fase de pré-processamento, foi implementado o *job* MapReduce que realiza a conversão do arquivo RDF em tabelas proprietárias de classe, utilizando como base os arquivos com a lista dos predicados associados às classes, gerados na fase anterior. O resultado deste *job* é a geração de várias tabelas proprietárias de classe no formato de arquivos TSV armazenados no HDFS.

Neste *job*, antes do método Map ser chamado, é feita uma chamada única ao método Setup. Em um *job* MapReduce os métodos Setup e CleanUp são usados para inicialização e limpeza a nível de tarefa. Dentro de uma tarefa, as inicializações acontecem com uma única chamada ao método Setup e, em seguida, todas as chamadas aos métodos Map e Reduce são feitas. No final, uma outra chamada única é feita ao método CleanUp antes de ser finalizada a tarefa. No nosso caso, no método Setup (executado somente uma vez) é implementado um *parser* sobre o documento RDF buscando por todos os sujeitos distintos associados à classe tratada na tabela, para isso é feita uma busca por triplas com o predicado `rdf:type` e com o objeto indicando a classe procurada. Todos os sujeitos encontrados tiveram sua URI armazenada em um *array*.

Na fase Map é implementado um *parser* sobre o documento RDF buscando por todas as triplas cujo sujeito é um dos sujeitos presentes no *array* gerado na fase de Setup, com isso somente serão trabalhadas as triplas associadas à classe tratada na tabela. O sujeito e predicado (sujeito, predicado) das triplas selecionadas irão compor a chave composta de saída (MapOutputKey) da fase Map e o predicado e objeto (predicado, objeto) irão compor o valor

composto de saída (MapOutputValue). Na fase Shuffle, os valores de saída da fase Map são agrupados pelo valor do sujeito e ordenados no grupo de acordo com o valor do predicado.

Em cada chamada a fase Reduce é recebido um sujeito e todos seus predicados, além dos objetos associados. Todos os objetos são agrupados de acordo com o predicado associado. No caso de predicados multivalorados, irão existir mais de um objeto associado ao predicado no grupo. Os valores dos predicados multivalorados devem ser cruzados com os outros valores das outras colunas da tabela, criando linhas duplicada na tabela, contendo todos os mesmos valores das outras colunas. Assim, o produto cartesiano (*cross product*) de cada grupo de predicado deve ser calculado para formar as linhas da tabela.

Neste ponto, a lista ordenada de predicados distintos associados à classe tratada na tabela, armazenada no HDFS, gerada na fase de pré-processamento, é recuperada no método Setup. As colunas da tabela proprietária devem seguir a mesma ordem em que os predicados aparecem na lista. Para cada predicado na lista ordenada, é verificado se existe um objeto no grupo de objetos associado ao predicado. Caso não exista, o valor nulo "NULL" é anexado à linha resultante. Caso exista, o valor do objeto é anexado à linha resultante. Como resultado, cada linha emitida no arquivo TSV segue a mesma ordem definida pela lista ordenada de predicados. O arquivo TSV final gerado é armazenado no HDFS.

A Figura 22 ilustra um exemplo de uma chamada ao método Reduce. A chamada ao método gera todas as linhas da tabela proprietária País (*Country*) para o sujeito MGC:Canada, que compõe a primeira parte da chave composta de entrada (sujeito, predicado) da fase Reduce. Entre os predicados associados a este sujeito está o predicado MGC:languages que possui múltiplos valores associados a ele. Para os outros predicados associados ao sujeito (MGC:ContinentName, MGC:GeonamesLink, MGC:areaInSqKm, MGC:capitalName, MGC:countryCode, MGC:currency, MGC:isInContinent, gn-ont:name, gn-ont:population, rdf:type) somente há um único valor associado a eles. Os valores são agrupados em onze grupos de acordo com o predicado a qual estão associados. Após o cálculo do produto cartesiano são geradas duas linhas na tabela proprietária para o sujeito MGC:Canada, que possuem dois valores diferentes para o atributo MGC:languages e os mesmos valores para os outros atributos.

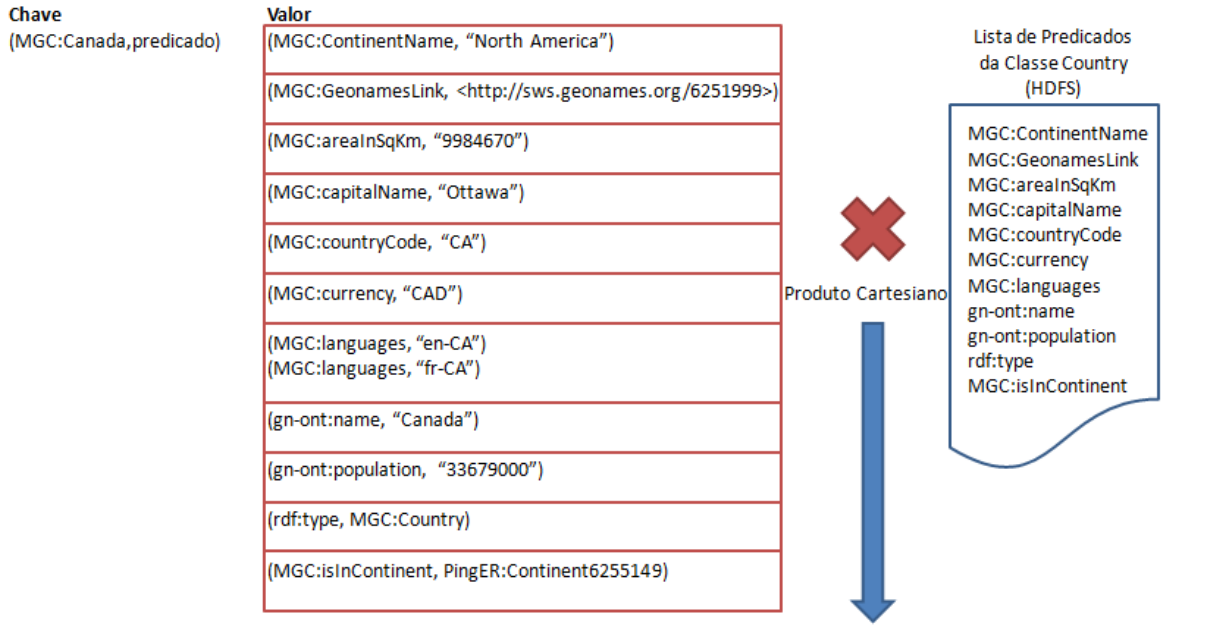

Tabela da Classe Country

Sujeito	MGC: Continent Name	MGC: Geonames Link	MGC: areaInSqKm	MGC: Capital Name	MGC: Country Code	MGC: currency	MGC: languages	gn-ont: name	gn-ont: population	rdf:type	MGC: isIn Continent
MGC: Canada	"North America"	<http://sws.geonames.org/6251999>	"9984670"	"Ottawa"	"CA"	"CAD"	"en-CA"	"Canada"	"33679000"	MGC: Country	PingER: Continent 6255149
MGC: Canada	"North America"	<http://sws.geonames.org/6251999>	"9984670"	"Ottawa"	"CA"	"CAD"	"fr-CA"	"Canada"	"33679000"	MGC: Country	PingER: Continent 6255149

Figura 22: Chamada ao método Reduce para gerar Tabela Proprietária da Classe "Country"

Caso o esquema de dados escolhido não tivesse sido o de Tabela Proprietária seria necessário implementar novos tipos de algoritmos MapReduce para realizar o mapeamento de arquivos RDF em tabelas. Caso o esquema escolhido fosse o de Tabela de Triplas seria necessário implementar somente a fase Map do algoritmo. Nesta fase seria implementado um *parser* em cada linha do documento RDF, identificando o sujeito, predicado e objeto da linha. Após isso, o valor do sujeito e do predicado seriam atribuídos à chave de saída (MapOutputKey) da fase Map e do objeto ao valor de saída (MapOutputValue) da fase Map. Este processo seria repetido para cada linha do arquivo RDF.

Neste caso, tanto a fase Shuffle quanto a fase Reduce seriam ignoradas, logo todos os pares chave-valor gerados durante a fase Map seriam escritos de forma não ordenada no HDFS. O resultado armazenado em forma de arquivo TSV no HDFS corresponderia a uma tabela de triplas, com três colunas (sujeito, predicado, objeto), contendo todas as triplas do arquivo RDF, armazenadas uma por linha na tabela.

Caso o esquema escolhido fosse o de Particionameto Vertical seria necessário implementar um *job* completo MapReduce. Na fase Map seria implementado um *parser* em

cada linha do documento RDF. O predicado retornado por este *parser* iria compor a chave de saída da fase Map e o sujeito e o objeto iriam compor o valor de saída. Este processo seria repetido para cada linha do arquivo RDF, gerando vários pares chave-valor. Na fase Shuffle, estes pares seriam ordenados e agrupados de acordo com a chave (predicado).

Logo, cada tarefa Reduce iria receber todos os valores que possuem a mesma chave, ou seja o mesmo predicado, montando uma tabela de Particionamento Vertical referente a um único predicado. O resultado de cada tarefa Reduce seria um conjunto de pares chave-valor, onde a chave corresponde ao sujeito e o valor ao objeto da tripla. Para os dados do PingER, seriam geradas 68 tabelas de Particionamento Vertical, referentes aos 68 predicados diferentes existentes no arquivo RDF. Todas as tabelas iriam possuir duas colunas (sujeito e objeto), onde todos os sujeitos e objetos da tabela seriam ligados pelo predicado que dá o nome à tabela.

5.2.3 Criação e carga de tabelas no Impala

Após a geração dos arquivos TSV no HDFS, os dados destes arquivos foram carregados em tabelas externas no Impala. O script SQL abaixo mostra como foi feita a criação da tabela proprietária de classe MGC_Country_External como uma tabela externa no Impala. O script foi executado a partir do Impala-shell. A cláusula LOCATION foi utilizada para especificar o caminho para o diretório HDFS onde o Impala lê o arquivo com os dados a serem carregados na tabela. A cláusula ROW FORMAT DELIMITED foi utilizada para indicar qual foi o caractere delimitador utilizado no arquivo a ser carregado na tabela, no caso o caractere foi o *tab*. Alguns caracteres especiais, como o ":" e "-", não podem ser utilizados nos nomes de tabelas ou colunas no Impala, com isso estes caracteres tiveram que ser substituídos.

```
CREATE EXTERNAL TABLE MGC_Country_External(
  subject STRING,
  MGC_ContinentName STRING,
  MGC_GeonamesLink STRING,
  MGC_areaInSqKm double,
  MGC_capitalName STRING,
  MGC_countryCode CHAR(2),
  MGC_currency CHAR(4),
  MGC_isInContinent STRING,
  MGC_languages STRING,
  gn_ont_name STRING,
```

```
  gn_ont_population INT,
  rdf_type STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/user/cloudera/Pinger/tabelaProprietariaCountry'
```

Como foi citado na Seção 4.3.2, também é possível criar tabelas internas no Impala, que dão origem a novos arquivos que são armazenadas em diferentes diretórios no HDFS. O script SQL abaixo mostra como foi feita a criação da tabela interna MGC_Country_Snappy. A cláusula STORED AS é utilizada para identificar o formato do arquivo a ser criado, no caso foi utilizado o formato Parquet, que utiliza uma abordagem de armazenamento orientada a coluna. A cláusula SET COMPRESSION_CODEC indica o tipo de algoritmo de compressão de arquivos a ser utilizado, no caso foi utilizada a compressão Snappy. Após a criação da estrutura da tabela, os dados da tabela externa MGC_Country_External foram carregados na tabela interna.

```
CREATE TABLE MGC_Country_Snappy(
  subject STRING,
  MGC_ContinentName STRING,
  MGC_GeonamesLink STRING,
  MGC_areaInSqKm double,
  MGC_capitalName STRING,
  MGC_countryCode CHAR(2),
  MGC_currency CHAR(4),
  MGC_isInContinent STRING,
  MGC_languages STRING,
  gn_ont_name STRING,
  gn_ont_population INT,
  rdf_type STRING)
STORED AS PARQUET;
set COMPRESSION_CODEC=snappy;
INSERT INTO MGC_Country_Snappy SELECT * FROM MGC_Country_External;
```

Ao final da criação das tabelas foi necessário calcular as suas estatísticas e de suas colunas, conforme script abaixo, para que fosse possível otimizar automaticamente as consultas sobre elas.

```
COMPUTE STATS MGC_Country_External;
COMPUTE STATS MGC_Country_Snappy;
```

5.3 2º COMPONENTE (TRADUTOR DE LINGUAGEM SPARQL PARA SQL)

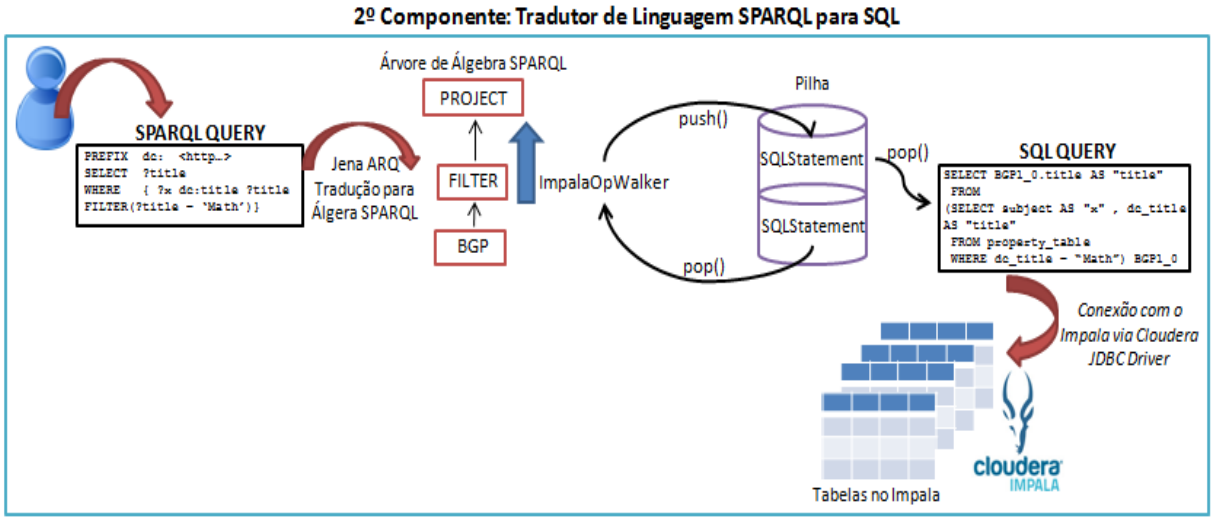

Figura 23: Esquema de funcionamento do Tradutor SPARQL para SQL

O segundo componente da aplicação *Impala MultiLOD* consiste em um tradutor de consultas escritas nas linguagem SPARQL para linguagem SQL. O tradutor implementado na linguagem JAVA trabalha com a versão 1.1 do SPARQL. O código-fonte do tradutor encontra-se disponível no Github[30].

O tradutor somente oferece suporte à cláusula DML SELECT. Isso se deve ao fato de que a aplicação *Impala MultiLOD* seria utilizada especificamente por usuários finais para realizar consultas analíticas sobre os dados e não executar alterações nestes dados.

Conforme o esquema ilustrado na Figura 23 o tradutor funciona da seguinte maneira: primeiramente, uma consulta SPARQL, gerada por usuários finais da aplicação a partir de uma interface gráfica, é lida e traduzida em uma árvore de álgebra SPARQL, que consiste em uma estrutura de operadores da álgebra SPARQL, já introduzidos na Seção 3.3.7. Esta tradução é realizada utilizando a biblioteca Jena ARQ (SEABORNE, 2010).

O pacote "org.apache.jena.sparql.algebra.op"[31] da biblioteca Jena ARQ possui classes que tratam cada um dos operadores SPARQL, possuindo métodos que permitem extrair informações sobre os argumentos associados a cada operador. Para entender como os operadores se relacionam dentro da árvore, eles são classificados pela biblioteca Jena ARQ em operadores raíz (Op0 - Não se relacionam a nenhum operador interno), unário (Op1 -

[30] https://github.com/raphaelapn/SPARQLtoSQLTranslator

[31] https://jena.apache.org/documentation/javadoc/arq/org/apache/jena/sparql/algebra/op/package-summary.html

Realizam uma operação sobre os resultados de um operador interno), binário (Op2 - Realizam uma operação sobre os resultados de dois operadores internos, o operador à esquerda é sempre o primeiro a ser avaliado) e n-ario (OpN - Realizam uma operação sobre os resultados de N operadores internos avaliados em ordem).

Após a geração da árvore de operadores da álgebra SPARQL, ela deve ser processada para ser gerada a tradução da consulta SPARQL para a linguagem SQL. O processo de tradução foi baseado na álgebra proposta em (PRUD'HOMMEAUX, BERTAILS, 2008) para o mapeamento de consultas na linguagem SPARQL para a linguagem SQL.

O processo de tradução ocorre da seguinte maneira: a árvore de operadores da álgebra SPARQL é percorrida de baixo para cima por um *parser*, ImpalaOpWalker, e seus operadores são traduzidos nesta ordem. Os resultados da tradução dos operadores percorridos são armazenados em uma estrutura de pilha. Ao serem percorridos, de acordo com o seu tipo, os operadores podem ser transformados em objetos que representam as instruções SQL: SELECT, UNION ou JOIN. Estas classes são filhas que herdam da classe pai abstrata SQLStatement.

Todos os objetos instanciados a partir das classes filhas de SQLStatement dão origem a subconsultas dentro da consulta SQL resultante. Entretanto, nem todos os operadores, como o ORDER, dão origem a novos objetos. Em vez disso, a instrução SQL mais atual, localizada no topo da pilha é alterada, adicionando, por exemplo, a cláusula ORDER BY a esta instrução. Na Seção 5.3.1 é explicado como cada operador da álgebra SPARQL é trabalhado no tradutor.

Conforme ilustrado na Figura 23, quando o objeto ImpalaOpWalker visita um nó da árvore de álgebra SPARQL, ele recupera a tradução mais recente referente à subárvore sob o nó atualmente visitado. Esta tradução é armazenada na pilha como um objeto da classe de SQLStatement. Com as informações sobre o nó visitado e o objeto SQLStatement recuperado da pilha, o nó visitado no momento pode ser traduzido, gerando um novo objeto SQLStatement, que também é empilhado. Este processo é repetido até que todos os nós da árvore de álgebra sejam traduzidos.

Na Seção 5.3.2 é exemplificado como é realizado o processo de tradução a partir de um exemplo de consulta SPARQL. Na Seção 5.3.3 é apresentada a interface gráfica construída para o tradutor e é explicado como foi feita a interligação do tradutor com o Impala.

5.3.1 Processo de tradução dos operadores da álgebra SPARQL para SQL

Nas próximas subseções é explicado como foi implementado o processo de tradução de cada operador da álgebra SPARQL.

5.3.1.1 Operador BGP (Padrão Básico de Grafo)

O operador BGP representa as triplas RDF que aparecem em uma consulta e constitui as folhas da árvore de álgebra, sendo o primeiro operador a ser traduzido, não se relacionando a nenhum operador interno (Op0). A tradução do operador BGP depende das triplas que estão representadas nele. Para o BGP que contém triplas com exatamente a mesma variável na posição do sujeito (caso muito comum em consultas do tipo *star-queries*), este é traduzido em um único objeto da classe SELECT. Sendo criada uma única instrução SQL SELECT que seleciona as colunas na tabela proprietária definidas pelos predicados nas triplas e o sujeito das triplas. A tabela proprietária que deve aparecer na cláusula FROM é identificada também a partir dos predicados nas triplas.

O exemplo abaixo mostra parte da tradução de uma consulta em formato de álgebra SPARQL onde o BGP possui o mesmo sujeito (`?country`) em todas as triplas.

```
(bgp
      (triple ?country MGC:capitalName ?capital)
      (triple ?country MGC:countryCode ?code)
      (triple ?country MGC:areaInSqKm ?area))
```

A sua tradução em uma instrução SQL SELECT é mostrada abaixo.

```
SELECT subject AS "country" ,
       MGC_capitalName AS "capital" ,
       MGC_countryCode AS "code" ,
       MGC_areaInSqKm AS "area"
 FROM MGC_Country_Snappy
```

Caso o sujeito da tripla seja uma URI, ele não é selecionado na cláusula SELECT, ao invés disso é adicionada uma condição na cláusula WHERE. O mesmo acontece caso o objeto seja uma URI ou um literal, o predicado referente a este objeto não é selecionado na cláusula SELECT, sendo adicionada uma condição na cláusula WHERE.

5.3.1.2 Operador Join

Caso as triplas representadas em um BGP possuam variáveis distintas na posição do sujeito, é gerado um objeto da classe SELECT para cada grupo de triplas que compartilham uma variável comum na posição do sujeito e posteriormente é criado um objeto da classe JOIN que junta os grupos de triplas.

O processo foi automatizado da seguinte maneira; as triplas do operador BGP são agrupadas por sujeito usando um HashMap cuja chave é o sujeito da tripla. Cada grupo de triplas cujas variáveis na posição do sujeito são iguais resulta na criação de um objeto SELECT que é traduzido em uma subconsulta dentro da consulta SQL resultante. Caso seja gerado mais de um grupo de triplas, um objeto JOIN é criado. O objeto JOIN desempilha dois objetos SELECT da pilha (Op2), que representam os grupos de triplas. A cláusula ON do objeto JOIN é determinada pelo cálculo das variáveis compartilhadas entre os grupos de triplas. Se dois grupos de triplas não compartilham variáveis em comum, é necessário calcular o produto cartesiano, utilizando o CROSS JOIN como tipo de junção.

O exemplo abaixo mostra parte da tradução de uma consulta em formato de álgebra SPARQL cujo BGP possui um grupo de triplas que compartilham a variável `?town` na posição do sujeito e outro grupo de triplas que compartilham a variável `?country` no sujeito.

```
(bgp
      (triple ?town PingER-ont:name ?name)
      (triple ?town MGC:isInCountry ?country)
      (triple ?country MGC:capitalName ?capital)
      (triple ?country MGC:countryCode ?code))
```

A sua tradução em uma instrução SQL SELECT é mostrada abaixo.

```
(SELECT BGP1_1.country AS "country" ,
        BGP1_1.capital AS "capital" ,
        BGP1_1.code AS "code" ,
        BGP1_0.town AS "town" ,
        BGP1_0.name AS "name"
 FROM
      (SELECT subject AS "country" ,
              MGC_capitalName AS "capital" ,
              MGC_countryCode AS "code"
      FROM MGC_Country_Snappy) BGP1_1
      JOIN
```

```
        (SELECT subject AS "town" ,
                PingER_ont_name AS "name" ,
                MGC_isInCountry AS "country"
         FROM MGC_Town_Snappy) BGP1_0
    ON(BGP1_1.country=BGP1_0.country)) BGP1
```

O grupo que compartilha a variável `?town` foi traduzido em um subconsulta que recebeu o *alias* `BGP1_0` para identificá-la. E o grupo que compartilha a variável `?country` foi traduzido em um subconsulta que recebeu o *alias* `BGP1_1` para identificá-la. Como os dois grupos de triplas possuem em comum a variável `?country`, esta variável determina a cláusula ON do objeto JOIN. Após os grupos serem juntados, o objeto JOIN é envolto em uma instrução SELECT, que recebeu o *alias* BGP1, cujos seletores são todas as colunas presentes nas cláusula SELECT dos grupos juntados, facilitando a referência aos resultados da junção.

Nem sempre o algoritmo desenvolvido calcula a ordem de junção mais apropriada para as tabelas. Entretanto, a partir da computação das estatísticas sobre as tabelas, o otimizador de consultas do Impala consegue reordenar automaticamente as tabelas, otimizando a ordem dos *joins*. Sendo o bastante, para evitar perdas no desempenho das consultas.

5.3.1.3 Operador LeftJoin

O operador LEFT JOIN é gerado na árvore de álgebra a partir da cláusula OPTIONAL da sintaxe SPARQL. O LEFT JOIN também dá origem a um objeto JOIN, porém o tipo do JOIN é atribuído como LEFT JOIN. O conjunto de triplas dentro da cláusula OPTIONAL também é um BGP, sendo traduzido em uma subconsulta SELECT. A cláusula ON do JOIN é identificada a partir do cálculo da variável em comum do objeto SELECT criado a partir da tradução da cláusula OPTIONAL com a do objeto SELECT desempilhado do topo da pilha. Caso não haja variável em comum, o tipo do JOIN é mudado para CROSS JOIN.

