7-12-2010

Efficient Storage of Semantic Web Data

Mihir S. Wagle
mwagle@purdue.edu

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Mihir Wagle

Entitled Efficient Storage of Semantic Web Data

For the degree of Master of Science

Is approved by the final examining committee:

Jeffrey Brewer
Chair

James Mohler

John Springer

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Jeffrey Brewer

Approved by: Gary Bertoline 7/9/10
Head of the Graduate Program Date

Graduate School Form 20
(Revised 1/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:
Efficient Storage of Semantic Web Data

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Teaching, Research, and Outreach Policy on Research Misconduct (VIII.3.1)*, October 1, 2008.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Mihir Wagle
Printed Name and Signature of Candidate

07/09/10
Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/viii_3_1.html

EFFICIENT STORAGE OF SEMANTIC WEB DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Mihir Wagle

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2010

Purdue University

West Lafayette, Indiana

To my parents for their continuous love and support.

ACKNOWLEDGMENTS

I feel fortunate to be a part of the Computer and Information Technology department at Purdue University. I am deeply grateful to my chair, Prof. Jeffrey Brewer for his invaluable guidance and support. I especially appreciate his kind and considerate nature and want to thank him for being so patient throughout my studies.

I would like to thank Prof. John Springer and Prof. James Mohler for their insightful comments and suggestions. I would also like to thank, Prof. Nathan Hartman and Micah Bojrab for providing me with access to their laboratory in order to perform the experiments.

TABLE OF CONTENTS

Page
LIST OF TABLES v
LIST OF FIGURES vi
ABSTRACT vii
CHAPTER 1. INTRODUCTION 1
1