5.3.1.4 Operador Union

O operador UNION é traduzido em um objeto UNION. Ele recebe dois objetos SELECT (Op2) extraídos da pilha, que representam as subconsultas que serão unidas pela cláusula UNION.

Cada instrução SELECT dentro de UNION deve ter o mesmo número de colunas. Além disso, as colunas em cada instrução SELECT devem estar na mesma ordem. Logo, quando os dois objetos SELECT são desempilhados é realizada uma verificação para

assegurar que eles selecionem o mesmo esquema de colunas. Caso um dos objetos SELECT selecione uma coluna que não é selecionada pelo outro objeto, é determinado que o outro SELECT selecione um valor NULL cujo *alias* é nome da coluna que somente é selecionado pelo primeiro objeto.

O exemplo abaixo mostra parte da tradução de uma consulta em formato de álgebra SPARQL com dois BGPs que representam as subconsultas que serão unidas pela cláusula UNION.

```
(union
      (bgp (triple ?town MGC:isInCountry ?country))
      (bgp
        (triple ?town MGC:isInCountry ?country)
        (triple ?town PingER-ont:population ?population)))
```

Como o segundo BGP seleciona uma variável (`?population`) que não é selecionado pelo primeiro BGP, o primeiro BGP é traduzido é um objeto SELECT que irá selecionar além das colunas `subject` e `MGC_isInCountry`, o valor NULL cujo *alias* será "`population`", conforme mostrado abaixo. Assim, é possível simular que as instruções SELECT dentro do UNION selecionam o mesmo número de colunas na mesma ordem, não causando nenhum erro de compilação.

```
((SELECT subject AS "town",
         MGC_isInCountry AS "country",
         null AS "population"
 FROM MGC_Town_Snappy) UNION
(SELECT subject AS "town",
        MGC_isInCountry AS "country",
        PingER_ont_population AS "population"
 FROM MGC_Town_Snappy)) UNION1
```

5.3.1.5 Operador Group

O operador GROUP é gerado na árvore de álgebra a partir da cláusula GROUP BY da sintaxe SPARQL. Ele recebe um objeto SELECT (Op1) desempilhado do topo da pilha, resultante da tradução do operador BGP. O operador GROUP é traduzido em um novo objeto SELECT, onde o objeto desempilhado é inserido na sua cláusula FROM, representando uma subconsulta. Entre os seletores do novo objeto SELECT criado estão a função de agregação definida na consulta SPARQL e os parâmetros definidos na cláusula GROUP BY. A cláusula

GROUP BY é adicionada ao objeto SELECT da mesma forma em que ela foi definida na consulta SPARQL.

O exemplo abaixo mostra parte da tradução de uma consulta em formato de álgebra SPARQL com os operadores GROUP e EXTEND. As variáveis `?source` e `?dest` dentro de GROUP representam os parâmetros presentes na cláusula GROUP BY. A expressão `((?.0 (min distinct ?value))` representa a função de agregação `(MIN(distinct ?value) as ?min)`, que foi utilizada em conjunto com a cláusula DISTINCT, e a variável `?.0` é definida para representar esta função. O operador EXTEND é utilizado em suporte ao GROUP, para substituir a variável `?.0` pela variável definida como o *alias* para a função de agregação, no caso a variável utilizada foi `?min`.

```
(extend ((?min ?.0))
     (group (?source ?dest) ((?.0 (min distinct ?value)))
       (bgp
         (triple ?observation PingER-ont:hasValue ?value)
         (triple ?observation PingER-ont:hasSourceNode ?source)
         (triple ?observation PingER-ont:hasDestinationNode ?dest))))
```

A sua tradução em uma instrução SQL SELECT é mostrada abaixo.

```
(SELECT BGP1_0.source AS "source",
        BGP1_0.dest AS "dest" ,
        min(distinct BGP1_0.value) AS "min"
 FROM (SELECT subject AS "observation" ,
              PingER_ont_hasValue AS "value" ,
              PingER_ont_hasSourceNode AS "source" ,
              PingER_ont_hasDestinationNode AS "dest"
       FROM qb_Observation_Snappy) BGP1_0
 GROUP BY source, dest) AGREG
```

5.3.1.6 Operador Filter

O operador FILTER é gerado na árvore de álgebra tanto a partir da cláusula FILTER quanto a partir da cláusula HAVING da sintaxe SPARQL. O operador não é traduzido em um novo objeto SELECT, o que ele faz é desempilhar um objeto SELECT da pilha (Op1) e adicionar uma condição à cláusula WHERE deste objeto ou adicionar uma cláusula HAVING a este objeto.

O exemplo abaixo mostra parte da tradução de uma consulta em formato de álgebra SPARQL com operadores FILTER. O operador FILTER que vem logo antes do operador BGP representa um filtro sobre as triplas no BGP, adicionando uma condição à cláusula WHERE do objeto SELECT gerado a partir da tradução do BGP. Já o operador FILTER que vem logo antes do operador EXTEND representa um filtro sobre a agregação realizada, adicionando uma cláusula HAVING no objeto SELECT gerado a partir da tradução do operador GROUP.

```
(filter (> ?.0 10)
  (extend ((?min ?.0))
   (group (?source ?dest) ((?.0 (min ?value)))
    (filter (= ?dest PingER:Node-pinger.slac.stanford.edu)
      (bgp
       (triple ?observation PingER-ont:hasValue ?value)
       (triple ?observation PingER-ont:hasSourceNode ?source)
       (triple ?observation PingER-ont:hasDestinationNode ?dest))))))
```

5.3.1.7 Operador Project

O operador PROJECT é gerado na árvore de álgebra a partir da cláusula SELECT da sintaxe SPARQL. A partir do momento que se alcança o operador PROJECT na árvore de álgebra, a percorrendo no sentido de baixo para cima, toda a subárvore já terá sido traduzida em um objeto SELECT (que provavelmente possuíra subconsultas) empilhado no topo da pilha.

O operador PROJECT é traduzido em um novo objeto SELECT que possui na sua cláusula FROM o objeto SELECT desempilhado do topo da pilha (Op1). As variáveis de projeção são adicionados como seletores do objeto SELECT.

5.3.1.8 Operador Order

O operador ORDER adiciona a cláusula ORDER BY a um objeto SELECT no topo da pilha (Op1). Na implementação, foi definido que a cláusula ORDER BY somente é adicionada ao objeto SELECT gerado a partir da tradução do operador PROJECT, não sendo adicionada em nenhuma das subconsultas que possam existir dentro deste objeto. De acordo com o estabelecido na sintaxe SQL-92, sintaxe seguida pelo Impala, foi definido que a cláusula ORDER BY sempre aparece antes das cláusulas LIMIT e OFFSET dentro do objeto SELECT, pois os resultados devem ser ordenados antes de serem fatiados.

5.3.1.9 Operador Slice

O operador SLICE é gerado na árvore de álgebra tanto a partir da cláusula LIMIT quanto da cláusula OFFSET da sintaxe SPARQL. Assim como o operador ORDER, o operador SLICE adiciona as cláusulas LIMIT e/ou OFFSET a um objeto SELECT no topo da pilha (Op1). Na implementação, foi definido que estas cláusulas somente são adicionadas ao objeto SELECT gerado a partir da tradução do operador PROJECT, não sendo adicionadas em nenhuma das subconsultas que possam existir dentro deste objeto.

O operador é representado na álgebra SPARQL a partir da expressão `(slice [START] [LENGHT])`. Caso o parâmetro `start` tenha sido definido, significa que deve ser adicionada uma cláusula OFFSET ao objeto SELECT no topo da pilha, já que foi definido o ponto onde a solução da consulta deve começar a ser apresentada. Caso o parâmetro `lenght` tenha sido definido, significa que também deve ser adicionada uma cláusula LIMIT ao objeto SELECT, já que foi definido o número de soluções que devem ser retornadas pela consulta. De acordo com as regras da sintaxe SQL-92, foi definido que a cláusula LIMIT, quando utilizado em conjunto com a cláusula OFFSET, deve sempre aparecer antes da claúsula OFFSET.

5.3.1.10 Operador Distinct

Caso a consulta SPARQL a ser traduzida possua uma cláusula DISTINCT na seleção das colunas, é configurado um *flag* que define que uma cláusula DISTINCT deve ser adicionada ao objeto SELECT no topo da pilha (Op1), gerado a partir da tradução do operador PROJECT.

5.3.2 Exemplo de tradução de consulta SPARQL para SQL

Nesta seção é mostrado um exemplo de tradução de uma consulta em linguagem SPARQL para a linguagem SQL. A consulta SPARQL a ser traduzida é ilustrada abaixo.

```
select ?country (count(distinct ?source) as ?count)  (avg(?value) as
?avg) where{
?observation PingER-ont:hasValue ?value .
?observation PingER-ont:hasSourceNode ?source .
?observation qb:measureType PingER:MetricPacketLoss.
?source MGC:isInTown ?town .
?town MGC:isInCountry ?country.
FILTER(bound(?value))
}
```

```
group by ?country
```

A tradução da consulta em formato de álgebra SPARQL, realizada a partir da biblioteca Jena ARQ, é ilustrada abaixo:

```
(project (?country ?count ?avg)
  (extend ((?count ?.0) (?avg ?.1))
    (group (?country)((?.0(count distinct ?source))(?.1(avg ?value)))
      (filter (bound ?value)
        (bgp
         (triple ?observation PingER-ont:hasValue ?value)
         (triple ?observation PingER-ont:hasSourceNode ?source)
         (triple ?observation qb:measureType PingER:MetricPacketLoss)
         (triple ?source MGC:isInTown ?town)
         (triple ?town MGC:isInCountry ?country)))))))
```

Os nós da árvore de álgebra são percorridos e traduzidos no sentido de baixo para cima. A tradução da consulta SPARQL seguiu a seguinte a ordem:

1. Primeiramente foi realizada a tradução do operador BGP, cujas triplas constituem as folhas da árvore de álgebra. É possível notar que as triplas possuem diferentes variáveis na posição do sujeito. As três primeiras triplas possuem a variável `?observation` como sujeito, a quarta possui a variável `?source` e a última a variável `?town`. Com isso, as triplas são agrupados por sujeito em três grupos. Depois é feito o cálculo das variáveis que são compartilhadas entre os grupos. Descobrindo que o primeiro grupo compartilha a variável `?source` com o segundo grupo. E o segundo grupo compartilha a variável `?town` com o terceiro grupo. A partir desta informação é possível traduzir cada grupo em subconsultas SELECT (que receberam os *alias* `BGP1_0`, `BGP1_1` e `BGP1_2`) ligadas por JOINs cuja cláusula ON é determinada pelas variáveis compartilhadas entre os grupos. O tipo do JOIN é setado como INNER JOIN, que é a junção padrão adotada pelo Impala. As subconsultas ligadas por cláusulas JOIN são envolvidas por uma instrução SELECT envelope, que recebeu o alias `BGP1`. Como último passo, o objeto SELECT recém-criado é empilhado. Abaixo é mostrada a tradução do operador BGP para a linguagem SQL.

```
(SELECT BGP1_0.observation AS "observation", BGP1_0.value AS "value",
BGP1_0.source AS "source", BGP1_1.town AS "town", BGP1_2.country AS
"country"
 FROM
(SELECT subject AS "observation", PingER_ont_hasValue AS "value",
PingER_ont_hasSourceNode AS "source"
 FROM qb_Observation_Snappy
 WHERE qb_measureType = PingER:MetricPacketLoss) BGP1_0
JOIN (SELECT subject AS "source", MGC_isInTown AS "town"
 FROM MGC_NodeInformation_Snappy) BGP1_1
ON(BGP1_0.source=BGP1_1.source)
JOIN (SELECT subject AS "town", MGC_isInCountry AS "country"
 FROM MGC_Town_Snappy) BGP1_2 ON(BGP1_1.town=BGP1_2.town)) BGP1
```

2. Em seguida, o operador FILTER é processado. O objeto SELECT da etapa anterior é recuperada a partir do topo da pilha. A função `(bound(?value))` presente dentro do filtro é traduzida em `(BGP1_0.value is not NULL)` e adicionada à cláusula WHERE do objeto SELECT. Após isso, o objeto SELECT modificado é empilhado novamente.
3. Em seguida, o operador GROUP é processado. O operador GROUP é traduzido em uma nova subconsulta da classe SELECT (que recebeu o *alias* `AGREG`) onde o objeto SELECT desempilhado é inserido na sua cláusula FROM. Entre os seletores do novo objeto SELECT criado estão as duas funções de agregação definidas na consulta SPARQL e o parâmetro `?country` definido na cláusula GROUP BY. A cláusula GROUP BY é adicionada ao objeto SELECT da mesma forma em que ela foi definida na consulta SPARQL. Como último passo, o objeto SELECT recém-criado é empilhado. Abaixo é mostrada a tradução do operador GROUP para a linguagem SQL.

```
(SELECT BGP1.country AS "country" , count(distinct BGP1.source) AS
"?.0" , avg(BGP1.value) AS "?.1"
 FROM
(SELECT BGP1_0.observation AS "observation" , BGP1_0.value AS "value"
, BGP1_0.source AS "source" , BGP1_1.town AS "town" , BGP1_2.country
AS "country"
 FROM
(SELECT subject AS "observation" , PingER_ont_hasValue AS "value" ,
PingER_ont_hasSourceNode AS "source"
 FROM qb_Observation_Snappy
 WHERE qb_measureType = PingER:MetricPacketLoss) BGP1_0
```

```
JOIN (SELECT subject AS "source" , MGC_isInTown AS "town"
 FROM MGC_NodeInformation_Snappy) BGP1_1
ON(BGP1_0.source=BGP1_1.source)
JOIN (SELECT subject AS "town" , MGC_isInCountry AS "country"
 FROM MGC_Town_Snappy) BGP1_2
ON(BGP1_1.town=BGP1_2.town)
 WHERE BGP1_0.value is not NULL) BGP1
 GROUP BY country)AGREG
```

4. Em seguida, o operador EXTEND é processado. O objeto SELECT da etapa anterior é recuperado a partir do topo da pilha. Os alias `"?.0"` e `"?.1"` atribuídos às funções de agregação são substituídos pelos alias `"count"` e `"avg"`. O objeto SELECT modificado é empilhado novamente.
5. Finalmente, o operador PROJECT (nó raiz) é processado. O operador PROJECT é traduzido em um novo objeto SELECT que possui na cláusula FROM o objeto SELECT desempilhado do topo da pilha, que representa a tradução da subárvore abaixo do operador PROJECT. As variáveis de projeção definidas na consulta SPARQL são adicionados como seletores do objeto SELECT. A tradução final da consulta para a linguagem SQL é mostrada abaixo.

```
SELECT AGREG.country AS "country" , AGREG.count AS "count" ,
AGREG.avg AS "avg"
 FROM
(SELECT BGP1.country AS "country" , count(distinct BGP1.source) AS
"count" , avg(BGP1.value) AS "avg"
 FROM
(SELECT BGP1_0.observation AS "observation" , BGP1_0.value AS "value"
, BGP1_0.source AS "source" , BGP1_1.town AS "town" , BGP1_2.country
AS "country"
 FROM
(SELECT subject AS "observation" , PingER_ont_hasValue AS "value" ,
PingER_ont_hasSourceNode AS "source"
 FROM qb_Observation_Snappy
 WHERE qb_measureType = PingER:MetricPacketLoss) BGP1_0
JOIN (SELECT subject AS "source" , MGC_isInTown AS "town"
 FROM MGC_NodeInformation_Snappy) BGP1_1
ON(BGP1_0.source=BGP1_1.source)
JOIN (SELECT subject AS "town" , MGC_isInCountry AS "country"
 FROM MGC_Town_Snappy) BGP1_2 ON(BGP1_1.town=BGP1_2.town)
```

```
WHERE BGP1_0.value is not NULL) BGP1
GROUP BY country) AGREG
```

5.3.3 Interface gráfica criada para o tradutor SPARQL para SQL

Suponha-se que o usuário final da aplicação *Impala MultiLOD* tem por objetivo não só executar consultas analíticas escritas na linguagem SPARQL sobre os dados armazenados no Impala e medir o desempenho destas consultas, como também verificar e validar o resultado do processo de tradução das consultas SPARQL em SQL. Devido a este perfil de usuário final, foi criada uma interface gráfica para que estes usuários possam digitar consultas escritas na linguagem SPARQL e visualizar o resultado da tradução em formato de álgebra SPARQL e na forma de consulta SQL.

Após a tradução automática para SQL, o usuário pode optar por executar a consulta traduzida diretamente no Impala. A conexão com o Impala é feita a partir de um *driver* JDBC fornecido pelo Cloudera. O *driver* transforma uma consulta escrita na linguagem SQL para sua forma equivalente em Impala SQL, linguagem de consulta utilizada pelo Impala.

A Figura 24 ilustra a primeira tela do tradutor. O primeiro campo texto é editável e é onde o usuário deve inserir a consulta que deseja na linguagem SPARQL. Já foram definidos na aplicação prefixos padrões do domínio PingER que não necessitam ser digitados novamente pelo usuário na consulta. Após a consulta ser traduzida, as suas traduções em formato de álgebra SPARQL e em linguagem SQL podem ser visualizadas no segundo e terceiro campos texto, respectivamente.

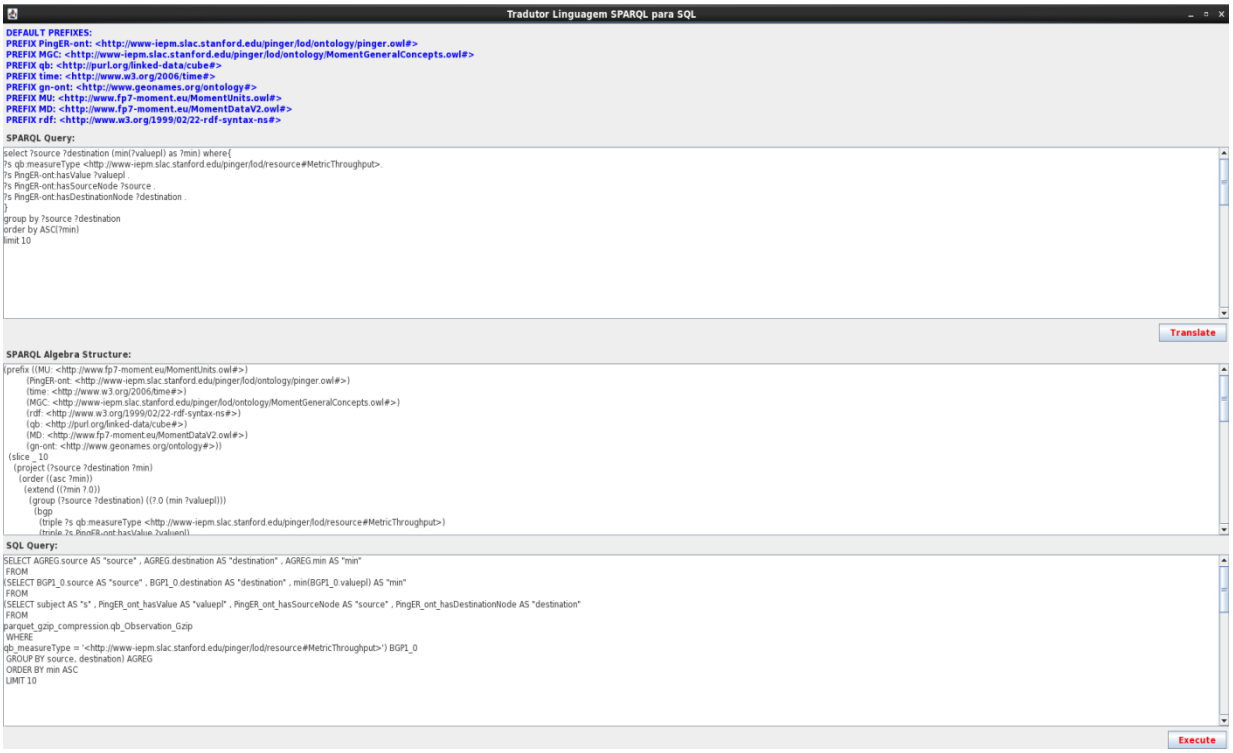

Figura 24: Tela onde é feita a tradução da consulta SPARQL

Caso o usuário concorde com o resultado da tradução, ele pode optar por executar a consulta no Impala. A Figura 25 ilustra a segunda tela do tradutor, que exibe o resultado da execução de uma consulta. Além do resultado, também é exibido o tempo gasto na execução.

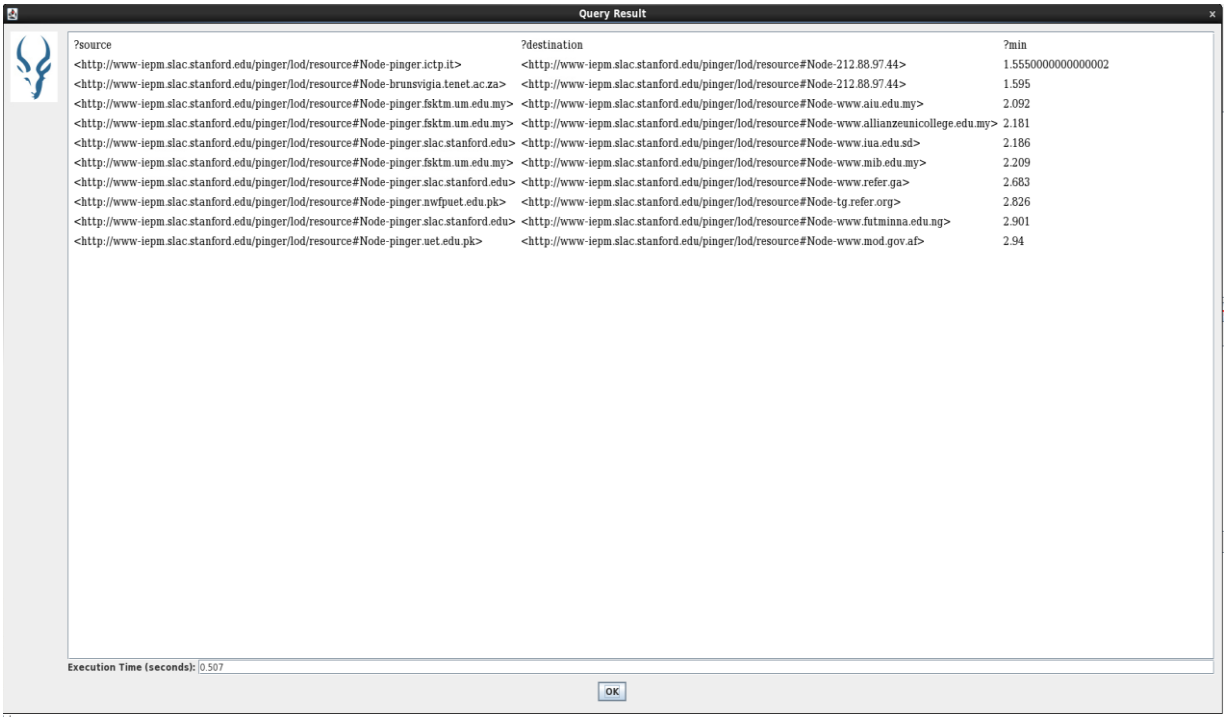

Figura 25: Tela com o resultado da execução da consulta SQL traduzida no Impala

6 EXPERIMENTO

Os experimentos realizados na aplicação foram feitos a partir de um estudo de caso que utilizou dados de medições reais originados do projeto *Ping End-to-end Reporting* (PingER) (COTTRELL, 2001). O estudo de caso teve por objetivo avaliar se a aplicação proposta nesta dissertação é adequada para a gestão dos conjuntos volumosos de dados no formato RDF do projeto e é capaz de melhorar o desempenho de consultas analíticas SPARQL tipicamente executadas sobre seus dados.

Na Seção 6.1 é feita uma explicação do estudo caso realizado, esclarecendo quais foram os seus objetivos e personagens envolvidos. Na Seção 6.2 é detalhada a etapa de pré-processamento realizada sobre os dados do PingER, na qual os dados foram extraídos de sua fonte de origem, mapeados para o formato RDF e carregados no HDFS.

Na Seção 6.3 são detalhados os testes executados para medir o desempenho geral da aplicação *Impala MultiLOD* utilizando conjuntos de dados selecionados do projeto PingER. Na Seção 6.4 os resultados de desempenho obtidos são comparados com os do sistema Virtuoso Triple Store, que foi carregado e testado de forma distribuída no mesmo ambiente computacional em que foi testada a aplicação *Impala MultiLOD*, utilizando os mesmos conjuntos de dados.

6.1 ESTUDO DE CASO COM DADOS DO PROJETO PINGER

Conforme explicado no Capítulo 2, o projeto PingER (COTTRELL, 2001) armazena conjuntos de dados reais oriundos de processos de medição de qualidade de *links* de Internet, desde 1998, utilizando uma granularidade horária e diária e 16 métricas de rede diferentes. A medição da qualidade de *links* de Internet é feita a partir de *pings* realizados a cada 30 minutos, que medem diferentes métricas de desempenho de rede e armazenam as medidas em arquivos CSV a cada hora.

Devido aos dados do PingER serem representados em níveis de granularidade baixos e serem gerados e armazenados em arquivos a cada hora, o volume e a taxa de crescimento destes dados é bem alta. Além disso, quando os dados do projeto são representados no formato RDF, o volume dos dados aumenta ainda mais. Por isso, foi desenvolvido um estudo de caso com o objetivo de avaliar uma solução para a gestão eficiente do volume de dados crescente do projeto e avaliação de consultas analíticas SPARQL sobre estes dados apresentando bons resultados de desempenho.

Os usuários finais da solução seriam não só os usuários que executariam consultas analíticas sobre os dados com o objetivo de obter informações consolidadas e estariam particulamente interessados nos tempos de execução das consultas, mas também os administradores dos dados do projeto, que também estariam preocupados com outras métricas de desempenho, tais como tempo de carga de conjuntos de dados e quantidade em disco consumida por arquivos de dados.

6.2 ETAPA DE PRÉ-PROCESSAMENTO

Nesta seção é apresentada a etapa de pré-processamento dos dados do projeto PingER, na qual os dados foram extraídos de sua fonte de origem, mapeados para o formato de triplas RDF e carregados no HDFS. Para isso, primeiramente, foi proposta uma nova versão da ontologia de domínio do projeto PingER baseada na estrutura do vocabulário RDF Data Cube, para servir como um vocabulário de referência e estrutura para representar e anotar os dados como triplas RDF. A descrição da nova versão da ontologia é feita na Seção 6.2.1.

Após isso, na Seção 6.2.2 são apresentados os conjuntos de dados do projeto PingER selecionados para testar o desempenho da aplicação. Na Seção 6.2.3 é explicado como foram realizados os processos de triplificação dos conjuntos de dados selecionados e na Seção 6.2.4 é explicado o processo de carga dos dados triplificados no HDFS.

6.2.1 Nova versão da Ontologia PingER baseada no Vocabulário RDF Data Cube

Como foi citado na Seção 2.3.1, os dados do domínio PingER são de natureza multidimensional. Entretanto, a ontologia original do PingER, apresentada na Seção 3.5.3, tem como foco representar os conceitos específicos do domínio, representando apenas parcialmente os aspectos de multidimensionalidade dos dados. Este fato motivou a tentativa de adaptar a estrutura da ontologia original, tornando-a compatível com a estrutura do vocabulário RDF Data Cube, explicitando, com isso, os aspectos de multidimensionalidade e as perspectivas de análise do domínio, e facilitando a criação de consultas analíticas sobre os dados.

Além disso, outras vantagens que podem ser atingidas a partir da adequação da estrutura da ontologia à estrutura do RDF Data Cube são facilitar a publicação dos dados do PingER na Web, utilizando um formato já adotado de forma mais ampla na representação de dados multidimensionais em RDF, possibilitando a ligação e combinação dos dados com informações relacionadas, além de favorecer a extensão do modelo, na medida em que a

estrutura do RDF Data Cube é extensível, e permite a introdução de novos conceitos à modelagem. A Figura 26 ilustra a modelagem da ontologia PingER, escrita na linguagem OWL, utilizando conceitos do vocabulário RDF Data Cube. Todos os prefixos (*namespaces*) utilizados na ontologia encontram-se definidos no Apêndice A desta dissertação.

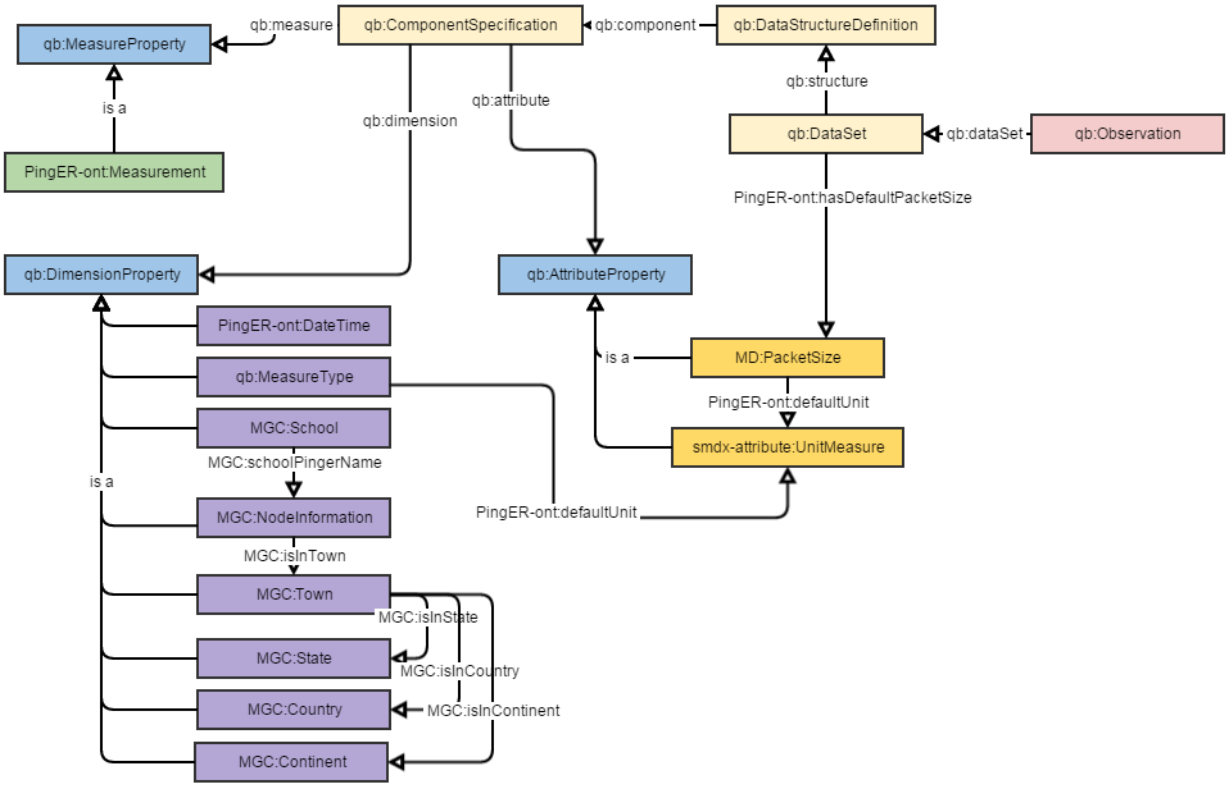

Figura 26: Modelagem da ontologia do domínio PingER utilizando conceitos do vocabulário RDF Data Cube

A adaptação da ontologia original começou a partir da representação das dimensões presentes no conjunto de dados do PingER, seguindo a estrutura do vocabulário RDF Data Cube. Para isso foram criadas classes para cada dimensão, definindo-as como subclasses da classe `qb:DimensionProperty`. As classes criadas foram `PingER-ont:DateTime`, `MGC:NodeInformation`, `MGC:School`, `MGC:Town`, `MGC:State`, `MGC:Country`, `MGC:Continent` e `qb:MeasureType`.

`PingER-ont:DateTime`: Seguindo o que foi feito em (SOUZA et al., 2014), foi utilizada a ontologia Time (HOBBS, PAN, 2006), a qual define a classe `DateTimeDescription`, para descrever intervalos de tempo. Foram utilizadas as propriedades `day`, `month`, `year`, `dayOfYear`, `dayOfWeek` e `unitType` para cobrir todos os intervalos de tempo. Além disso, foi utilizada a propriedade `displayValue` já presente na ontologia PingER. A declaração da dimensão é ilustrada abaixo:

```
#DateTime Dimension Definition
PingER-ont:DateTime rdf:type qb:DimensionProperty;
      rdfs:subPropertyOf time:DateTimeDescription;
      time:unitType TemporalUnit
      time:year         "Year"^^xsd:gYear
      time:month        "Month"^^xsd:gMonth
      time:day          "Day"^^xsd:gDay
      time:dayOfYear    "DayOfYear"^^xsd:int
      time:dayOfWeek    "DayOfWeek"^^xsd:string
      PingER-ont:displayValue "DisplayValue"^^xsd:string
```

MGC:NodeInformation: Para representar cada nó da rede foi criada esta dimensão, que pode acomodar dois papéis em relação a uma observação: o papel de nó monitor (propriedade `PingER-ont:hasSourceNode`) ou o papel de nó monitorado (propriedade `PingER-ont:hasDestinationNode`).

Para realizar a identificação da localização do nó de rede é informado em qual cidade o nó se localiza, através da propriedade `MGC:isInTown`. A cidade apontada é uma instância da dimensão `MGC:Town`, que foi criada para armazenar informações hierárquicas, apontando em qual estado, país e continente a cidade se localiza. Da mesma forma foram criadas as dimensões `MGC:State`, `MGC:Country` e `MGC:Continent`, que também possuem informações hierárquicas. Todas as classes que realizam a localização geográfica do nó foram relacionadas à classe `gn-ont:Feature` da ontologia Geonames (VATANT; WICK, 2012) e também foram definidas como subclasses de `qb:DimensionProperty`, por se tratarem de dimensões.

Informações sobre o nó, como seu IP e seu nome completo, também estão presentes na estrutura da dimensão. As declarações das dimensões `MGC:NodeInformation` e `MGC:Town` são ilustradas abaixo:

```
#NodeInformation Dimension Definition
MGC:NodeInformation rdf:type qb:DimensionProperty;
      PingER-ont:hasNodeName        "NameNode"^^xsd:string;
      PingER-ont:hasNodeFullName    "FullName"^^xsd:string;
      PingER-ont:hasNodeNickName    "NickName"^^xsd:string;
      PingER-ont:hasNodeSiteName    "SiteName"^^xsd:string;
      PingER-ont:hasNodeIP          "NodeIP"^^xsd:string;
      PingER-ont:hasNodeURL         "NodeURL"^^xsd:string;
      PingER-ont:hasLocationDescription   "LocDesc"^^xsd:string;
      PingER-ont:hasProjectType     "Project"^^xsd:string;
      PingER-ont:hasTraceServer     "TraceServer"^^xsd:string;
```

```
      PingER-ont:hasGMTOffset       "NodeGMT"^^xsd:string;
      PingER-ont:hasPingServer      "PingServer"^^xsd:string;
      PingER-ont:hasDataServer      "DataServer"^^xsd:string;
      PingER-ont:hasAppUser         "AppUser"^^xsd:string;
      PingER-ont:PingERLat          "PingERLat"^^xsd:string;
      PingER-ont:PingERLong         "PingERLong"^^xsd:string;
      MGC:isInTown                  MGC:Town;

#Town Dimension Definition
MGC:Town rdf:type qb:DimensionProperty;
      rdfs:subPropertyOf      gn-ont:Feature;
      POS:lat                 "lat"^^xsd:string;
      POS:long                "long"^^xsd:string;
      PingER-ont:population   "pop"^^xsd:int;
      PingER-ont:name         "name"^^xsd:string;
      PingER-ont:postalcode   "postalCode"^^xsd:string;
      MGC:GeoState            "GeoState"^^xsd:string;
      MGC:GeoCountry          "GeoCountry"^^xsd:string;
      MGC:GeoGMTOffset        "GeoGMTOffset"^^xsd:string;
      MGC:GeoContinent        "GeoContinent"^^xsd:string;
      MGC:GeonamesLink        gn-ont:Town;
      gn-ont:parentADM1       gn-ont:State;
      gn-ont:parentCountry    gn-ont:Country;
      MGC:isInCountry         MGC:Country;
      MGC:isInContinent       MGC:Continent;
      MGC:isInState           MGC:State;
```

`MGC:School`: Dimensão criada para identificar a qual Instituição de Ensino um nó de rede está associado. Ela se relaciona com a dimensão `MGC:NodeInformation` através da propriedade `MGC:schoolPingerName`. A declaração da dimensão é ilustrada abaixo:

```
#School Dimension Definition
PingER:School rdf:type qb:DimensionProperty;
      MGC:DBPediaLink    "Link"^^xsd:string
      MGC:schoolType     "Type"^^xsd:string
      MGC:schoolNumberOfStudents "Students"^^xsd:int
      MGC:schoolNumberOfUgradStudents "UgradStudents"^^xsd:int
      MGC:schoolNumberOfGradStudents      "GradStudents"^^xsd:int
      MGC:schoolEndowment "Endowment"^^xsd:bigint
      MGC:schoolFacultySize "Size"^^xsd:int
      MGC:schoolName    "Name"^^xsd:string
```

```
MGC:schoolPingerName    MGC:NodeInformation
```

qb:MeasureType: As medidas realizadas no conjunto de dados do PingER podem ser de diferentes naturezas, variando de acordo com a métrica de rede escolhida para a medição. Por isso, é necessário que a ontologia dê suporte a múltiplos tipos de medidas. Para estes casos, no vocabulário RDF Data Cube, é criada a dimensão `qb:MeasureType`. A instância desta dimensão indica a métrica em particular que está sendo coberta pela observação. Além da propriedade `PingER-ont:displayValue`, que indica o nome da métrica, a propriedade `PingER-ont:defaultUnit` também é utilizada para representar a unidade padrão de medida, que é uma instância do atributo `smdx-attribute:UnitMeasure`. A declaração da dimensão é ilustrada abaixo:

```
#MeasureType Dimension Definition
qb:MeasureType rdf:type qb:DimensionProperty;
      PingER-ont:displayValue "NameMeasure"^^xsd:string;
      PingER-ont:defaultUnit  smdx-attribute:UnitMeasure;
```

Os atributos de uma observação podem ser qualquer metadado que descreva como a medição foi feita. Dentre os possíveis atributos do domínio PingER estão unidade de medida (`smdx-attribute:UnitMeasure`) e tamanho de pacote de dados (`MD:PacketSize`).

sdmx-attribute:UnitMeasure: Para definir o atributo de unidade de medida foi utilizado o atributo pré-definido `sdmx-attribute:UnitMeasure`. Como os valores observados no domínio PingER podem possuir tipos diferentes de unidade de medida, variando de acordo com a métrica escolhida, foi inserida a propriedade `MU:hasSymbol`, para identificar o símbolo utilizado para representar a unidade de medida. A declaração do atributo é ilustrada abaixo:

```
#UnitMeasure Attribute Definition
sdmx-attribute:UnitMeasure rdf:type qb:AttributeProperty;
      MU:hasSymbol      "unit"^^xsd:string
```

MD:PacketSize: O atributo tamanho de pacote define o tamanho do pacote de dados utilizado em uma medição. Para relacionar um valor ao atributo foi criada a propriedade `MGC:PacketSizeValue` e para representar a unidade padrão de medida do tamanho de pacote, que é uma instância do atributo `smdx-attribute:UnitMeasure`, foi utilizada a propriedade `PingER-ont:defaultUnit`. A declaração do atributo é ilustrada abaixo:

```
#PacketSize Attribute Definition
MD:PacketSize rdf:type qb:AttributeProperty;
      MGC:PacketSizeValue     "PacketSizeValue"^^xsd:int
      PingER-ont:defaultUnit  smdx-attribute:UnitMeasure;
```

Para combinar os componentes (dimensões, medida e atributos) em uma especificação para a estrutura do *data set*, é necessário declarar um recurso `qb:DataStructureDefinition` que por sua vez fará referência a um conjunto de recursos `qb:ComponentSpecification`. O `qb:DataStructureDefinition` pode ser reutilizável em todos os outros *data sets* que possuam a mesma estrutura. A declaração da estrutura do *data set* PingER é ilustrada abaixo:

```
#Define DataStructureDefinition
PingER:DataStructureDefinition1 rdf:type qb:DataStructureDefinition;
      # The dimensions
    qb:component [ qb:dimension PingER-ont:DateTime;];
    qb:component [ qb:dimension qb:MeasureType;];
    qb:component [ qb:dimension MGC:NodeInformation;];
    qb:component [ qb:dimension MGC:School;];
    # The measure(s)
    qb:component [ qb:measure PingER-ont:Measurement];
    # The attributes
    qb:component [ qb:attribute sdmx-attribute:UnitMeasure;
                   qb:attribute MD:PacketSize;] .
```

Um recurso representando o *data set* é criado e identificado com o tipo `qb:DataSet`, ligado a sua estrutura correspondente através da propriedade `qb:structure`. O domínio PingER possui como tamanho de pacote de dados padrão 100 bytes, este atributo é uniforme em todo o conjunto de dados e não muda entre as observações. Para atender a esse tipo de situação o vocabulário RDF Data Cube permite aos componentes serem anexados a um nível elevado na estrutura aninhada. Assim, a declaração do `MD:PacketSize` é anexada ao nível do *Data Set* e não ao de observação, sendo ligada à definição do *data set* através da propriedade `PingER-ont:hasDefaultPacketSize`.

A declaração do *data set* PingER é ilustrada abaixo:

```
#Define Data Set
PingER:Dataset1 rdf:type qb:DataSet;
      qb:structure PingER:DataStructureDefinition1;
      PingER-ont:hasDefaultPacketSize    PingER:PacketSize100 .
```

Cada observação realizada é representada como uma instância do tipo `qb:Observation`. A observação é ligada ao *data set* correspondente usando a propriedade `qb:DataSet`. No caso do domínio PingER, como a dimensão `MGC:NodeInformation` executa dois papéis diferentes (nó monitor e nó monitorado), duas instâncias desta dimensão serão ligadas a uma observação através das propriedades `PingER-ont:hasSourceNode` e `PingER-ont:hasDestinationNode`. A Figura 27 ilustra como a classe `qb:Observation` se relaciona com as outras classes da ontologia. Um exemplo de instância de uma observação é o seguinte:

```
#Observation instance
PingER:Observation-
pinger.slac.stanford.edu_www.bankofbotswana.bw_AverageRTT_13Jan01
rdf:type qb:Observation;
      qb:dataSet PingER:Dataset1;
      PingER-ont:hasDateTime  PingER:Time13Jan01;
      qb:measureType    PingER:MetricAverageRTT;
      PingER-ont:hasSourceNode      MGC:Node-pinger.slac.stanford.edu;
      PingER-ont:hasDestinationNode MGC:Node-www.bankofbotswana.bw;
      PingER-ont:hasValue     "363.585"
```

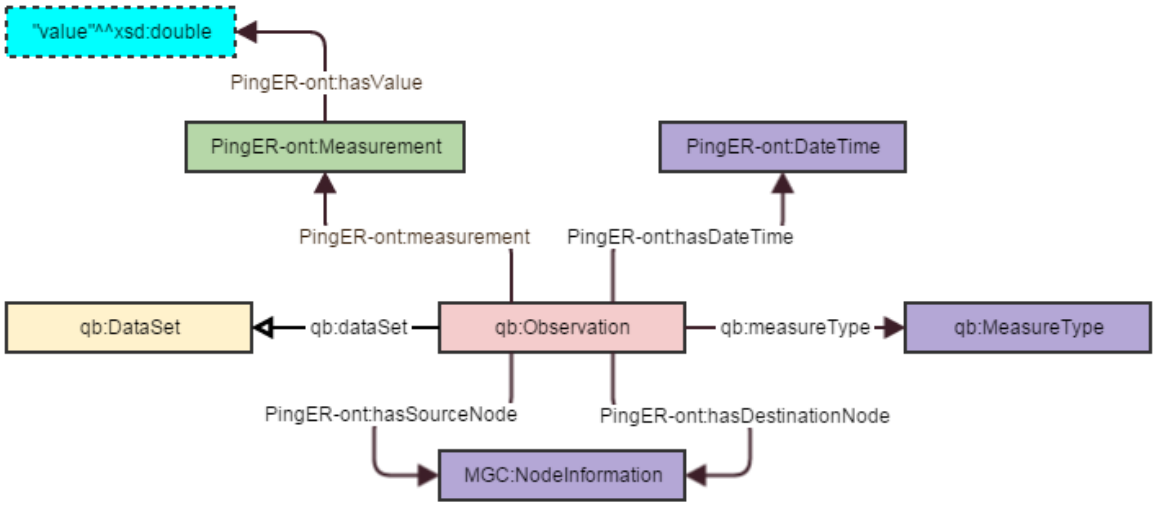

Figura 27: Relacionamentos da classe qb:Observation na ontologia do domínio PingER

6.2.2 Conjuntos de dados do PingER selecionados para testar a aplicação

Para testar o desempenho da aplicação *Impala MultiLOD*, verificando a forma como a aplicação reage quando há um aumento no número de recursos compartilhados foi necessário recolher amostras volumosas de dados do PingER, que após triplificadas dessem origem a 3

conjuntos de dados em formato RDF com, aproximadamente, 100, 200 e 300 milhões de triplas.

Para gerar cada conjunto foram baixados arquivos, a partir do FTP público do SLAC, referentes a 5 métricas de rede fixas, aumentando os intervalos de tempo de medição para conseguir chegar no volume de dados pretendido para cada conjunto. As 5 métricas de rede consideradas foram: Packet Loss, Unreachability, Unpredictability, Maximum Round Trip Time (RTT) e Mean Opinion Score (MOS). Uma explicação mais detalhada de cada uma destas métricas é feita no Apêndice B desta dissertação.

Todos os arquivos baixados possuem os seguintes níveis de detalhe:

- *Nível de detalhe geográfico*: Medições de nó para nó.
- *Nível de detalhe de tempo*: Granularidade de dia.
- *Tamanho do pacote de dados*: 100 bytes.

A opção de gerar conjuntos de dados com tamanhos crescentes, começando com 100 milhões de triplas e crescendo seguindo uma escala de 100 milhões de triplas, foi baseada em *benchmarks*, tais como o SP2Bench (SCHMIDT et al., 2009), que seguem este tipo de escala de crescimento para avaliar o desempenho de sistemas de gestão de dados RDF. Entretanto, alguns *benchmarks*, tais como o BSBM (BIZER, SCHULTZ, 2009), adotam escalas de crescimento maiores e chegam a gerar conjuntos de dados com bilhões de triplas. Em trabalhos futuros pretendemos testar a aplicação com conjuntos de dados que contenham bilhões de triplas, porém, para isso, será necessário trabalhar em um nível granularidade de tempo menor (granularidade horária) e selecionar dados referentes a uma quantidade maior de métricas de rede.

A Tabela 3 mostra os intervalos de tempo de medição considerados em cada conjunto de dados trabalhado. Além disso, é mostrada a quantidade de arquivos baixados para gerar cada conjunto de dados e a quantidade de medidas não nulas e nulas, quantificadas a partir do sistema estatístico R[32], presentes nestes arquivos baixados.

O total de triplas gerado a partir de cada conjunto de arquivos baixados foi estimado da seguinte forma: cada medida não nula (uma observação) na ontologia do PingER gera 7 triplas e ao todo temos 49428 triplas representando os dados gerais presentes nas dimensões; assim, o total de triplas gerado para cada conjunto de arquivos baixados pode ser calculado a partir da fórmula *((total de medidas não nulas em todos os arquivos)*7 + 49428))*. A

[32] http://www.r-project.org/

estimativa da quantidade real de triplas presente em cada conjunto de dados também é mostrada na Tabela 3.

É possível notar a partir dos resultados expostos na Tabela 3, que foi necessário, aproximadamente, um intervalo de tempo de um ano para gerar um conjunto de dados com o dobro do tamanho do anterior. Este intervalo de tempo relativamente alto se deve às escolhas de agregação tomadas para as amostras geradas. Se fosse escolhido trabalhar no menor nível de granularidade temporária possível para os dados do PingER, granularidade horária, este intervalo de tempo seria menor.

Tabela 3: Conjuntos de dados do PingER selecionados para testar a aplicação

Tamanho Aproximado do Conjunto de Dados	**Intervalo de Tempo de Medição Considerado**	**Total de Arquivos Baixados**	**Total de Medidas Nulas**	**Total de Medidas Não Nulas**	**Estimativa do Nº de Triplas**
100M	Dez/2011 até Fev/2013 (15 meses)	75 arquivos	4865039	15279601	107006635
200M	Dez/2011 até Abr/2014 (29 meses)	145 arquivos	10025955	29381320	205718668
300M	Dez/2011 até Abr/2015 (41 meses)	205 arquivos	15268277	43420095	303990093

6.2.3 Triplificação

O processo de triplificação foi realizado utilizando a ferramenta ETL4LOD[33] (MENDONÇA, CRUZ, CAMPOS, 2014), produzida pelo GRECO-UFRJ, na qual a ferramenta Pentaho Data Integration[34] foi estendida com *plugins* para tratamentos de dados RDF e acesso via SPARQL. A ferramenta Pentaho Data Integration, também chamada de Kettle, é a ferramenta de *workflow* ETC integrante do conjunto de ferramentas da plataforma de BI Pentaho (CASTERS, BOUMAN, VAN DONGEN, 2010).

A triplificação dos conjuntos de dados foi feita a partir de dois *workflows* ETC independentes (capazes de serem executados em paralelo) que realizaram a publicação como triplas RDF dos dados gerais (armazenados nas tabelas de dimensão) e dos dados de medidas de rede (armazenados na tabela fato). Os *workflows* foram executados de forma não

[33] http://greco.ppgi.ufrj.br/lodbr/index.php/principal/etl4lod-2/

[34] http://community.pentaho.com/projects/data-integration/

distribuída, em um único nó do *cluster* montado com configuração de 7.6GB RAM, Intel Core I7-4770 (4 *cores*) e 1TB Hard Disk.

6.2.3.1 Triplificação dos dados gerais

Além dos dados específicos de medidas, a aplicação utilizou dados sobre conceitos gerais, que foram armazenados nas tabelas de dimensão, tais como dados geográficos (continentes, países, estados, cidades), temporais, de universidades e dos nós de redes (monitores e monitorados). Estes dados foram extraídos a partir de diferentes fontes de dados, que são explicadas mais à frente.

A triplificação dos dados gerais foi realizada a partir de um *workflow* ETC no Kettle, especificado a partir de uma transformação (*transformation*), que consiste em um conjunto de passos conectados, onde cada passo, denominado *step*, é responsável por uma atividade de extração, transformação ou carga de dados. A conexão entre dois passos de uma transformação é feita através de um *transformation hop*, conexão que permite que os dados fluam em único sentido e de maneira assíncrona.

Em resumo, o *workflow* ETC criado realizou a extração dos dados a partir de seu formato original, transformação dos dados para o formato de triplas RDF e carga dos dados em um arquivo no formato N-Triples.

Para cada tipo de dado trabalhado, foram criados fluxos independentes no *workflow* ETC, que foram rodados paralelamente. A Figura 28 ilustra uma parte do *workflow*, com os dois fluxos responsáveis pela triplificação dos dados presentes nas dimensões Continente (*Continent*) e País (*Country*). A seguir, são explicados os seus principais passos de acordo com as marcações feitas na Figura 29.

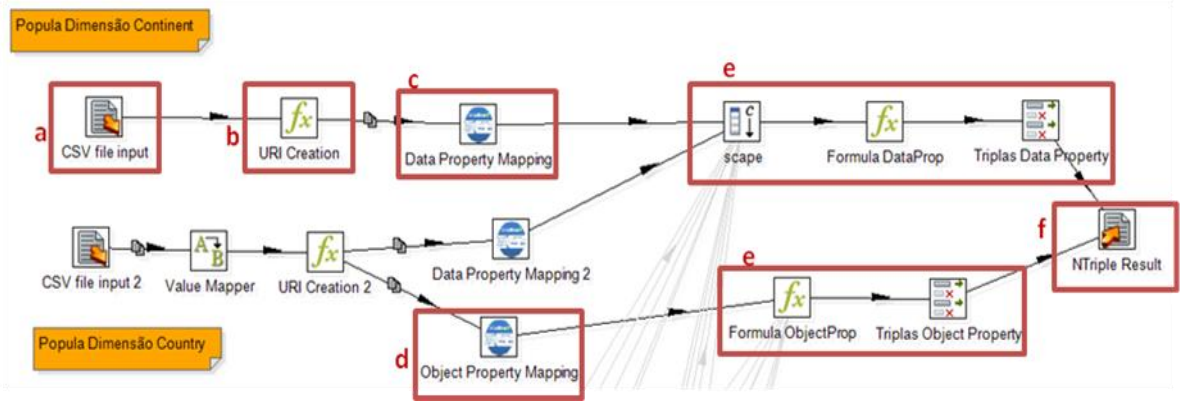

Figura 28: Parte do *workflow* ETC para triplificação de dados gerais

a) Entrada de dados: Neste passo foram extraídos dados a partir de arquivos CSV. Para cada tipo de dado geral trabalhado foi necessário realizar os seguintes procedimentos, antes do processo de triplificação, para extrair os dados a partir de sua fonte de origem e mapeá-los para arquivos no formato CSV, que serviram de entrada para este passo:
    - Dimensão Continente: Por constituírem um conjunto estático e pequeno de dados, foram instanciados diretamente no arquivo CSV.
    - Dimensão País: A partir da URL <http://peric.github.io/GetCountries/> foi possível recuperar os dados de todos os países presentes no Geonames[35] no formato CSV. O Geonames é uma base de dados geográfica disponível e acessível através de *web services*.
    - Dimensão Nó de Rede: Foi criado um código na linguagem JAVA que realizou um HTTP GET na página do PingER[36], que contém um arquivo no formato PERL com todos os dados relativos a cada nó (monitor e monitorado), como o IP, Latitude, Longitude, Nome do nó, etc. O conteúdo deste arquivo foi lido e transformado para o formato JSON, o qual, posteriormente, foi transformado para o formato CSV, gerando o arquivo NodeDetails.csv.
    - Dimensões Cidade e Estado: Também foi criado um código na linguagem JAVA onde, para cada nó presente no arquivo NodeDetails.csv foi executado um HTTP GET no *web service* do Geonames para recuperar os dados da cidade populada mais próxima da latitude e longitude do nó (*NearbyPlaceName*). Dentre os dados retornados, constava o estado em que a cidade se encontra. Os dados foram retornados no formato JSON, convertidos para o formato CSV e inseridos no arquivo Town.csv. Este arquivo serviu de entrada no *workflow* ETC tanto para o fluxo responsável pela triplificação dos dados presentes na dimensão Cidade quanto para o fluxo responsável pela triplificação dos dados presentes na dimensão Estado.
    - Dimensão Universidade: Também foi criado um código na linguagem JAVA onde, para cada nó presente no arquivo NodeDetails.csv, foi realizada um consulta no terminal SPARQL da DBPedia[37] para encontrar universidades cujo nome fosse similar ao nome completo do nó. Se fossem encontradas informações sobre a

[35] http://www.geonames.org/

[36] http://www-iepm.slac.stanford.edu/pinger/pingerworld/all-nodes.cf

[37] http://dbpedia.org/sparql

universidade, tais como número de estudantes, tipo de universidade - pública ou particular e fundos monetários, estas informações seriam inseridas em um arquivo CSV.

- Dimensão Métrica de Rede: As informações sobre as métricas foram instanciadas diretamente no arquivo CSV.
- Dimensão Tempo: A instanciação do intervalo de tempo considerado no experimento foi feita através do próprio Kettle, conforme retratado na Figura 29. Diferentemente dos fluxos de carga das outras dimensões, não foi necessário utilizar uma fonte de dados, pois é possível gerar uma sequência de datas, dado um intervalo de tempo, e formatar estas datas conforme o desejado, dentro da própria ferramenta.

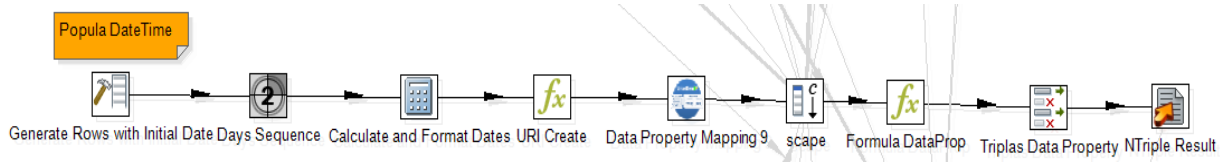

Figura 29: Fluxo para carga da dimensão Tempo

b) Formatação da URI: Formata as URIs de objetos que serão utilizadas em passos posteriores.
c) Propriedade de tipo de dados (*Data Property Mapping*): Transforma linhas do fluxo de entrada em triplas RDF no fluxo de saída utilizando um mapeamento *Data Property*, que relaciona um objeto da ontologia a um dado tipado.
d) Propriedade de objeto (*Object Property Mapping*): Transforma linhas do fluxo de entrada em triplas RDF no fluxo de saída utilizando um mapeamento *Object Property*, que relacionam um objeto a outro objeto da ontologia.
e) Formatação das triplas RDF
f) Carga das triplas geradas em um arquivo N-Triples.

Ao todo o processo de triplificação deu origem a 49428 triplas representando os dados presentes nas dimensões e levou um tempo de 1 segundo para ser executado, produzindo um arquivo N-Triples com tamanho de 7.6 MB.

6.2.3.2 Triplificação dos dados de medidas de rede

A Figura 30 ilustra uma visão panorâmica do *workflow* ETC criado para realizar a triplificação dos conjuntos de dados de medidas de rede selecionados na Seção 6.1.2. Este

workflow considera como granularidade de tempo dos conjuntos de dados a granularidade diária, permitindo que a triplificação dos dados coletados em diferentes dias de um mês possa ser feita de forma independente e paralela, o que acelera a triplificação.

Para isso foram criados 62 fluxos independentes no *workflow*, onde 31 fluxos são responsáveis por transformar dados coletados em cada dia de um mês em triplas RDF utilizando um mapeamento *Data Property*. E os outros 31 fluxos transformam os dados coletados em cada dia de um mês em triplas RDF utilizando um mapeamento *Object Property*. Todos os fluxos são rodados simultaneamente e paralelamente. A seguir são explicados os principais passos do *workflow* ETC de acordo com as marcações na Figura 30.

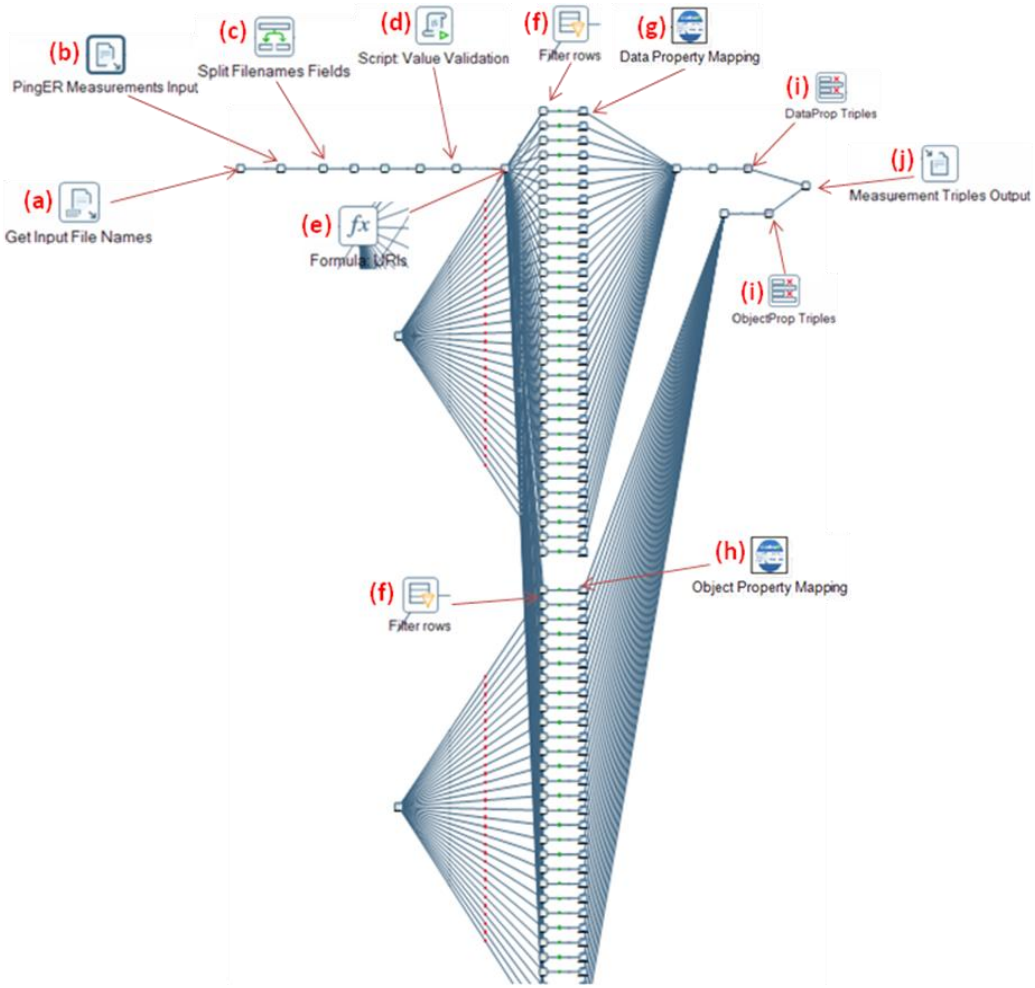

Figura 30: Visão panorâmica do *workflow* ETC que realiza a triplificação dos dados de medidas de rede

a) Definição do caminho para o diretório com os arquivos de dados: Os arquivos CSV com os dados de medidas de rede foram recuperados a partir do FTP público disponibilizado pelo SLAC. A partir deste FTP foi possível recuperar arquivos CSV de forma compactada referentes a todos os tipos de métricas de rede e granularidades existentes, fato que

acelerou bastante o processo de extração dos dados a partir da sua fonte de origem. Todos os arquivos selecionados a partir do FTP foram descompactados e inseridos em um diretório a partir do qual foram acessados.

b) Leitura dos nomes dos arquivos de entrada.
c) *Split* dos nomes dos arquivos de entrada: Permite identificar qual métrica de rede, tamanho de pacote de dados, mês e ano estão sendo tratados no arquivo de entrada.
d) *Script* para tratar a variação da quantidade de dias em um mês.
e) Formatação da URI: Formata as URIs de objetos que serão utilizadas em passos posteriores.
f) Filtro de medida não nula: Confere se há uma medida não nula para o dia considerado. Se sim, segue para o próximo passo.
g) Geração de triplas RDF no fluxo de saída utilizando o mapeamento *Data Property.*
h) Geração de triplas RDF no fluxo de saída utilizando o mapeamento *Object Property*.
i) Formatação das triplas RDF.
j) Carga das triplas geradas em um arquivo N-Triples.

A triplificação dos conjuntos de arquivos de medidas de rede que deram origem a aproximadamente 100, 200 e 300 milhões de triplas, descritos na Tabela 3, foi feita a partir de *workflows* ETC executados separadamente. Na Tabela 4 é mostrado o tempo total para triplificação dos conjuntos de dados, o número total de triplas gerado e o tamanho dos arquivos N-Triples gerados.

Considerando o grande volume de dados manipulados no *workflow* ETC, a quantidade de triplas geradas como produto de saída e o fato que a execução foi feita de forma não distribuída, em uma única máquina, os tempos gastos nos processos de triplificação se mostraram plausíveis, de acordo com as condições impostas.

Tabela 4: Estatísticas dos processos de triplificação dos conjuntos de dados de medidas

Tamanho aproximado do conjunto de dados	**Número de triplas gerado**	**Tempo gasto na triplificação**	**Tamanho do arquivo N-Triples gerado**
100M	106957207	3h 45min 13s	26.9 GB
200M	205669240	6h 56min 12s	51.7 GB
300M	303940665	10h 19min 53s	76.4 GB

6.2.4 Carga dos arquivos N-Triples no HDFS

A carga dos arquivos N-Triples no HDFS foi realizada a partir de um *job* no Kettle, ilustrado na Figura 31. Um *job* também consiste de um conjunto de passos conectados. No entanto, os passos de um *job*, denominados *job entries*, são responsáveis por executar um *transformation*, outro *job* ou atividades auxiliares como manipular e transferir arquivos. O *Big Data Plugin*[38] é um *plugin* disponível para o Kettle que estende as funcionalidades da ferramenta, adicionando novos *steps* e *job entries* que permitem que o Kettle trabalhe com tecnologias de *Big Data*, como Hadoop, Amazon Elastic Map Reduce, Apache Sqoop, Oozie, entre outras. O Kettle pode ser configurado para se comunicar com as distribuições mais populares do Hadoop, inclusive com diferentes versões do CDH.

No *job* ilustrado na Figura 31 foi utilizado o *job entry Hadoop Copy Files*, disponibilizado a partir da adição do *Big Data Plugin* no Kettle. Sua função é copiar arquivos armazenados localmente para dentro de um *cluster* Hadoop. Para isso é necessário que sejam identificados o endereço do Namenode do HDFS, cuja porta padrão é a 8020, um usuário HDFS com permissão de escrita, o diretório na máquina local em que o arquivo a ser copiado está localizado e o diretório no HDFS onde será inserido o arquivo. A Figura 32 mostra a interface onde foi feita a identificação do endereço do Namenode (*node4.crbd.ufrj.br:8020*) e a localização do diretório no HDFS onde foi copiado o arquivo de origem (diretório */user/cloudera/Pinger*).

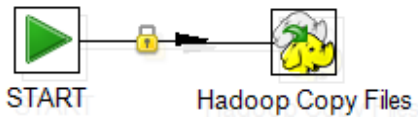

Figura 31: *Job* para carga de arquivos de dados no HDFS

[38] http://wiki.pentaho.com/display/BAD/Pentaho+Big+Data+Community+Home

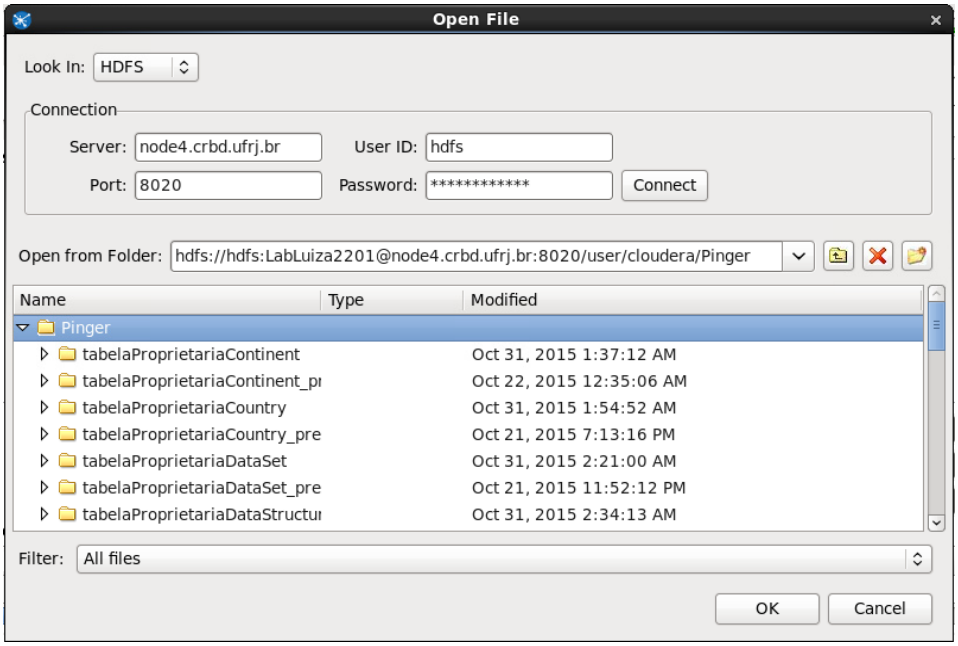

Figura 32: Interface de configuração de conexão do Kettle com o HDFS

A Tabela 5 mostra o tempo levado para carregar o arquivo N-Triples de dados gerais e os arquivos N-Triples de medidas de rede, gerados na etapa de triplificação, no HDFS. Também é mostrado o tamanho ocupado por estes arquivos no *cluster* sem e com o fator de replicação (de 3 vezes o tamanho do bloco de dados) aplicado pelo HDFS.

Tabela 5: Estatísticas dos processos de carga dos arquivos N-Triples no HDFS

Arquivo	**Tempo para carga no HDFS**	**Tamanho no HDFS**	**Tamanho no HDFS com replicação**
Arquivo de Dados Gerais	2s	7.6 MB	22.8 MB
Arquivo de medidas de rede com 100M de triplas	43min 1s	26.9 GB	80.7 GB
Arquivo de medidas de rede com 200M de triplas	1h 22min 40s	51.7 GB	155.1 GB
Arquivo de medidas de rede com 300M de triplas	2h 02min 18s	76.4 GB	229.2 GB

6.3 TESTES DE DESEMPENHO EXECUTADOS SOBRE A APLICAÇÃO *IMPALA MULTILOD*

Os testes realizados para medir o desempenho geral da aplicação *Impala MultiLOD* utilizando conjuntos de dados selecionados do projeto PingER foram executados em duas etapas. A primeira etapa teve por objetivo medir o desempenho das tabelas proprietárias de classe criadas no Impala em 3 formatos diferentes:

- Tabelas Externa com formato de arquivo TSV e sem algoritmo de compressão
- Tabela Interna com formato de arquivo Parquet (orientado por coluna) e algoritmo de compressão Snappy
- Tabela Interna com formato de arquivo Parquet (orientado por coluna) e algoritmo de compressão Gzip

Optou-se por criar tabelas internas utilizando o formato de arquivo Parquet, pois ele é especificamente recomendado pelo Cloudera (CLOUDERA, 2016) para melhorar o desempenho de consultas analíticas, que utilizam funções de agregação, visto que utiliza uma abordagem de armazenamento orientada a coluna, que permite acesso direto a dados de uma coluna, sem precisar ler a linha inteira da tabela. Os algoritmos de compressões Snappy e Gzip foram adotados pois são os únicos algoritmos que podem ser utilizados com o formato de arquivo Parquet, onde o algoritmo de compressão padrão é o Snappy (CLOUDERA, 2016).

O desempenho das diferentes tabelas criadas no Impala foi medido a partir das seguintes métricas de desempenho:

1. Tamanho em disco ocupado pelas tabelas
2. Tempos de criação, inserção de dados e cálculo de estatísticas
3. Tempo de execução de consultas analíticas SQL tipicamente executadas sobre os dados do PingER

Foram definidas tais métricas de desempenho não só porque são métricas de interesse dos usuários finais da aplicação, definidos na Seção 6.1, mas também porque são métricas tipicamente adotadas em *benchmarks*, como o TPC-H (COUNCIL, 2005), para medir o desempenho de sistemas de gestão de dados.

A segunda etapa teve por objetivo verificar o desempenho do tradutor de consultas SPARQL para SQL criado e a corretude dos seus resultados. Para isso as métricas adotadas foram:

1. Tempo de execução do mesmo conjunto de consultas analíticas utilizadas para testar as tabelas no Impala, porém escritas na linguagem SPARQL
2. Corretude das respostas produzidas a partir da execução das consultas traduzidas

Após obter os tempos de execução das consultas analíticas SPARQL que foram traduzidas para SQL e executadas a partir do tradutor no Impala, estes tempos de execução foram comparados com os tempos de execução das mesmas consultas escritas diretamente em SQL e executadas diretamente no Impala, sem a intervenção do tradutor. Esta comparação

teve por objetivo descobrir se a tradução automática das consultas para SQL afetou negativamente o desempenho das consultas.

6.3.1 Teste do desempenho do 1º componente da aplicação (Carga e Preparação do Banco de Dados)

Após a carga dos arquivos N-Triples no HDFS foram executados algoritmos MapReduce que realizaram o mapeamento dos arquivos em tabelas proprietárias de classe no formato de arquivos TSV, armazenados no HDFS, conforme explicado na Seção 5.2.2.

A Tabela 6 mostra a quantidade de linhas e colunas presentes em cada tabela proprietária de classe gerada no formato de arquivo TSV. Todos os dados de medidas de rede triplificados foram armazenados na tabela proprietária Observação (*Observation*), que corresponde à tabela fato. Como foram processados 3 arquivos de medidas diferentes, com aproximadamente 100, 200 e 300 milhões, foram geradas 3 tabelas proprietárias de Observação.

Na Tabela 6, também é apresentada a quantidade de triplas RDF que foram armazenadas em cada tabela.

Tabela 6: Quantidade de linhas e colunas em cada Tabela Proprietária de Classe criada

Tipo da Tabela	Tabela Proprietária de Classe	Nº de triplas armazenadas na tabela	Nº de colunas	Nº de linhas
Dimensão	Continent	35	6	7
	Country	2750	12	250
	State	2544	7	424
	Town	13200	17	825
	NodeInformation	18734	18	1102
	MeasureType	48	4	16
	DateTime	9976	9	1247
	School	2141	11	264
Fato	Observation (100M)	106957207	8	15279601
	Observation (200M)	205669240	8	29381320
	Observation (300M)	303940665	8	43420095

A partir dos resultados expostos é possível notar que a abordagem de tabela proprietária de classe, que adota tabelas largas com muitas colunas, diminui bastante o número final de linhas em cada tabela. No caso da tabela proprietária Observação, cada 7

triplas presentes no arquivo N-Triples são acomodadas em uma única linha da tabela. Com isso, um arquivo N-Triples que possuía 106957207 triplas descrevendo medidas de rede foi transformado em uma tabela proprietária de 15279601 linhas. Esse fato ratifica a escolha do esquema de Tabela Proprietária como um esquema capaz de armazenar os dados RDF de forma econômica.

Os dados dos arquivos TSV gerados foram carregados em tabelas externas e internas no Impala, com o objetivo de testar seus desempenhos gerais e verificar qual tipo de tabela é a mais adequada para armazenar os dados da aplicação *Impala MultiLOD*.

Os arquivos TSV armazenados no HDFS foram carregados, primeiramente, em tabelas externas no Impala. O procedimento para criação e carga de tabelas externas foi mostrado na Seção 5.2.3. Estes arquivos após serem carregados continuaram a residir no mesmo local no HDFS. Como foram carregados arquivos TSV já existentes no HDFS em tabelas externas, não foi possível optar por criar partições nestas tabelas, pois gerariam novos arquivos de dados.

Já durante a criação das tabelas internas no Impala, que durante a carga dos seus dados dão origem a novos arquivos de dados na área de trabalho do Impala, foi possível optar por criar partições nas tabelas. A técnica de particionamento dividiu fisicamente os dados durante a carga, com base em valores de uma ou mais colunas, gerando um diretório separado para armazenar os arquivos de cada partição. Com isso, as partições criadas nas tabelas foram identificadas a partir das colunas que são mais testadas no conjunto de consultas SQL executado sobre as tabelas. Esse fato permitiu examinar apenas os arquivos de dados dos diretórios correspondentes aos valores das colunas testadas, reduzindo a leitura de dados.

Conforme explicado na Seção 5.2.3, durante a criação da estrutura das tabelas internas é necessário identificar o formato do arquivo de dados a ser criado e o tipo de algoritmo de compressão de arquivo a ser utilizado. Foram testados 2 tipos de tabelas internas, uma que adotava o formato de arquivo Parquet e o algoritmo de compressão Snappy e outra que também adotava o formato Parquet, mas adotava o algoritmo de compressão Gzip.

Para testar o desempenho das 3 opções de tabela criadas no Impala, foram consideradas as seguintes métricas de desempenho:

1. Tamanho em disco ocupado pelas tabelas
2. Tempos de criação, inserção de dados e cálculo de estatísticas
3. Tempo de execução de consultas analíticas SQL tipicamente executadas sobre os dados do PingER

6.3.1.1 Tamanho em disco ocupado pelas tabelas

A Tabela 7 mostra, para cada tipo de tabela testada, o espaço ocupado em disco. Além disso, são mostradas as colunas que determinaram as partições nas tabelas (caso tenham sido criadas) e o número de partições de arquivos que foram criadas, determinadas pela quantidade de valores diferentes existentes na coluna.

Tabela 7: Tamanho das tabelas externas e internas testadas e partições criadas

Tabela Proprietária de Classe	**Tipo da Tabela**	**Formato do arquivo de dados**	**Coluna que determinou a partição**	**Nº de partições**	**Tamanho da Tabela**
Continent	Externa	TSV	X	1	1.73KB
	Interna	Parquet(Snappy)	X	1	1.55KB
		Parquet(Gzip)	X	1	1.65KB
Country	Externa	TSV	X	1	80.46KB
	Interna	Parquet(Snappy)	MGC:isInContinent	7	33.34KB
		Parquet(Gzip)	MGC:isInContinent	7	29.91KB
State	Externa	TSV	X	1	156.92KB
	Interna	Parquet(Snappy)	X	1	25.65KB
		Parquet(Gzip)	X	1	20.06KB
Town	Externa	TSV	X	1	435.50KB
	Interna	Parquet(Snappy)	MGC:isInCountry	173	573.43KB
		Parquet(Gzip)	MGC:isInCountry	173	660.34KB
NodeInformation	Externa	TSV	X	1	475.86KB
	Interna	Parquet(Snappy)	X	1	181.13KB
		Parquet(Gzip)	X	1	130.93KB
MeasureType	Externa	TSV	X	1	3.26KB
	Interna	Parquet(Snappy)	X	1	1.54KB
		Parquet(Gzip)	X	1	1.51KB
DateTime	Externa	TSV	X	1	262.56KB
	Interna	Parquet(Snappy)	time:year	5	33.73KB
		Parquet(Gzip)	time:year	5	23.88KB
School	Externa	TSV	X	1	85.98KB
	Interna	Parquet(Snappy)	MGC:schoolType	26	74.07KB
		Parquet(Gzip)	MGC:schoolType	26	77.13KB

Observation (100M)	Externa	TSV	X	1	8.12GB
	Interna	Parquet(Snappy)	qb:measureType	5	218.93MB
		Parquet(Gzip)	qb:measureType	5	100.55MB
Observation (200M)	Externa	TSV	X	1	15.58GB
	Interna	Parquet(Snappy)	qb:measureType	5	429.08MB
		Parquet(Gzip)	qb:measureType	5	197.94MB
Observation (300M)	Externa	TSV	X	1	23.01GB
	Interna	Parquet(Snappy)	qb:measureType	5	650.70MB
		Parquet(Gzip)	qb:measureType	5	306.35MB

A partir dos valores expostos, chega-se à conclusão que o tipo de tabela interna com formato de arquivo Parquet com compactação Gzip gera arquivos de dados consideravelmente menores em relação aos outros tipos de tabelas testadas, na maioria dos casos. De acordo com a documentação oficial do Cloudera (CLOUDERA, 2016), o algoritmo Gzip é reconhecido por atingir o mais alto nível de compressão em relação a outros tipos de algoritmos, proporcionando uma maior economia de espaço em disco.

Outra conclusão interessante tirada é que nas tabelas em que se criou um número grande de partições, caso das tabelas Town e School, foram gerados arquivos de dados pequenos - com poucas linhas - em cada partição e o custo de compactar estes arquivos pequenos nem sempre se mostrou vantajoso. No caso da tabela Town, na qual foram geradas 173 partições nas tabelas internas, o tamanho total dos arquivos de dados compactados gerados foi maior do que o tamanho do arquivo de dados não compactado e não particionado gerado a partir da criação da tabela externa.

A coluna que determinou as partições da tabela Observation (tabela fato) foi a coluna qb:measureType, que define o tipo de métrica que está sendo coberta pela observação. Como foram trabalhadas cinco métricas de rede fixas nos três conjuntos de dados utilizados, o número de partições da tabela Observation não aumentou com o crescimento do volume de dados na tabela. Ou seja, é possível notar que a tabela Observation com conjuntos de dados com 100, 200 e 300 milhões de triplas possui o mesmo número de partições (5 partições). Caso o número de partições aumentasse com o aumento do tamanho do conjunto de dados, o desempenho de consultas sobre a tabela fato poderia melhorar. Em trabalhos futuros pretende-se realizar uma exploração mais aprofundada do projeto de particionamento de tabelas e a criação de partições crescentes.

6.3.1.2 Tempos de criação, inserção de dados e cálculo de estatísticas

A Tabela 8 mostra o tempo (em segundos) levado para realizar a criação das tabelas (cláusula CREATE TABLE), inserção de dados (cláusula INSERT INTO) e cálculo de estatísticas (cláusula COMPUTE STATS), conforme explicado na Seção 5.2.3. Para as tabelas externas não foi necessário realizar o procedimento de inserção de dados na tabela, já que elas apontam para arquivos de dados pré-existentes localizados no HDFS e estas ligações são definidas durante a criação das tabelas. Por isso foi marcado um X nas células da coluna referente ao tempo de inserção de dados em tabelas externas (formato de arquivo TSV).

Tabela 8: Tempo de criação, inserção de dados e cálculo de estatísticas nas tabelas testadas

Tabela Proprietária de Classe	**Tempo de Criação**			**Tempo para Inserção de Dados**			**Tempo para Cálculo de Estatísticas**		
	TSV	Snappy	Gzip	TSV	Snappy	Gzip	TSV	Snappy	Gzip
Continent	0.11s	0.16s	0.15s	X	0.31s	0.31s	1.12s	0.91s	1.01s
Country	0.10s	0.16s	0.16s	X	5.17s	5.42s	1.01s	0.91s	1.01s
State	0.08s	0.16s	0.29s	X	5.01s	5.39s	4.45s	1.41s	1.31s
Town	0.10s	0.15s	0.18s	X	10.67s	9.26s	5.23s	5.62s	5.33s
NodeInformation	0.08s	0.17s	0.19s	X	1.00s	3.12s	1.21s	1.01s	1.11s
MeasureType	0.12s	0.16s	0.16s	X	5.62s	3.97s	1.01s	0.91s	0.91s
DateTime	0.11s	0.14s	0.15s	X	7.19s	6.36s	6.10s	1.41s	1.11s
School	0.13s	0.13s	0.16s	X	4.77s	5.52s	1.11s	1.41s	1.61s
Observation(100M)	0.12s	0.15s	0.19s	X	66.91s	62.37s	36.13s	9.64s	9.34s
Observation(200M)	0.14s	0.15s	0.14s	X	116.36s	101.28s	84.21s	18.07s	18.05s
Observation(300M)	0.15s	0.14s	0.12s	X	175.89s	143.57s	90.19s	26.08s	25.57s

A partir dos resultados expostos, é possível notar que o tempo de criação das tabelas externas, no geral, foi menor que o tempo de criação das tabelas internas, para ambos os tipos de compressão. As tabelas internas com compressão Snappy apresentaram intervalos de tempo para inserção de dados menores que os das tabelas internas com compressão Gzip quando poucos dados foram inseridos, como foi o caso das tabelas School e Country. Entretanto, para o caso de tabelas grandes, como as tabelas Observation, em que foi inserida uma quantidade maior de dados, a tabela interna com compressão Gzip obteve um desempenho melhor. Fato parecido ocorreu no cálculo de estatísticas. Quando foram

calculadas estatíticas em cima de um volume grande de dados, caso das tabelas Observation, a compressão Gzip apresentou um desempenho um pouco melhor que a Snappy.

6.3.1.3 Tempo de execução do conjunto de consultas analíticas SQL

Após os dados terem sido inseridos nas tabelas e as estatísticas calculadas, um conjunto de dez consultas SQL do tipo OLAP que são tipicamente executadas sobre os dados do PingER, com o objetivo de obter informações consolidadas e basear estudos que se relacionam à qualidade de *links* de rede ao redor do mundo, foram elaboradas e executadas para testar o desempenho geral dos 3 tipos de tabelas criadas no Impala.

Todas as consultas definidas foram baseadas nas consultas analíticas do *benchmark* TPC-H (COUNCIL, 2005). Logo, as consultas possuem alto grau de complexidade, envolvendo diferentes operadores de agregação, agrupamento, filtros e modificadores de solução. O Quadro 4 mostra o objetivo, as características e os operadores envolvidos em cada consulta. No Apêndice C são listadas todas consultas SQL elaboradas.

Quadro 4: Conjunto de consultas SQL criado para testar o desempenho dos 3 tipos de tabelas no Impala

	Objetivo	**Características**	**Operadores**
Q1	Verifica os 10 pares de nós que registraram a maior taxa de perda de pacotes de rede	Utiliza 1 fragmento da tabela Observation, Agrupa em 2 grupos, Não faz join	MAX, ORDER BY DESC, LIMIT
Q2	Verifica os 10 pares de nós que tiveram a menor taxa de qualidade de voz, apresentando os resultados retornados a partir do 5º par de nós	Utiliza 1 fragmento da tabela Observation, Agrupa em 2 grupos, Não faz join, Utiliza Offset	MIN, ORDER BY ASC, OFFSET, LIMIT
Q3	Recupera as métricas de rede (Packet Loss e Unreachability) e seus valores médios agregados por país para todos os nós monitorados cujo nó monitor é o SLAC	Utiliza Union, Cada subconsulta utiliza 1 fragmento da tabela Observation, Agrupa em 4 grupos, Cada suconsulta faz 3 Inner Joins	AVG, UNION, FILTER, JOIN
Q4	Verifica o valor da média de pacotes perdidos entre nós monitores nos EUA e monitorados no Paquistão, entre os meses de janeiro e abril de todos os anos avaliados, cujo valor seja maior que 0.75	Utiliza 1 fragmento da tabela Observation e 2 fragmentos da tabela Town, Agrupa em 2 grupos, Utiliza Having, Faz 5 Inner Joins	AVG, FILTER, BETWEEN, HAVING, JOIN, ORDER BY ASC

Q5	Verifica o valor da média de pacotes perdidos, de inacessibilidade e tempo máximo de viagem ida-e-volta entre nós monitores nos EUA e monitorados no Paquistão, entre os meses de janeiro e abril de todos os anos avaliados, cujo valor seja maior que 0.75	Idem à Q4, mas utiliza 3 fragmentos da tabela Observation e agrupa em 3 grupos	AVG, FILTER, BETWEEN, HAVING, JOIN, ORDER BY ASC
Q6	Compara os valores médios das métricas de rede MOS, Maximum RTT e Unpredictability para nós monitorados localizados em diferentes continentes do planeta, agregando os resultados por continente	Utiliza 3 fragmentos da tabela Observation, Agrupa em 2 grupo, Faz 3 Inner Joins	AVG, JOIN, ORDER BY ASC
Q7	Investiga a relação entre o tipo de universidade (publica ou privada) com a média da taxa de inacessibilidade das universidades	Utiliza 1 fragmento da tabela Observation e 2 fragmentos da tabela School, Agrupa em 1 grupo, Faz 2 Inner Joins	AVG, FILTER, JOIN, ORDER BY DESC
Q8	Verificar a relação entre a qualidade da rede e a qualidade da universidade. Para tal, recupera-se o número de fundos arrecadados pela universidade e também, caso esteja disponível, o seu número de alunos universitários, bem como a média das suas métricas de rede.	Percorre todos os fragmentos da tabela Observation, Agrupa em 4 grupos, Faz 2 Inner Joins e 1 Left Join	AVG, FILTER, JOIN, LEFT JOIN, ORDER BY ASC
Q9	Verifica os valores máximos e mínimos registrados para cada métrica de rede em todos os meses dos anos avaliados	Percorre todos os fragmentos das tabelas Observation e DateTime, Realiza 2 tipos de agregação (MAX e MIN), Agrupa em 3 grupos, Faz 1 Inner Join	MAX, MIN, JOIN, ORDER BY ASC
Q10	Relaciona a soma da população do continente com a média da taxa de perda de pacotes dos nós monitores no continente	Utiliza 1 fragmento da tabela Observation, Realiza 2 tipos de agregação (SUM e AVG), Agrupa em 1 grupo, Faz 4 Inner Joins	SUM DISTINCT, AVG, FILTER, JOIN, ORDER BY DESC

As consultas listadas no Quadro 4 foram executadas em cima das tabelas externas e internas utilizando os 3 conjuntos de dados de medidas já mencionados, com 100, 200 e 300 milhões de triplas, aproximadamente. Os experimentos foram conduzidos utilizando a abordagem *cold start* em um ambiente de rede dedicada, onde após cada execução de uma consulta, o banco de dados era reiniciado, limpando a sua memória *cache*. Assim, os

resultados de desempenho obtidos a partir da execução de uma consulta não foram influenciados pelas execuções anteriores.

Todas as consultas SQL foram executadas a partir do Impala-shell e a técnica utilizada para realizar junções entre tabelas foi a de *broadcast*, técnica padrão adotada pelo Impala onde a tabela do lado direito do *join* é considerada menor do que a tabela do lado esquerdo, e os seus conteúdos são enviados para todos os nós envolvidos na consulta. Na Tabela 9 são mostrados os tempos de execução (em segundos) de cada consulta em cada tipo de tabela testada utilizando 3 conjuntos de dados de tamanhos diferentes.

Tabela 9: Tempos de execução de cada consulta SQL em cima de tabelas externas e internas utilizando conjuntos de dados de medidas com 100M, 200M e 300M de triplas, aproximadamente

Consulta	**Tabela Externa (TSV)**			**Tabela Interna (Parquet/Snappy)**			**Tabela Interna (Parquet/Gzip)**		
	100M	**200M**	**300M**	**100M**	**200M**	**300M**	**100M**	**200M**	**300M**
Q1	25.31s	50.49s	52.53s	0.81s	0.93s	1.10s	0.87s	0.96s	1.20s
Q2	24.29s	51.88s	51.95s	0.73s	0.83s	1.03s	0.75s	0.92s	1.06s
Q3	25.47s	69.72s	77.57s	1.82s	1.98s	2.25s	1.85s	2.03s	2.27s
Q4	24.97s	51.99s	54.22s	1.06s	1.47s	1.53s	1.08s	1.49s	1.63s
Q5	27.39s	50.56s	53.12s	1.10s	1.63s	1.90s	1.14s	1.73s	2.06s
Q6	24.11s	52.11s	53.93s	1.50s	2.26s	2.99s	1.63s	2.58s	3.08s
Q7	25.49s	52.04s	54.66s	0.62s	0.81s	1.03s	0.71s	0.82s	1.11s
Q8	26.25s	51.89s	53.04s	1.10s	1.53s	1.99s	1.18s	1.60s	2.07s
Q9	23.90s	52.07s	54.65s	1.27s	1.86s	2.55s	1.39s	1.93s	2.61s
Q10	25.29s	52.88s	54.21s	1.59s	2.18s	3.07s	1.68s	2.27s	3.08s

A partir dos resultados expostos chega-se à conclusão que a abordagem de tabela interna utilizando compressão Snappy apresentou tempos de execução menores que os apresentados pelas outras abordagens, considerando todas as consultas realizadas nos 3 conjuntos de dados testados.

O desempenho da abordagem de tabela externa se mostrou bastante insatisfatória. Isso se deve ao fato desta armazenar os arquivos de dados no formato nativo, sem compressão, e não fazer o uso da estratégia de particionamento. Com isso, a taxa de leitura de dados foi bem maior do que a apresentada pelas outras abordagens para todos os tipos de consultas executadas, transferindo mais bytes a partir do disco para a memória, levando a produção de resultados de desempenho bastante inferiores.

Já a abordagem de tabela interna utilizando compressão Snappy obteve resultados um pouco melhores do que os apresentados pela compressão Gzip, considerando os 3 conjuntos de dados avaliados. Isso se deve ao fato de que a velocidade de descompressão e leitura de dados do algoritmo Snappy é bastante superior a apresentada pelo algoritmo Gzip, o que compensa o fato do algoritmo Snappy ser menos econômico que o Gzip. Logo, apesar do algoritmo Snappy gerar arquivos de dados maiores, transferindo uma taxa maior de bytes a partir do disco para a memória, ele ainda consegue apresentar tempos de execução de consultas menores que os apresentados pelo algoritmo Gzip, devido a sua alta velocidade de leitura de dados.

O desempenho superior do algoritmo Snappy em relação ao algoritmo Gzip é citado em (CLOUDERA, 2015), no qual é apontada a rápida taxa de descompressão do algoritmo Snappy, apresentando uma velocidade de leitura de dados bastante eficiente. Entretanto, sua velocidade de escrita de dados é apontada como não muito eficiente. Ambas afirmativas foram comprovadas através deste experimento.

Analisando individualmente o desempenho de cada consulta, observou-se que as consultas Q1 e Q2 apresentaram alguns dos menores tempos de execução, tanto para as abordagens Snappy quanto para Gzip, para os 3 conjuntos de dados avaliados. Isso se deve ao fato destas consultas serem do tipo *star-queries*, que buscam por múltiplas propriedades de uma mesma classe de sujeito, consultando somente a tabela proprietária referente a classe do sujeito tratado, sem necessitar realizar nenhuma operação de junção com outra tabela.

As consultas Q3 a Q10 são consultas do tipo *chain-queries*, que buscam por propriedades de sujeitos pertencentes a classes diferentes e por isso precisam realizar operações de junção entre diferentes tabelas proprietárias de classe, onde a articulação é feita a partir do sujeito de uma tripla e o objeto de outra tripla (KHODABAKHSH, KHATAMI, 2015).

As consultas que apresentaram alguns dos maiores tempos de execução foram Q3, por utilizar um operador UNION combinado com um operador de agregação, Q6 e Q10, por realizar uma alta agregação dos dados, produzindo resultados agregados utilizando a granularidade de Continente e Q8 e Q9, por não fazer uso da estratégia de fragmentação da tabela Observation, percorrendo todas suas partições. As consultas Q4 e Q7 apresentaram baixos tempos de execução, por percorrerem somente uma partição da tabela Observation. A consulta Q5 é idêntica à consulta Q4, mas percorre 3 partições da tabela Observation, processando uma quantidade maior de dados, com isso os tempos de execução obtidos foram um pouco maiores que os obtidos a partir da consulta Q4.

6.3.1.4 Considerações gerais

Diante dos resultados apresentados durante esta seção, chega-se a conclusão que o tipo de tabela mais adequado para ser adotado na implementação final da aplicação *Impala MultiLOD* é o de tabela interna utilizando o formato de arquivo de dados Parquet e algoritmo de compressão Snappy. Esta abordagem apresentou os melhores resultados de desempenho durante a execução do conjunto de consultas SQL. Além disso, as diferenças entre as velocidades de escrita de dados apresentadas pelo algoritmo Snappy e Gzip não foram muito relevantes. Também não foi muito relevante a economia de espaço em disco na criação de arquivos de dados, onde a diferença dos tamanhos dos arquivos gerados por ambos algoritmos não ultrapassou 350MB. Logo, o que realmente foi considerado foi a influência positiva da compressão Snappy sobre o desempenho do sistema.

6.3.2 Teste do desempenho do 2º componente da aplicação (Tradutor de Linguagem SPARQL para SQL)

Para realizar a avaliação do desempenho do tradutor de linguagem SPARQL para SQL criado e validar os resultados produzidos por este tradutor foi utilizado o mesmo conjunto de consultas apresentado no Quadro 4, porém com as consultas escritas na linguagem SPARQL. As consultas SPARQL serviram de entrada para o tradutor, sendo traduzidas automaticamente para a linguagem SQL e executadas diretamente no Impala. Após isso, as respostas obtidas a partir da execução das consultas traduzidas foram comparadas com as respostas obtidas na subseção anterior, Seção 6.3.1, onde as consultas escritas diretamente na linguagem SQL foram executadas no Impala, sem a intermediação do tradutor.

O conjunto de consultas SPARQL utilizado está listado no Apêndice D. As métricas que foram consideradas para testar o tradutor SPARQL para SQL construído foram:

1. Tempo de execução de consultas analíticas SPARQL tipicamente executadas sobre os dados do PingER
2. Corretude das respostas produzidas a partir da execução das consultas traduzidas

6.3.2.1 Tempo de execução do conjunto de consultas analíticas SPARQL

Na Tabela 10 são mostrados os tempos de execução (em segundos) de cada consulta SPARQL traduzidas e executadas no Impala a partir do intermédio do tradutor, utilizando 3 conjuntos de dados de tamanhos diferentes. Os experimentos também foram conduzidos adotando a abordagem *cold start* em um ambiente de rede dedicada. Lembrando, que os tempos de execução apresentados não levam em conta o tempo de tradução das consultas, que, em geral, foram intervalos de tempo muito pequenos, que não afetaram o desempenho do tradutor.

Tabela 10: Tempos de execução de cada consulta SPARQL traduzidas e executadas a partir do tradutor utilizando conjuntos de dados de medidas com 100M, 200M e 300M de triplas, aproximadamente

Consulta	**Tradutor - Tabelas Proprietárias Internas (Parquet/Snappy)**		
	100M	**200M**	**300M**
Q1	0.82s	0.95s	1.11s
Q2	0.73s	0.85s	1.04s
Q3	3.62s	6.87s	9.85s
Q4	1.06s	1.48s	1.56s
Q5	1.13s	1.63s	1.98s
Q6	1.59s	2.32s	3.03s
Q7	0.62s	0.86s	1.09s
Q8	1.10s	1.54s	1.99s
Q9	1.29s	1.88s	2.58s
Q10	1.59s	2.22s	3.10s

Os gráficos ilustrados na Figura 33 apresentam a comparação entre os tempos de execução obtidos a partir da execução das consultas traduzidas para SQL através do tradutor e a execução das mesmas consultas escritas diretamente em SQL, executadas através do Impala-shell sem a intermediação do tradutor, utilizando a abordagem de tabela interna com compressão Snappy.

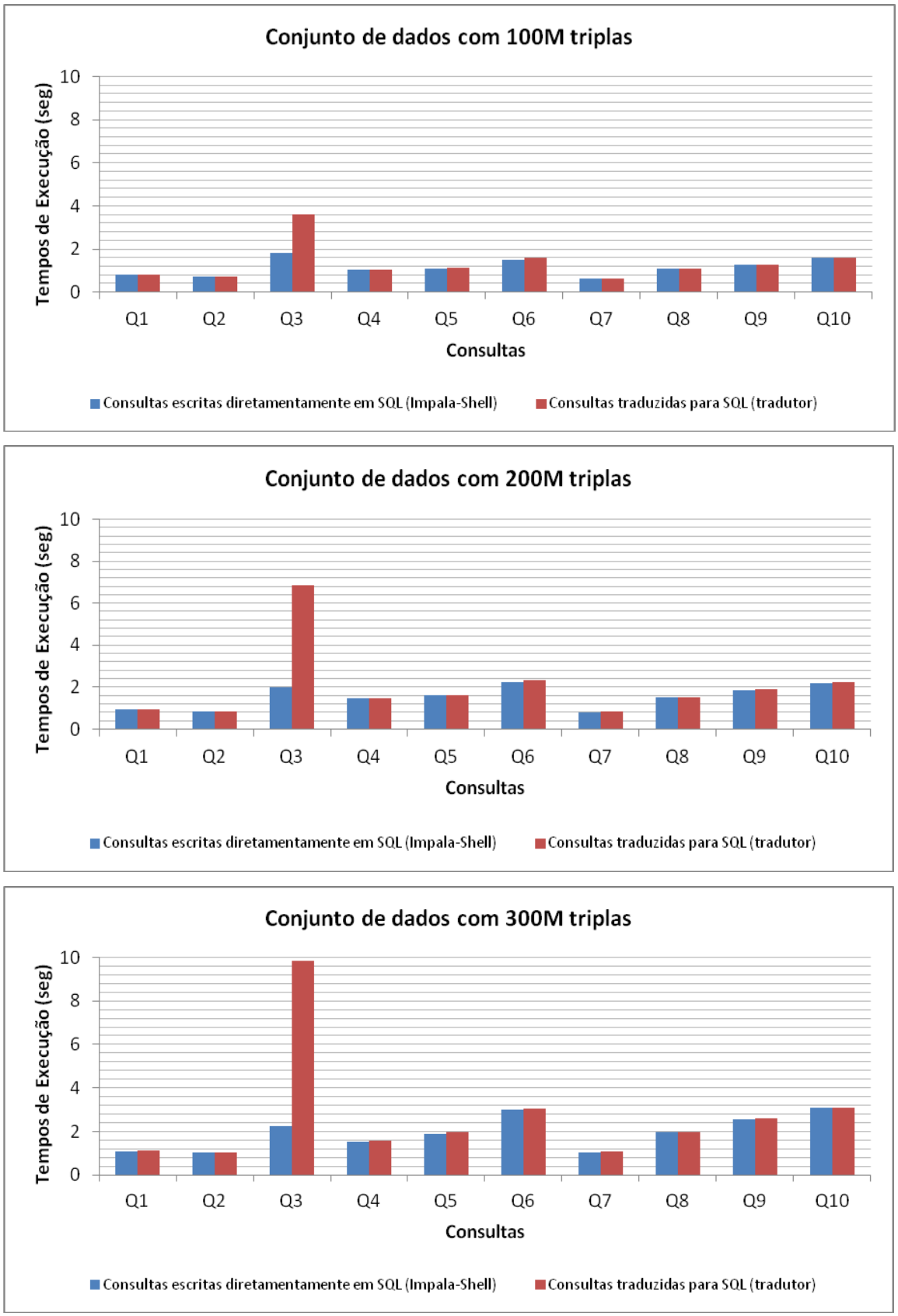

Figura 33: Comparação dos tempos de execução das consultas SPARQL traduzidas para SQL a partir do tradutor com os tempos de execução das mesmas consultas escritas diretamente em SQL executadas sobre 3 conjuntos de dados com 100M, 200M e 300M triplas, aproximadamente

A partir da análise da Tabela 10 e dos gráficos comparativos, chega-se à conclusão que os intervalos de tempo obtidos a partir da execução das consultas traduzidas se aproximaram aos obtidos a partir da execução das mesmas consultas escritas diretamente em SQL e executadas diretamente no Impala-shell, apresentando uma diferença média de 0,024s. Mostrando que a lógica de tradução praticada pela aplicação e o estabelecimento de conexão com o Impala via *driver* JDBC não afetaram negativamente o desempenho da aplicação, na maioria dos casos.

Entretanto, também é possível notar a partir da análise dos gráficos que a execução da consulta Q3 traduzida para SQL, a partir do tradutor, apresentou tempos de execução mais longos que os apresentados quando a mesma consulta escrita diretamente em SQL foi executada, para os 3 conjuntos de dados avaliados, registrando uma diferença média de 4,76s.

A queda do desempenho da consulta traduzida pode ser explicada a partir da complexidade de tradução da consulta Q3, que utiliza um operador UNION combinado com um operador de agregação, onde as duas subconsultas que são unidas são do tipo *chain-queries* e executam cada uma 3 *joins* entre tabelas diferentes. Logo, de acordo com a lógica de tradução implementada, cada uma das subconsultas unidas pelo operador UNION é traduzida em 5 subconsultas, onde 4 destas realizam seleções sobre 4 tabelas proprietárias de classe diferentes, seleções que são articuladas a partir de operadores JOIN. E uma subconsulta envolve todas as outras 4 subconsultas, onde seus seletores são todas as colunas presentes nas cláusulas SELECT dos grupos juntados, facilitando a referência aos resultados da junção. A tradução da consulta Q3 para SQL pode ser visualizada no Apêndice E desta dissertação.

Verificou-se que o gargalo da consulta Q3 traduzida não está na execução de cada uma das subconsultas unidas pelo operador UNION e sim no processo de união e agregação dos resultados das subconsultas.

6.3.2.2 Corretude das respostas às consultas traduzidas

A Figura 34 mostra a resposta à execução da consulta Q1 do conjunto de consultas SQL executada diretamente no Impala-shell, sem a intermediação do tradutor. A Figura 35 mostra a resposta à execução da mesma consulta Q1, porém, escrita na linguagem SPARQL e traduzida para SQL a partir do tradutor. É possível notar que as duas formas de execução produziram os mesmos resultados. Em todos os casos em que foram executadas consultas traduzidas a partir do tradutor, foram produzidas respostas iguais às respostas obtidas a partir da execução da mesmas consultas, escritas diretamente na linguagem SQL, executadas a partir do Impala-shell, o que valida a lógica de tradução implementada.

```
[node3.crbd.ufrj.br:21000] > SELECT PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode, MAX(PingER_ont_hasValue) as maximo
                           >
                           > FROM qb_Observation300M_Snappy
                           >
                           > WHERE qb_measureType = '<http://www-iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>'
                           >
                           > GROUP BY PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode
                           >
                           > ORDER BY maximo DESC
                           >
                           > LIMIT 10;
Query: select PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode, MAX(PingER_ont_hasValue) as maximo
FROM qb_Observation300M_Snappy
WHERE qb_measureType = '<http://www-iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>'
GROUP BY PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode
ORDER BY maximo DESC
LIMIT 10
+------------------------------------------------------------------------------------------+----------------------------------------------------------------------------------------+--------+
| pinger_ont_hassourcenode                                                                 | pinger_ont_hasdestinationnode                                                          | maximo |
+------------------------------------------------------------------------------------------+----------------------------------------------------------------------------------------+--------+
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-rainbow.inp.nsk.su>          | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-ping.slac.stanford.edu>    | 97.556 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-pingerlhr-pu.pern.edu.pk>    | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-www.kcn.unima.mw>          | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-ford.cefet-rj.br>            | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-mail.qanet.gm>             | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-pinger.iba-suk.edu.pk>       | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-www.afe.mr>                | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-pinger.cemb.edu.pk>          | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-tg.refer.org>              | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-ford.cefet-rj.br>            | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-ping.cern.ch>              | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-ford.cefet-rj.br>            | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-lbl.csc.cuhk.edu.hk>       | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-sau.seecs.edu.pk>            | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-www.kcn.unima.mw>          | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-quest.seecs.edu.pk>          | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-www.afe.mr>                | 96.774 |
| <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-pinger.comsats.edu.pk>       | <http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-ftp.physics.carleton.ca>   | 96.774 |
+------------------------------------------------------------------------------------------+----------------------------------------------------------------------------------------+--------+
```

Figura 34: Consulta Q1 escrita diretamente na linguagem SQL executada a partir do Impala-shell

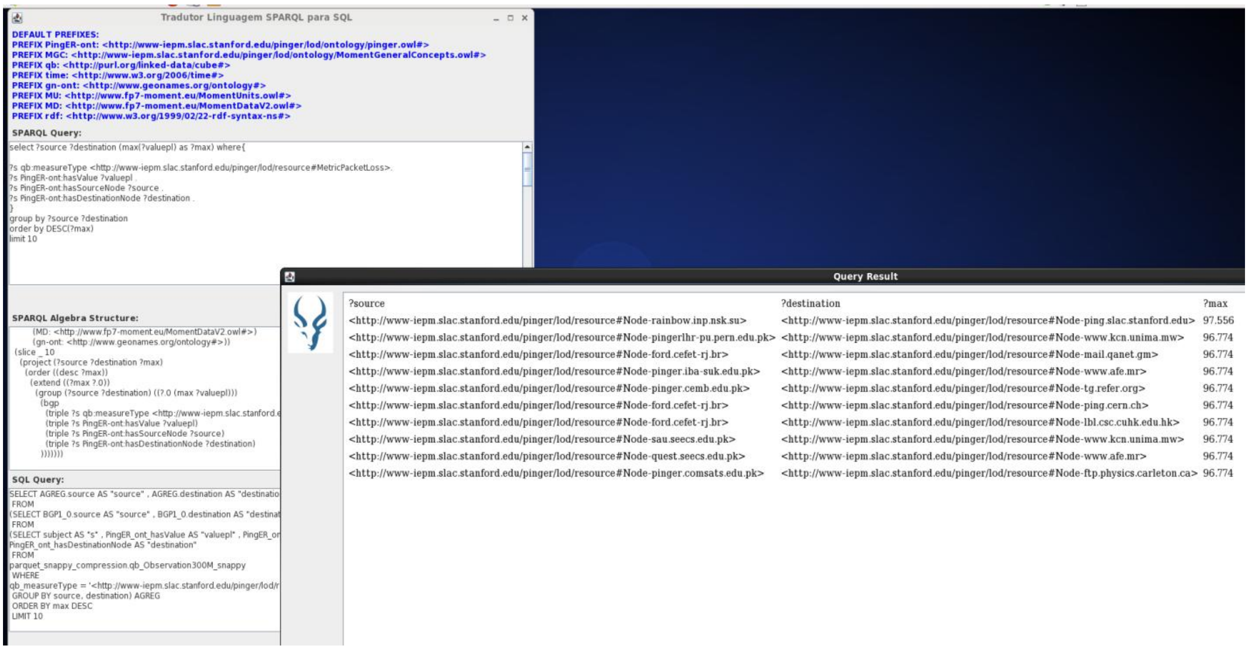

Figura 35: Consulta Q1 escrita na linguagem SPARQL, traduzida para SQL e executada a partir do tradutor de linguagem SPARQL para SQL

6.3.2.3 Considerações gerais

Diante dos resultados apresentados durante esta seção, chega-se a conclusão que a tradução automática das consultas SPARQL para a linguagem SQL executada a partir do tradutor não afetou negativamente o desempenho das consultas, com exceção da tradução da consulta Q3, que utiliza uma cláusula UNION combinada com um operador de agregação. A execução da consulta Q3 traduzida apresentou resultados de desempenho inferiores aos obtidos a partir da execução das outras consultas traduzidas, fato que mostra que ainda existe

um potencial para a otimização do módulo do tradutor responsável pela tradução do operador UNION.

Além disso, verificou-se que todas as consultas traduzidas, sem exceções, geraram respostas corretas ao serem executadas, o que valida a lógica de tradução implementada.

6.4 COMPARAÇÃO COM O VIRTUOSO TRIPLE STORE

Os resultados de desempenho obtidos a partir da avaliação da aplicação *Impala MultiLOD* foram comparados com os do sistema Virtuoso Triple Store, repositório de triplas apoiado por SGBDR que foi testado de forma distribuída no mesmo ambiente computacional e com os mesmos conjuntos de dados utilizados na aplicação *Impala MultiLOD*.

A solução de armazenamento adotada pelo Virtuoso Triple Store é convencional e gira em torno de uma única tabela (DB.DBA.RDF_QUAD) com 4 colunas, que armazena um "quad", i.e. uma tripla mais o nome de seu grafo, por linha. As colunas representam o grafo (G), sujeito (S), predicado (P) e objeto (O). As colunas G, S e P são do tipo IRI_ID que referência uma coluna na tabela DB.DBA.RDF_IRI, que armazena todas as IRIs (também chamadas de URIs) distintas presentes nos grafos. A coluna O é do tipo ANY. Se o valor de O não for uma *string*, como um número ou data ou IRI, ele é armazenado em sua representação binária nativa. Se for uma *string* "muito curta" (20 caracteres ou menos), ele também é armazenado na sua forma nativa. Se for uma *string* longa ou um literal RDF com tipo não definido, seu valor é armazenado na tabela DB.DBA.RDF_OBJ (ERLING, MIKHAILOV, 2009).

A partir da versão 6.0.0, o Virtuoso Triple Store já possui configurado um esquema de índices padrão composto por 2 índices totais (PSOG e POGS) e 3 índices parciais (SP, OP e GS). A ideia é favorecer as consultas que possuem os predicados especificados nos padrões básicos de grafo, assim um "quad" pode ser acessado de forma eficiente quando um P e pelo menos um S ou O são conhecidos (ERLING, MIKHAILOV, 2009).

Foi utilizado o Virtuoso Universal Server[39], versão 7.2, para montar um *cluster* com quatro máquinas (mesmas máquinas utilizadas no *cluster* Hadoop introduzido na Seção 4.4), onde cada máquina executou uma instância do Vituoso Triple Store. A configuração do *cluster* foi declarada em um arquivo "cluster.ini" que foi armazenado no diretório inicial do Virtuoso de cada uma das máquinas que compõem o *cluster*. Este arquivo declara os endereços e portas de todos os nós que compõem o *cluster* e quais destes nós é o nó local e

[39] http://docs.openlinksw.com/virtuoso/

qual é o nó mestre. Cada nó no *cluster* tem seu próprio banco de dados e arquivos de log. Adicionalmente, cada nó possui um porta de escuta que é usado para comunicações com os outros nós.

A comparação entre os desempenhos gerais verificados a partir da avaliação do Virtuoso Triple Store distribuído e da aplicação *Impala MultiLOD* foi feita a partir das seguinte métricas de desempenho:

1. Tempo para carga dos conjuntos de dados
2. Tamanho em disco ocupado pelo banco de dados
3. Tempo de execução de consultas analíticas SPARQL tipicamente executadas sobre os dados do PingER

6.4.1 Tempo para carga dos conjuntos de dados

Os mesmos arquivos no formato N-Triples carregados no HDFS, conforme explicado na Seção 6.2.4, foram carregados, inicialmente, em somente um nó do *cluster* Virtuoso. A carga dos dados foi feita utilizando o mecanismo de *bulk loading*, no qual múltiplas linhas de dados (blocos de dados) são inseridas no banco de dados de uma vez só. Para isso, foi necessário configurar o Virtuoso, no nó local, para que ele passasse a usar uma quantidade suficiente de memória que viabilizasse o procedimento, sendo configurado para usar entre 2/3 e 3/5 da capacidade de memória RAM das máquinas.

A Tabela 11 mostra os tempos necessários para carga de cada arquivo N-Triples utilizando a estratégia de *bulk loading* no nó local. É possível notar que foram obtidos resultados bastante satisfatórios, onde os tempos de carga foram até menores que os observados quando os mesmos arquivos foram carregados no banco de dados distribuído HDFS. Entretanto, é importante ressaltar que a estratégia de *bulk loading* é custosa. Quando executada, consome praticamente todos os recursos de memória RAM da máquina, inviabilizando a execução de outras tarefas em paralelo no sistema.

Tabela 11: Estatísticas dos processos de carga de arquivos no formato N-Triples no Virtuoso Triple Store utilizando a estratégia de *bulk loading*

Arquivo	Tempo para carga no Virtuoso	Tamanho do arquivo
Arquivo de Dados Gerais	2.4s	7.6 MB
Arquivo de medidas de rede com 100M de triplas	12min 31s	26.9 GB
Arquivo de medidas de rede com 200M de triplas	28min 38s	51.7 GB
Arquivo de medidas de rede com 300M de triplas	44min 2s	76.4 GB

Após a carga dos dados em um banco de dados de somente um nó do *cluster*, este banco de dados inicial foi copiado para todos os outro nós. Assim, todos os bancos de dados no *cluster* compartilhavam precisamente o mesmo esquema. Dentro de um *cluster* Virtuoso, as tabelas dos bancos de dados podem ser declaradas como particionadas ou replicadas. Caso a tabela seja declarada como particionada, cada partição da tabela será armazenada em um único nó do *cluster*. Caso a tabela seja declarada como replicada, a tabela e todo seu conteúdo irão existir em todos os nós do *cluster*.

A desvantagem de se declarar uma tabela sem partições dentro do *cluster* é que a tabela será armazenada em todos os nós com conteúdo independente. Isso geralmente não é desejável, pois as consultas feitas sobre a tabela serão respondidas a partir de dados locais. Logo, uma mesma consulta poderá retornar dados diferentes dependendo do nó em que será executada.

Diante das desvantagens citadas, após a cópia do banco de dados inicial para todos os nós do *cluster*, foi realizado o procedimento de partição das tabelas entre os nós. As partições das tabelas foram declaradas nos arquivos de configuração "virtuoso.ini" de cada nó do *cluster*. As linhas das tabelas que não pertenciam às partições gerenciadas por cada nó foram deletadas.

6.4.2 Tamanho em disco ocupado pelo banco de dados

Antes de realizar o procedimento de cópia do banco de dados inicial para todos os nós do *cluster* e a partição das tabelas foi realizada a verificação do tamanho do banco de dados, olhando o tamanho do arquivo “virtuoso.db”. O tamanho do arquivo observado após a carga de todos os dados foi de 6.9GB, ocupando um espaço em disco maior que o ocupado por todas as tabelas da aplicação *Impala MultiLOD*, as quais ocupam juntas 1.3GB em disco.

6.4.3 Tempo de execução do conjunto de consultas analíticas SPARQL

O mesmo conjunto de consultas apresentado no Quadro 4, porém com as consultas escritas na linguagem SPARQL, listadas no Apêndice D desta dissertação, foi utilizado para testar o desempenho do Virtuoso Triple Store distribuído. Na Tabela 12 são mostrados os tempos de execução (em segundos) de cada consulta SPARQL utilizando 3 conjuntos de dados de tamanhos diferentes.

As consultas foram executadas no nó mestre do *cluster* a partir da linha de comando ISQL, utilitário do Virtuoso que permite a execução de consultas SPARQL e outros *scripts*. Os experimentos também foram conduzidos adotando a abordagem *cold start* em um ambiente de rede dedicada.

Tabela 12: Tempos de execução das consultas SPARQL executadas a partir do Virtuoso Triple Store carregado com conjuntos de dados de medidas com 100M, 200M e 300M de triplas, aproximadamente

Consulta	**Virtuoso Triple Store**		
	100M	**200M**	**300M**
Q1	0.96s	1.22s	1.48s
Q2	0.88s	0.99s	1.18s
Q3	4.18s	7.83s	11.15s
Q4	1.15s	1.56s	1.67s
Q5	1.29s	1.81s	2.20s
Q6	1.72s	2.51s	3.22s
Q7	0.75s	0.99s	1.20s
Q8	1.25s	1.73s	2.27s
Q9	1.43s	2.04s	2.72s
Q10	1.77s	2.43s	3.33s

Os gráficos ilustrados na Figura 36 apresentam a comparação entre os tempos de execução obtidos a partir da execução das consultas SPARQL diretamente no Virtuoso e a execução das mesmas consultas SPARQL traduzidas para SQL e executadas no Impala a partir da aplicação *Impala MultiLOD*, que teve seu desempenho avaliada nas seções anteriores.

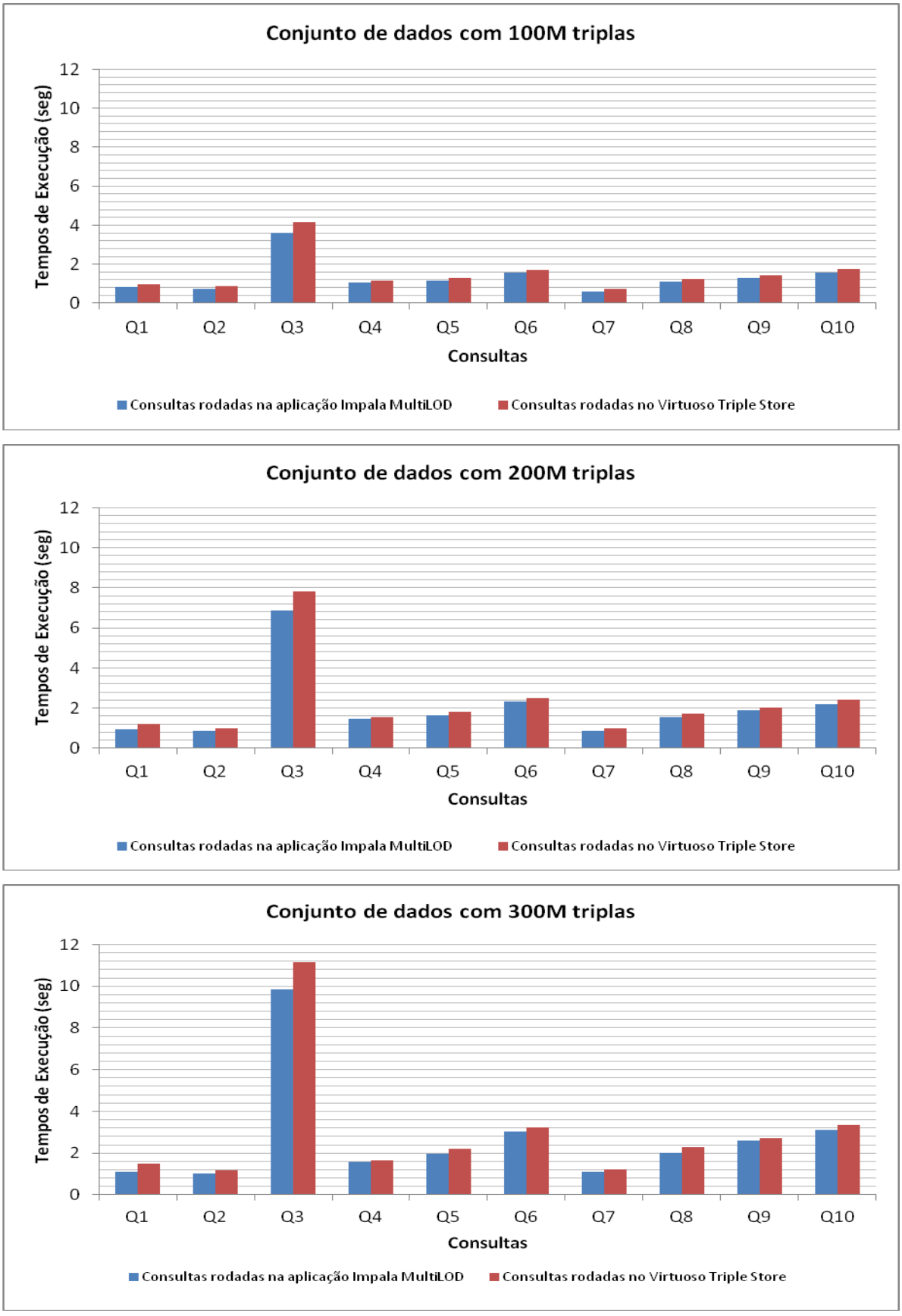

Figura 36: Comparação dos tempos de execução das consultas SPARQL traduzidas e executadas a partir da aplicação *Impala MutiLOD* com os tempos de execução das mesmas consultas SPARQL executadas a partir do Virtuoso Triple Store utilizando três conjuntos de dados com 100M, 200M e 300M triplas, aproximadamente

A partir da análise da Tabela 12 e dos gráficos comparativos, chega-se à conclusão de que os tempos de respostas obtidos a partir da execução das consultas SPARQL no *cluster* Virtuoso foram superficialmente maiores do que os obtidos a partir da execução das mesmas consultas SPARQL traduzidas para SQL e executadas no Impala a partir do intermédio da aplicação *Impala MultiLOD*, para todas as consultas avaliadas. As diferenças entre os intervalos de tempos aumentaram à medida que as consultas foram executadas em conjuntos de dados com tamanhos maiores.

É possível notar que os resultados de desempenho obtidos a partir da execução das consultas SPARQL no Virtuoso seguiram o mesmo padrão obtido a partir da execução das consultas no Impala. As consultas Q1 e Q2 continuaram apresentando os menores tempos de execução, por serem consultas do tipo *star-queries*, que buscam por múltiplas propriedades de uma mesma classe de sujeito, sendo consultas mais seletivas. Já as consultas Q6, Q8, Q9 e Q10 continuaram apresentando alguns dos maiores tempos de execução, por se tratarem de consultas do tipo *chain-queries*, que realizam diversas operações de junção e trabalham com os dados em um nível alto de agregação.

Também é importante observar que a consulta Q3, que obteve os piores resultados de desempenho na avaliação da aplicação *Impala MultiLOD*, apresentou tempos de resposta também pouco satisfatórios quando executada no Virtuoso, para todos os conjuntos de dados avaliados. Mostrando que a complexidade da consulta Q3 também interferiu nos resultados de desempenho obtidos no Virtuoso.

6.4.4 Considerações gerais

Diante dos resultados apresentados durante esta seção, chega-se à conclusão de que o Virtuoso Triple Store, executado de forma distribuída no mesmo ambiente computacional que a aplicação *Impala MultiLOD*, apresentou resultados de desempenho inferiores aos apresentados pela aplicação *Impala MultiLOD* na execução de consultas SPARQL. Esse fato justifica a escolha inicial pela utilização da solução *Impala MultiLOD* como mecanismo para execução de consultas analíticas sobre os dados do projeto PingER.

Apesar do desempenho inferior apresentado pelo Virtuoso na execução de consultas SPARQL, vale ressaltar o bom desempenho do sistema ao executar a carga de grandes conjuntos de dados em um único nó do *cluster* utilizando a estratégia de *bulk loading*, apresentando tempos de carga bastante eficientes.

7 CONCLUSÃO

Nesta dissertação foi proposta uma abordagem para apoiar a execução de consultas analíticas SPARQL sobre grandes volumes de dados multidimensionais expressos no formato de grafos RDF. Para esse fim foi desenvolvida uma aplicação, chamada *Impala MultiLOD*, que permite a execução de consultas analíticas SPARQL de forma distribuída, se baseando, para isso, em tecnologias de distribuição e de armazenamento oferecidas pelo *framework* Apache Hadoop. A aplicação foi estruturada a partir de dois componentes:

- 1º componente (Carga e Preparação do Banco de Dados): Componente responsável pela carga dos dados RDF em tabelas no Impala, seguindo o esquema de armazenamento de Tabela Proprietária de Classe. De acordo com o que foi discutido na Seção 5.2.1, este tipo de esquema foi o que mais apresentou vantagens em relação aos outros ao modelar dados RDF com características multidimensionais para processamento analítico e armazenamento no Impala. Neste tipo de esquema, sujeitos de uma mesma classe, identificada a partir da propriedade `rdf:type`,são armazenados juntos em uma mesma tabela. Como efeito, todas as tabelas armazenam o atributo `rdf:type` para indicar a classe que está sendo representada pela tabela. Além do `rdf:type` e do sujeito como chave, a tabela proprietária também pode armazenar outras propriedades ligadas aos sujeitos da classe como atributos.
- 2º componente (Tradutor de Linguagem SPARQL para SQL): Componente responsável pela tradução de consultas SPARQL, geradas pelos usuários finais da aplicação a partir de uma interface gráfica, em consultas SQL equivalentes, executadas diretamente no Impala a partir de conexão via *driver* JDBC. O resultado das consultas é exibido no formato de triplas RDF. A versão do SPARQL com a qual o tradutor trabalha é a 1.1, que oferece suporte a operadores de agregação. O funcionamento do tradutor e a lógica de tradução implementada foram explicados na Seção 5.3 desta dissertação.

A aplicação foi testada utilizando conjuntos de dados de medidas reais originados do projeto PingER (COTTRELL, 2001), detalhado na Seção 2.3, que monitora o desempenho de *links* de Internet ao redor do mundo, utilizando mais de 10 métricas de rede diferentes. Os experimentos foram realizados em um *cluster* composto por quatro máquinas, sendo utilizada a distribuição aberta do Apache Hadoop: CDH versão 5.3.2. Na Seção 4.4 foram detalhadas as características do *cluster* montado. Após terem sido realizados os experimentos na

aplicação, detalhados no Capítulo 6, os resultados de desempenho obtidos foram comparados com os do sistema Virtuoso Triple Store, que foi carregado e testado de forma distribuída no mesmo ambiente computacional e com os mesmos conjuntos de dados utilizados na aplicação *Impala MultiLOD*. Os resultados obtidos após o término dos experimentos foram resumidos a seguir:

- A etapa de triplificação dos dados foi a mais demorada, devido ao grande volume de dados que foram manipulados nos *workflows* ETC criados e o fato de que a execução dos *workflows* foi feita de forma não distribuída, em uma única máquina.
- A carga dos arquivos RDF, gerados a partir da etapa de triplificação, no HDFS foi feita a partir de um *job* no Kettle, que permite copiar arquivos armazenados de forma local para dentro do *cluster* Hadoop. A carga foi realizada em intervalos de tempos aceitáveis, levando uma média de 1min 36s para carga de 1GB de dados no HDFS.
- A opção de tabela interna com formato de arquivo de dados Parquet e algoritmo de compressão Snappy foi a que apresentou os melhores resultados de desempenho durante a execução das consultas SQL no Impala. As diferenças entre as velocidades de escrita de dados e a capacidade de economia de espaço em disco na criação de arquivos de dados apresentadas pelos algoritmos Snappy e Gzip não foram muito relevantes.
- A opção por realizar o particionamento de dados em algumas das tabelas criadas mostrou-se vantajosa. Verificou-se que as consultas que buscaram em somente uma partição de uma tabela, caso da consulta Q4, apresentaram tempos de resposta menores do que os apresentados pelas consultas que buscaram em mais de uma partição ou tiveram que percorrer todas as partições de uma tabela, caso da consulta Q5.
- A tradução automática das consultas SPARQL para a linguagem SQL executada a partir do tradutor não afetou negativamente o desempenho das consultas, na maioria dos casos. As consultas traduzidas, sem exceções, geraram respostas corretas ao serem executadas no Impala.
- A execução da consulta Q3 traduzida apresentou resultados de desempenho inferiores aos obtidos a partir da execução das outras consultas SPARQL traduzidas, fato que mostra que ainda existe um potencial para otimização do módulo do tradutor responsável pela tradução do operador UNION.

- O Virtuoso Triple Store, executado de forma distribuída, apresentou resultados de desempenho inferiores aos apresentados pela aplicação *Impala MultiLOD* na execução das consultas SPARQL. A diferença nos resultados de desempenho foi maior quando o sistema foi avaliado utilizando conjuntos de dados de tamanhos crescentes.
- O Virtuoso Triple Store apresentou tempos de carga bastante eficientes ao efetuar a carga de grandes conjuntos de dados em um único nó do *cluster* utilizando a técnica de *bulk loading.*

7.1 CONTRIBUIÇÕES

Até o momento tem havido poucos esforços no sentido de aprimorar a atividade de processamento analítico de dados multidimensionais cuja estrutura subjacente é baseada em grafo. Uma área importante que tende a ganhar força com o amadurecimento e inserção das iniciativas de Web Semântica e LOD no mercado, onde as organizações possuem a necessidade de realizar análises complexas sobre dados disponíveis a partir de diversas fontes, baseando tomadas de decisão de apoio.

Diante deste cenário, esta dissertação propôs uma abordagem para apoiar a execução de consultas SPARQL analíticas sobre grandes volumes de dados multidimensionais em formato RDF. Nossa abordagem se baseou em tecnologias de distribuição já existentes para permitir a execução das consultas SPARQL de forma distribuída. Foram realizados experimentos com conjuntos de dados de medidas reais representados em um nível de granularidade baixo, o que permitiu a geração de consultas analíticas complexas e especializadas. Para todas as consultas SPARQL executadas a partir da aplicação foram obtidos tempos de resposta na ordem de segundos, considerando todos os conjuntos de dados de diferentes tamanhos testados.

Além da construção de uma aplicação sobre o Impala que permitiu a execução e avaliação de consultas SPARQL analíticas sobre grandes volumes de dados multidimensionais em RDF armazenados no HDFS, podem ser citadas como outras contribuições desta dissertação:

- Compilação de materiais e aprendizado na construção, gerência e manutenção de um cluster físico Hadoop.
- Aplicação prática de técnicas de banco de dados distribuídos sobre um cenário real de muitos dados.

- Estudo da sintaxe e álgebra das linguagens SPARQL e SQL e implementação de uma lógica de tradução de consultas SPARQL para SQL.
- Adaptação da ontologia do domínio do projeto PingER tornando-a compatível com a estrutura do vocabulário RDF Data Cube.
- Submissão e aceitação do artigo "Measurement Ontology Pattern Language Applied to Network Performance Measurement" no Seminário de Pesquisa em Ontologias do Brasil (Ontobras 2015), São Paulo, Brazil, 2015.
- Submissão do artigo "An Approach for Analytical Processing of Large Volumes of Graph Data" ao IEE Intelligent Systems Journal - Special Issue on Big Data, 2016.

7.2 LIMITAÇÕES

A seguir são listadas as principais limitações encontradas na dissertação:

- Os experimentos realizados sobre a aplicação desenvolvida nesta dissertação foram restritos a um cenário de avaliação (cenário dos dados do projeto PingER). Com isso todos os resultados e conclusões obtidos foram restritos a esse cenário.
- Dentro do estudo de caso proposto que utilizou os dados do projeto PingER, a solução final desenvolvida nesta dissertação não foi avaliada por usuários reais do projeto.
- A solução proposta nesta dissertação foi testada utilizando somente um tipo de SGBD, no caso o Cloudera Impala.

7.3 TRABALHOS FUTUROS

Os trabalhos futuros podem ser divididos em trabalhos específicos para o aperfeiçoamento da aplicação *Impala MultiLOD* e trabalhos relacionados a novas possibilidades de experimentos que poderiam ser realizados na aplicação.

a) Aperfeiçoamento da aplicação *Impala MultiLOD*

- Os experimentos realizados sobre a aplicação mostraram que a tradução de uma consulta SPARQL complexa que utiliza um operador UNION não foi ideal e, portanto, deixa espaço para a otimização do módulo do tradutor responsável pela tradução do operador UNION e também de outros módulos que ainda possam ser otimizados.
- Exploração mais aprofundada do projeto de particionamento durante a criação de tabelas no Impala. Verificando se a criação de partições crescentes em tabelas fato

melhoraria o desempenho de consultas sobre este tipo de tabela, cujo tamanho tende a crescer progressivamente, aumentando com isso o número de partições da tabela.

- Criação de uma interface Web para a aplicação, que permita o acesso de múltiplos usuários ao mesmo tempo.

b) Novas possibilidades de experimentos

- Realizar experimentos utilizando uma versão mais nova do Cloudera Distributed Hadoop (CDH). A versão utilizada nos experimentos realizados nesta dissertação foi a 5.3.2. No último acesso realizado no site[40] oficial do Cloudera, em Junho de 2016, verificou-se que está disponível a versão 5.7.1.
- Realizar experimentos utilizando o serviço em nuvem da Amazon Elastic Compute Cloud (Amazon EC2), não sendo mais necessário manter um *cluster* físico.
- Testar a aplicação utilizando diferentes tipos de SGBD, como por exemplo o Apache Hive, e verificar qual apresenta o melhor desempenho.
- Carregar um volume maior de dados do projeto PingER na aplicação, atingindo a escala de bilhões de triplas e computar a sua escalabilidade.
- Computar a escalabilidade da aplicação quando múltiplos usuários realizam acesso simultâneo.
- Computar a escalabilidade da aplicação quando a quantidade de nós no *cluster* é variada.
- Fazer com que a aplicação desenvolvida nesta dissertação, carregada com os dados do projeto PingER, seja testada e avaliada por usuários finais e administradores de dados envolvidos no projeto.
- Realizar experimentos na aplicação em outros cenários de avaliação. Utilizando, por exemplo, os *benchmarks* OLAP TPC-H e TPC-DS (COUNCIL, 2005) mapeados para o formato RDF.

[40] http://www.cloudera.com/documentation/enterprise/release-notes/topics/rg_release_notes_cdh.html

REFERÊNCIAS

AUER, S. et al. **Sparqlify: a SPARQL-SQL rewriter**. 2015. Disponível em: <http://aksw.org/Projects/Sparqlify.html>. Acesso em: 13 jul. 2016.

BARBIERI, C. **BI2-Business intelligence**: modelagem & qualidade. Rio de Janeiro: Elsevier, 2011.

BARCELLOS, M. P.; FALBO, R. A.; FRAUCHES, V. Towards a measurement ontology pattern language. In: JOINT WORKSHOP ONTO.COM/ODISE ON ONTOLOGIES IN CONCEPTUAL MODELING AND INFORMATIONS SYSTEMS ENGINEERING, 1., 2014, Rio de Janeiro. **Proceedings**... [S.l.: s.n.], 2014.

BERNERS-LEE, T. **Semantic web road map**. 1998. Disponível em: <http://www.w3.org/DesignIssues/Semantic.html>. Acesso em: 20 jun. 2016.

_____. **Linked data**. 2009. Disponível em: <http://www.w3.org/DesignIssues/LinkedData.html>. Acesso em: 20 jun. 2016.

BIZER, C.; SEABORNE, A. **The D2RQ Platform-Treating Non-RDF Databases as Virtual RDF Graphs.** 2004. Disponível em: <http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/Bizer-D2RQ-ISWC2004.pdf>. Acesso em: 25 jul. 2016

BIZER, C.; SCHULTZ, A. The Berlin SPARQL benchmark. **International Journal on Semantic Web and Information Systems**, Hershey, v. 5, n. 2, p. 1-24, 2009.

BORTHAKUR, D. **HDFS architecture guide**. Maryland: Apache Software Foundation, 2008.

CASTERS, M.; BOUMAN, R.; VAN DONGEN, J. **Pentaho Kettle solutions**: building open source ETL solutions with Pentaho Data Integration. Indianapolis: Wiley, 2010.

CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and OLAP technology. **Sigmod Record**, New York, v. 26, n. 1, p. 65-74, 1997.

CHAUHAN, A. **Learning Cloudera Impala**. Birminghan: Packt Publishing, 2013.

CHONG, E. I. et al. An efficient SQL-based RDF querying scheme. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 31., 2005, Trondheim. **Proceedings**... [S.l.: s.n.], 2005. p. 1216-1227.

CLOUDERA. **Cloudera Enterprise 5.3.x Documentation**. 2016. Disponível em: <http://www.cloudera.com/content/www/en-us/documentation/enterprise/5-3-x/topics/cdh_intro.html>. Acesso em: 30 jun. 2016.

_____. **The Apache Impala Cookbook**. 2015. Disponível em: <http://blog.cloudera.com/blog/2014/12/the-impala-cookbook/>. Acesso em: 29 jun. 2016.

COTTRELL, L. **Internet End-to-end Performance Monitoring**. 2001. Disponível em: <http://www-iepm.slac.stanford.edu>. Acesso em: 29 jun. 2016.

COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. **Distributed systems**: concepts and design. 4. ed. Harlow; New York: Addison-Wesley, 2005.

COUNCIL, T. P. P. **Transaction processing performance council**. 2005. Disponível em: <www.tpc.org>. Acesso em: 30 jul. 2016

CUDRÉ-MAUROUX, P. et al. NoSQL databases for RDF: an empirical evaluation. In: ALANI, H. et al (Ed.). **The Semantic Web–ISWC 2013**: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II. Berlin: Springer, 2013. p. 310-325. (Lecture Notes in Computer Science, 8219).

CYGANIAK, R.; REYNOLDS, D.; TENNISON, J. **The RDF data cube vocabulary**. 2014. Disponível em: <http://www.w3.org/TR/vocab-data-cube/>. Acesso em: 11 jun. 2016.

DJURIC, D.; GASEVIC, D.; DEVEDZIC, V. The Tao of Modeling Spaces. **Journal of Object Technology**, [S.l.], v. 5, n. 8, p. 125-147, 2006.

ELLIOTT, B. et al. A complete translation from SPARQL into efficient SQL. In: INTERNATIONAL DATABASE ENGINEERING & APPLICATIONS SYMPOSIUM, 13., 2009, Calabria. **Proceedings**... New York: ACM, 2009. p. 31-42.

ERLING, O. Virtuoso, a Hybrid RDBMS/Graph Column Store. **IEEE Data Engineering Bulletin**, [S.l.], v. 35, n.1, p. 3-8, 2012.

ERLING, O.; MIKHAILOV, I. RDF Support in the Virtuoso DBMS. In: PELLEGRINI, T. et al. (Ed.). **Networked Knowledge-Networked Media**: integrating knowledge management, new media technologies and semantic systems. Berlin: Springer, 2009. p. 7-24. (Studies in Computational Intelligence, 221).

_____. **Business Intelligence Extensions for SPARQL**. 2009. Disponível em: <http://www.openlinksw.com/weblog/oerling/bisparql2.pdf>. Acesso em: 12 jul. 2016.

ETCHEVERRY, L.; VAISMAN, A. QB4OLAP: a new vocabulary for OLAP cubes on the semantic web. In: INTERNATIONAL WORKSHOP ON CONSUMING LINKED DATA, 3., 2012, Boston. **Proceedings**... New York: ACM, 2012. p. 27-38.

FINKELSTEIN, L.; LEANING, M. S. A review of the fundamental concepts of measurement. **Measurement**, Amsterdam, v. 2, n. 1, p. 25-34, 1984.

FUNDULAKI, I. et al. **D1. 1.1: Overview and Analysis of Existing Benchmark Frameworks**. Disponível em: <http://ldbcouncil.org/sites/default/files/LDBC_D1.1.1.pdf>. Acesso em: 12 maio 2016.

GHASEMI, S.; LUK, W.-S.; ALRAYES, N. M2RML: Multidimensional to RDF Mapping Language. In: INTERNATIONAL WORKSHOP ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 25., 2014, Munich. **Proceedings**... New York: IEEE, 2014. p. 263-267.

GRUBER, T. R. A translation approach to portable ontology specifications. **Knowledge Acquisition**, London, v. 5, n. 2, p. 199-220, 1993.

GUO, Y.; PAN, Z.; HEFLIN, J. LUBM: a benchmark for OWL knowledge base systems. **Web Semantics: Science, Services and Agents on the World Wide Web**, Amsterdam, v. 3, n. 2-3, p. 158-182, 2005.

HAQUE, A.; PERKINS, L. **Distributed RDF triple store using HBase and hive**. 2012. Disponível em: <https://www.albert.cm/dl/cloud_rdf_paper.pdf>. Acesso em: 30 maio 2016.

HARRIS, S.; GIBBINS, N. **3store: efficient bulk RDF storage**. 2003. Disponível em: <http://ceur-ws.org/Vol-89/harris-et-al.pdf>. Acesso em 14 jun. 2016.

HARRIS, S.; SEABORNE, A.; PRUD'HOMMEAUX, E. **SPARQL 1.1 query language**. 2013. Disponível em: <http://www.w3.org/TR/sparql11-overview/>. Acesso em: 14 jun. 2016.

HOBBS, J. R.; PAN, F. **Time ontology in OWL**. 2006. Disponível em: <https://www.w3.org/TR/2006/WD-owl-time-20060927/>. Acesso em: 29 maio 2016.

INTERNATIONAL STANDARD**. ISO 17369**: Statistical data and metadata exchange (SDMX). Geneva: ISO, 2005.

KÄMPGEN, B.; HARTH, A. Transforming statistical linked data for use in OLAP systems. In: INTERNATIONAL CONFERENCE ON SEMANTIC SYSTEMS, 7., 2011, Graz. **Proceedings**... New York: ACM, 2011. p. 33-40.

_____. OLAP4LD–A framework for building analysis applications over governmental statistics. In: PRESUTTI, V. et al. (Ed.). **The Semantic Web**: ESWC 2014 Satellite Events. Berlin: Springer, 2014. p. 389-394. (Lecture Notes in Computer Science, 8798).

KHODABAKHSH, M.; KHATAMI, M. Performing the SPARQL Query by Means of Mapreduce. **International Journal of Advanced Research in Computer Science and Software Engineering**, Uttar Pradesh, v. 5, n. 5, p. 66-75, 2015.

KIMBALL, R.; ROSS, M. **The data warehouse toolkit**: the complete guide to dimensional modeling. 2. ed. New York: Wiley, 2002.

MENDONÇA, R. R. de; CRUZ, S. M. S. da; CAMPOS, M. L. M. Gerência de Proveniência Multigranular em Linked Data com a Abordagem ETL4LinkedProv. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADDOS, 29., 2014, Curitiba. **Anais**... [S.l.: s.n.], 2014. p. 87-96.

NEU, A. R. **Distributed Evaluation of SPARQL queries with Impala and MapReduce**. 2014. 131 f. Tese (Doutorado)-Universidade de Friburgo, Friburgo, 2014.

ORACLE. **OLAP Application Developer's Guide**. 2003. Disponível em: < https://docs.oracle.com/cd/B19306_01/olap.102/b14349/sql_queries.htm#CHDBDJCF >. Acesso em: 12 maio 2016.

PALAZZI, D.; TYGEL, A. **Visualização de Dados Estatísticos Representados como Dados Abertos Ligados**. 2014. Disponível em: <http://cirandas.net/alantygel/academico/relatorio-visualizacao-de-dados-estatisticos-representados-como-dados-abertos-ligados.pdf>. Acesso em: 20 jun. 2016.

PESCE, M. L. **RdXel**: um conjunto de ferramentas para manipulação de dados estatísticos em rdf por meio de planilhas. 2012. 81 f. Dissertação (Mestrado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2012.

PRUD'HOMMEAUX, E.; BERTAILS, A. **A Mapping of SPARQL Onto Conventional SQL**. 2008. Disponível em: <https://www.w3.org/2008/07/MappingRules/SpringerTemplate/rdb2rdf.pdf>. Acesso em: 30 maio 2016.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL query language for RDF**. 2008. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. Acesso em: 25 jun. 2016.

RAO, S. **Monitoring and Measurement in the Next Generation Technologies.** 2010. Disponível em: <http://cordis.europa.eu/pub/fp7/ict/docs/future-networks/projects-moment-factsheet_en.pdf>. Acesso em: 13 jun. 2016.

SALAS, P. E. R. **OLAP2Datacube**: an On-Demand transformation Framework from OLAP to RDF Data Cubes. 2015. 95 f. Tese (Doutorado em Informática)- Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2015.

SALVADOR, A. et al. A Semantically Distributed Approach to Map IP Traffic Measurements to a Standardized Ontology. **International Journal of Computer Networks & Communications**, Chennai, v. 2, n. 1, p. 13-31, 2010.

SCHÄTZLE, A.; PRZYJACIEL-ZABLOCKI, M.; LAUSEN, G. PigSPARQL: mapping SPARQL to pig latin. In: INTERNATIONAL WORKSHOP ON SEMANTIC WEB INFORMATION MANAGEMENT,3., 2011, Athens. **Proceedings**... New York: ACM, 2011.

SCHMIDT, M. et al. SP2Bench: a SPARQL performance benchmark. In: IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 25., 2009, Shanghai. **Proceedings**... New York: IEEE, 2009. p. 222-233.

SEABORNE, A. **The SPARQL Algebra**. 2007. Disponível em: <https://www.w3.org/2001/sw/DataAccess/rq23/rq24-algebra.html>. Acesso em: 12 jun. 2016.

_____. **ARQ-A SPARQL Processor for Jena**. 2010. Disponível em: <http://jena.sourceforge.net/ARQ>. Acesso em: 12 jun. 2016.

SEQUEDA, J. F., MIRANKER, D. P. Ultrawrap: SPARQL execution on relational data. **Journal of Web Semantics**, v. 22, 2013.

SMITH, M. K.; WELTY, C.; MCGUINNESS, D. L. **OWL web ontology language guide**. 2004. Disponível em: <https://www.w3.org/TR/owl-guide/>. Acesso em: 26 jun. 2016.

SOUZA, R. et al. Linked open data publication strategies: application in networking performance measurement data. In: ASE INTERNATIONAL CONFERENCE ON BIG DATA SCIENCE AND COMPUTING,2., 2014, Stanford. **Proceedings**... [S.l.: s.n.], 2014.

STEWART, R. J.; TRINDER, P. W.; LOIDL, H.-W. Comparing high level mapreduce query languages. In: INTERNATIONAL WORKSHOP ON ADVANCED PARALLEL PROCESSING TECHNOLOGIES, 9., 2011, Shanghai. **Proceedings**... Berlin: Springer, 2011. p. 58-72.

THOMSEN, E. **OLAP solutions**: building multidimensional information systems. New York: Wiley, 1997.

VATANT, B.; WICK, M. **Geonames ontology**. 2012. Disponível em: <http://www.geonames.org/ontology/documentation.html>. Acesso em: 16 jun. 2016

WHITE, T. **Hadoop**: the definitive guide. Massachusetts: O'Reilly, 2012.

WILKINSON, K. **Jena property table implementation**. Palo Alto: Hewlett-Packard, 2006. (HPL-2006-140).

YAHOO DEVELOPER NETWORK. **Module 4**: MapReduce. 2016. Disponível em: <https://developer.yahoo.com/hadoop/tutorial/module4.html>. Acesso em: 13 jun. 2016.

ZIKOPOULOS, P.; EATON, C. **Understanding big data**: analytics for enterprise class hadoop and streaming data. New York: McGraw-Hill, 2011.

Apêndice A - Mapeamento dos prefixos utilizados na ontologia do domínio do Projeto PingER

```
PREFIX PingER: <http://www-iepm.slac.stanford.edu/pinger/lod/resource#>
PREFIX PingER-ont: <http://www-
iepm.slac.stanford.edu/pinger/lod/ontology/pinger.owl#>
PREFIX MGC: <http://www-
iepm.slac.stanford.edu/pinger/lod/ontology/MomentGeneralConcepts.owl#>
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX gn-ont: <http://www.geonames.org/ontology#>
PREFIX MU: <http://www.fp7-moment.eu/MomentUnits.owl#>
PREFIX MD: <http://www.fp7-moment.eu/MomentDataV2.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sdmx-attribute: <http://purl.org/linked-data/sdmx/2009/attribute#>
```

Apêndice B - Explicação das métricas de rede utilizados no estudo de caso do Projeto PingER

Packet Loss: Taxa de perda de pacotes de rede entre pares de nós. Mais detalhes em <http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html#loss>.

Unreachability: Taxa de inacessibilidade de um nó. Mais detalhes em <http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html#reachability>.

Unpredictability: Indicador de percentagem da imprevisibilidade do desempenho de um *ping*. Mais detalhes em <http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html#unpredict>.

Maximum Round Trip Time (RTT): Valor máximo de uma amostra de tempos totais de viagem ida-e-volta medidos num dado período de tempo.

Mean Opinion Score (MOS): Métrica que mede a qualidade de voz. Os valores do MOS são: 1 = mau; 2 = pobre; 3 = justo; 4 = bom; 5 = excelente. A faixa típica de qualidade de voz sobre conexões TCP/IP é de 3,5 a 4,2. Mais detalhes em <http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html#mos>.

Apêndice C - Conjunto de consultas SQL elaborado para testar os dados do PingER

```
SELECT PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode,
MAX(PingER_ont_hasValue) as maximo
FROM qb_Observation_Snappy
WHERE qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>'
GROUP BY PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode
ORDER BY maximo DESC
LIMIT 10
```

Consulta Q1

```
SELECT PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode,
MIN(PingER_ont_hasValue) as minimo
FROM qb_Observation_Snappy
WHERE qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMOSMeasurement>'
GROUP BY PingER_ont_hasSourceNode, PingER_ont_hasDestinationNode
ORDER BY minimo ASC
LIMIT 10
OFFSET 5
```

Consulta Q2

```
SELECT obs.qb_measureType,country.MGC_countryCode,
country.gn_ont_population, country.MGC_areaInSqKm,
AVG(obs.PingER_ont_hasValue)
FROM qb_Observation_Snappy obs join MGC_NodeInformation_Snappy node
      on obs.PingER_ont_hasDestinationNode = node.subject
join MGC_Town_Snappy town
      on node.MGC_isInTown = town.subject
join MGC_Country_Snappy country
      on town.MGC_isInCountry = country.subject
WHERE obs.qb_measureType = '<http://www
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>'
and obs.PingER_ont_hasSourceNode = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Node-pinger.slac.stanford.edu>'
GROUP BY obs.qb_measureType, country.MGC_countryCode,
country.gn_ont_population, country.MGC_areaInSqKm
UNION
SELECT obs.qb_measureType, country.MGC_countryCode,
country.gn_ont_population, country.MGC_areaInSqKm,
AVG(obs.PingER_ont_hasValue)
FROM qb_Observation_Snappy obs join MGC_NodeInformation_Snappy node
      on obs.PingER_ont_hasDestinationNode = node.subject
join MGC_Town_Snappy town
      on node.MGC_isInTown = town.subject
join MGC_Country_Snappy country
      on town.MGC_isInCountry = country.subject
WHERE obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricUnreachability>'
and obs.PingER_ont_hasSourceNode = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Node-pinger.slac.stanford.edu>'
GROUP BY obs.qb_measureType, country.MGC_countryCode,
country.gn_ont_population, country.MGC_areaInSqKm
```

Consulta Q3

```
SELECT tempo.time_year, tempo.time_month, AVG(obs.PingER_ont_hasValue)
FROM qb_Observation_Snappy obs join PingER_ont_DateTime_Snappy tempo
      on obs.PingER_ont_DateTime = tempo.subject
join MGC_NodeInformation_Snappy source
      on obs.PingER_ont_hasSourceNode = source.subject
join MGC_Town_Snappy townSource
      on source.MGC_isInTown = townSource.subject
join MGC_NodeInformation_Snappy dest
      on obs.PingER_ont_hasDestinationNode = dest.subject
join MGC_Town_Snappy townDest
      on dest.MGC_isInTown = townDest.subject
WHERE obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>'
and tempo.time_month BETWEEN 1 AND 4
and townSource.MGC_isInCountry = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country6252001>'
and townDest.MGC_isInCountry = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country1168579>'
GROUP BY tempo.time_year,tempo.time_month
HAVING (AVG(obs.PingER_ont_hasValue) > 0.75)
ORDER BY tempo.time_year, tempo.time_month
```

Consulta Q4

```
SELECT tempo.time_year, tempo.time_month, obs.qb_measureType,
AVG(obs.PingER_ont_hasValue)
FROM qb_Observation_Snappy obs join PingER_ont_DateTime_Snappy tempo
      on obs.PingER_ont_DateTime = tempo.subject
join MGC_NodeInformation_Snappy source
      on obs.PingER_ont_hasSourceNode = source.subject
join MGC_Town_Snappy townSource
      on source.MGC_isInTown = townSource.subject
join MGC_NodeInformation_Snappy dest
      on obs.PingER_ont_hasDestinationNode = dest.subject
join MGC_Town_Snappy townDest
      on dest.MGC_isInTown = townDest.subject
WHERE (obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricUnreachability>' ||
obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>' ||
obs.qb_measureType =  '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMaximumRoundTripDelayMeasu
rement>')
and tempo.time_month BETWEEN 1 AND 4
and townSource.MGC_isInCountry = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country6252001>'
and townDest.MGC_isInCountry = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country1168579>'
GROUP BY tempo.time_year, tempo.time_month, obs.qb_measureType
HAVING (AVG(obs.PingER_ont_hasValue) > 0.75)
ORDER BY tempo.time_year, tempo.time_month, obs.qb_measureType
```

Consulta Q5

```
SELECT obs.qb_measureType, cont.gn_ont_name, AVG(obs.PingER_ont_hasValue)
as media
FROM qb_Observation_Snappy obs join MGC_NodeInformation_Snappy dest
      on obs.PingER_ont_hasDestinationNode = dest.subject
join MGC_Town_Snappy townDest
      on dest.MGC_isInTown = townDest.subject
join MGC_Continent_Snappy cont
      on townDest.MGC_isInContinent = cont.subject
WHERE (obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMOSMeasurement>' ||
obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMaximumRoundTripDelayMeasu
rement>' || obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPingUnpredictabilityMeasur
ement>')
GROUP BY obs.qb_measureType, cont.gn_ont_name
ORDER BY obs.qb_measureType, cont.gn_ont_name
```

Consulta Q6

```
SELECT school.MGC_SchoolType, AVG(obs.PingER_ont_hasValue) as media
FROM qb_Observation_Snappy obs join MGC_NodeInformation_Snappy dest
      on obs.PingER_ont_hasDestinationNode = dest.subject
join MGC_School_Snappy school
      on dest.PingER_ont_hasNodeName = school.MGC_SchoolPingerName
WHERE obs.qb_measureType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricUnreachability>'
AND (school.MGC_SchoolType = 'Public university' || school.MGC_SchoolType =
'Private university')
GROUP BY school.MGC_SchoolType
ORDER BY media DESC
```

Consulta Q7

```
SELECT obs.qb_measureType, school.MGC_SchoolPingerName,
schoolOptional.MGC_SchoolNumberOfUgradStudents, school.MGC_SchoolEndowment,
AVG(obs.PingER_ont_hasValue) as media
FROM qb_Observation_Snappy obs join MGC_NodeInformation_Snappy dest
      on obs.PingER_ont_hasDestinationNode = dest.subject
join MGC_School_Snappy school
      on dest.PingER_ont_hasNodeName = school.MGC_SchoolPingerName
left join MGC_School_Snappy schoolOptional
      on school.subject = schoolOptional.subject
WHERE school.MGC_SchoolEndowment is not null
GROUP BY obs.qb_measureType, school.MGC_SchoolPingerName,
schoolOptional.MGC_SchoolNumberOfUgradStudents, school.MGC_SchoolEndowment
ORDER BY obs.qb_measureType, school.MGC_SchoolEndowment
```

Consulta Q8

```
SELECT tempo.time_year, tempo.time_month, obs.qb_measureType,
max(obs.PingER_ont_hasValue) as maximo, min(obs.PingER_ont_hasValue) as
minimo
FROM qb_Observation_Snappy obs join PingER_ont_DateTime_Snappy tempo
      on obs.PingER_ont_DateTime = tempo.subject
GROUP BY tempo.time_year, tempo.time_month, obs.qb_measureType
ORDER BY tempo.time_year, tempo.time_month
```

Consulta Q9

```
SELECT contSource.gn_ont_name,
sum(distinct(countrySource.gn_ont_population)) as soma,
avg(obs.PingER_ont_hasValue) as media
FROM qb_Observation_Snappy obs join MGC_NodeInformation_Snappy source
      on obs.PingER_ont_hasSourceNode = source.subject
join MGC_Town_Snappy townSource
      on source.MGC_isInTown = townSource.subject
join MGC_Country_Snappy countrySource
      on townSource.MGC_isInCountry = countrySource.subject
join MGC_Continent_Snappy contSource
      on townSource.MGC_isInContinent = contSource.subject
WHERE obs.qb_measureType =   '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>'
and countrySource.gn_ont_population is not null
GROUP BY contSource.gn_ont_name
ORDER BY media DESC
```

Consulta Q10

Apêndice D - Conjunto de consultas SPARQL elaborado para testar os dados do PingER

```
SELECT ?source ?destination (max(xsd:double(?valuepl)) as ?max) WHERE{
?s qb:measureType <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>.
?s PingER-ont:hasValue ?valuepl .
?s PingER-ont:hasSourceNode ?source .
?s PingER-ont:hasDestinationNode ?destination .
}
GROUP BY ?source ?destination
ORDER BY DESC(?max)
LIMIT 10
```

Consulta Q1

```
SELECT ?source ?destination (min(xsd:double(?valuepl)) as ?min) WHERE{
?s qb:measureType <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMOSMeasurement>.
?s PingER-ont:hasValue ?valuepl .
?s PingER-ont:hasSourceNode ?source .
?s PingER-ont:hasDestinationNode ?destination .
}
GROUP BY ?source ?destination
ORDER BY ASC(?min)
LIMIT 10
OFFSET 5
```

Consulta Q2

```
SELECT ?MetricType ?destinationcc ?pop ?area (avg(xsd:double(?value)) as
?Average) WHERE {
      {
      ?s PingER-ont:hasSourceNode <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Node-pinger.slac.stanford.edu> .
      ?s qb:measureType ?MetricType .
      ?s PingER-ont:hasValue ?value .
      ?s PingER-ont:hasDestinationNode ?destination .
      ?destination MGC:isInTown ?destinationt .
      ?destinationt MGC:isInCountry ?destinationc.
      ?destinationc MGC:countryCode ?destinationcc.
      ?destinationc gn-ont:population ?pop .
      ?destinationc MGC:areaInSqKm ?area .
      FILTER(?MetricType = "<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>")
      }
      UNION
      {
      ?s PingER-ont:hasSourceNode <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Node-pinger.slac.stanford.edu> .
      ?s qb:measureType ?MetricType .
      ?s PingER-ont:hasValue ?value .
      ?s PingER-ont:hasDestinationNode ?destination .
      ?destination MGC:isInTown ?destinationt .
      ?destinationt MGC:isInCountry ?destinationc.
      ?destinationc MGC:countryCode ?destinationcc.
      ?destinationc gn-ont:population ?pop .
      ?destinationc MGC:areaInSqKm ?area .
      FILTER(?MetricType = "<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricUnreachability>")
      }
}
GROUP BY ?MetricType ?destinationcc ?pop ?area
```

Consulta Q3

```
SELECT ?year ?month (avg(xsd:double(?value)) AS ?avg) WHERE{
?s PingER-ont:hasValue ?value .
?s PingER-ont:DateTime ?tempo .
?s qb:measureType <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss> .
?s PingER-ont:hasSourceNode ?source .
?s PingER-ont:hasDestinationNode ?dest .
?source MGC:isInTown ?sourcet .
?sourcet MGC:isInCountry <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country6252001>.
?dest MGC:isInTown ?destt .
?destt MGC:isInCountry <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country1168579> .
?tempo time:month ?month .
?tempo time:year ?year .
FILTER(xsd:integer(?month) >= "1"^^xsd:integer && xsd:integer(?month) <=
"4"^^xsd:integer) .
}
GROUP BY ?year ?month
HAVING(avg(xsd:double(?value)) > "0.75"^^xsd:double)
ORDER BY ?year ?month
```

Consulta Q4

```
SELECT ?year ?month ?metric (avg(xsd:double(?value)) AS ?avg) WHERE{
?s PingER-ont:hasValue ?value .
?s PingER-ont:DateTime ?tempo .
?s qb:measureType ?metric .
?s PingER-ont:hasSourceNode ?source .
?s PingER-ont:hasDestinationNode ?dest .
?source MGC:isInTown ?sourcet .
?sourcet MGC:isInCountry <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country6252001>.
?dest MGC:isInTown ?destt .
?destt MGC:isInCountry <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#Country1168579> .
?tempo time:month ?month .
?tempo time:year ?year .
FILTER(xsd:integer(?month) >= "1"^^xsd:integer && xsd:integer(?month) <=
"4"^^xsd:integer && (?metric = <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricUnreachability> || ?metric
= <http://www-iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>
|| ?metric = <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMaximumRoundTripDelayMeasu
rement>)) .
}
GROUP BY ?year ?month ?metric
HAVING(avg(xsd:double(?value)) > "0.75"^^xsd:double)
ORDER BY ?year ?month ?metric
```

Consulta Q5

```
SELECT ?metric ?destcname (avg(xsd:double(?value)) as ?avg) WHERE{
?s qb:measureType ?metric .
?s PingER-ont:hasValue ?value .
?s PingER-ont:hasDestinationNode ?dest .
?dest MGC:isInTown ?destt .
?destt MGC:isInContinent ?destc .
?destc gn-ont:name ?destcname .
FILTER(?metric = <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMOSMeasurement> || ?metric
= <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricMaximumRoundTripDelayMeasu
rement> || ?metric = <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPingUnpredictabilityMeasur
ement>) .
}
GROUP BY ?metric ?destcname
ORDER BY ?metric ?destcname
```

Consulta Q6

```
SELECT ?type (avg(xsd:double(?value)) as ?avg) WHERE{
?s qb:measureType <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricUnreachability>.
?s PingER-ont:hasValue ?value .
?s PingER-ont:hasDestinationNode ?dest .
?dest PingER-ont:hasNodeName ?nodename .
?school MGC:SchoolPingerName ?nodename .
?school MGC:SchoolType ?type .
FILTER(?type = 'Public university' || ?type = 'Private university') . }
GROUP BY ?type
ORDER BY DESC(?avg)
```

Consulta Q7

```
SELECT ?MetricType ?nodename ?ugradstudents ?endowment
(avg(xsd:double(?value)) as ?Average) WHERE{
?s qb:measureType ?MetricType .
?s PingER-ont:hasValue ?value .
?s PingER-ont:hasDestinationNode ?dest .
?dest PingER-ont:hasNodeName ?nodename .
?school MGC:SchoolPingerName ?nodename .
?school MGC:SchoolEndowment ?endowment .
FILTER(bound(?endowment)) .
OPTIONAL{?school MGC:SchoolNumberOfUgradStudents ?ugradstudents}
}
GROUP BY ?MetricType ?nodename ?ugradstudents ?endowment
ORDER BY ?MetricType ?endowment
```

Consulta Q8

```
SELECT ?year ?month ?MetricType (max(xsd:double(?value)) as ?maximo)
(min(xsd:double(?value)) as ?minimo) WHERE{
?s qb:measureType ?MetricType .
?s PingER-ont:hasValue ?value .
?s PingER-ont:DateTime ?tempo .
?tempo time:year ?year .
?tempo time:month ?month
}
GROUP BY ?year ?month ?MetricType
ORDER BY ?year ?month
```

Consulta Q9

```
SELECT ?contname (sum(distinct xsd:double(?population)) as ?sum)
(avg(xsd:double(?value)) as ?avg) WHERE{
?s PingER-ont:hasValue ?value .
?s qb:measureType <http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>.
?s PingER-ont:hasSourceNode ?source .
?source MGC:isInTown ?sourcet .
?sourcet MGC:isInCountry ?country .
?sourcet MGC:isInContinent ?continent .
?country gn-ont:population ?population.
?continent gn-ont:name ?contname .
FILTER(bound(?population))
}
GROUP BY ?contname
ORDER BY DESC(?avg)
```

Consulta Q10

Apêndice E - Consulta Q3 do conjunto de consultas SPARQL traduzida para SQL a partir do tradutor

```
SELECT AGREG.MetricType AS "MetricType" , AGREG.destinationcc AS
"destinationcc" , AGREG.pop AS "pop" , AGREG.area AS "area" , AGREG.Average
AS "Average"
 FROM (SELECT UNION1.MetricType AS "MetricType" , UNION1.destinationcc AS
"destinationcc" , UNION1.pop AS "pop" , UNION1.area AS "area" ,
avg(UNION1.value) AS "Average"
 FROM ((SELECT BGP1_1.destination AS "destination" , BGP1_1.destinationt AS
"destinationt" , BGP1_0.s AS "s" , BGP1_0.MetricType AS "MetricType" ,
BGP1_0.value AS "value" , BGP1_2.destinationc AS "destinationc" ,
BGP1_3.destinationcc AS "destinationcc" , BGP1_3.pop AS "pop" , BGP1_3.area
AS "area"
 FROM (SELECT subject AS "destination" , MGC_isInTown AS "destinationt"
 FROM MGC_NodeInformation_snappy) BGP1_1
JOIN (SELECT subject AS "s" , qb_measureType AS "MetricType" ,
PingER_ont_hasValue AS "value" , PingER_ont_hasDestinationNode AS
"destination" FROM qb_Observation_snappy WHERE PingER_ont_hasSourceNode =
'<http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-
pinger.slac.stanford.edu>') BGP1_0
ON(BGP1_1.destination=BGP1_0.destination)
JOIN (SELECT subject AS "destinationt" , MGC_isInCountry AS "destinationc"
 FROM MGC_Town_snappy) BGP1_2 ON(BGP1_1.destinationt=BGP1_2.destinationt)
JOIN (SELECT subject AS "destinationc" , MGC_countryCode AS "destinationcc"
, gn_ont_population AS "pop" , MGC_areaInSqKm AS "area"
 FROM MGC_Country_snappy) BGP1_3
ON(BGP1_2.destinationc=BGP1_3.destinationc)
 WHERE (BGP1_0.MetricType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricPacketLoss>'))
UNION
(SELECT BGP2_1.destination AS "destination" , BGP2_1.destinationt AS
"destinationt" , BGP2_0.s AS "s" , BGP2_0.MetricType AS "MetricType" ,
BGP2_0.value AS "value" , BGP2_2.destinationc AS "destinationc" ,
BGP2_3.destinationcc AS "destinationcc" , BGP2_3.pop AS "pop" , BGP2_3.area
AS "area"
 FROM (SELECT subject AS "destination" , MGC_isInTown AS "destinationt"
 FROM MGC_NodeInformation_snappy) BGP2_1
JOIN (SELECT subject AS "s" , qb_measureType AS "MetricType" ,
PingER_ont_hasValue AS "value" , PingER_ont_hasDestinationNode AS
"destination" FROM qb_Observation_snappy WHERE PingER_ont_hasSourceNode =
'<http://www-iepm.slac.stanford.edu/pinger/lod/resource#Node-
pinger.slac.stanford.edu>') BGP2_0
ON(BGP2_1.destination=BGP2_0.destination)
JOIN (SELECT subject AS "destinationt" , MGC_isInCountry AS "destinationc"
 FROM MGC_Town_snappy) BGP2_2 ON(BGP2_1.destinationt=BGP2_2.destinationt)
JOIN (SELECT subject AS "destinationc" , MGC_countryCode AS "destinationcc"
, gn_ont_population AS "pop" , MGC_areaInSqKm AS "area"
 FROM MGC_Country_snappy) BGP2_3
ON(BGP2_2.destinationc=BGP2_3.destinationc)
 WHERE (BGP2_0.MetricType = '<http://www-
iepm.slac.stanford.edu/pinger/lod/resource#MetricUnreachability>')) UNION1
 GROUP BY MetricType, destinationcc, pop, area) AGREG
```

Consulta Q3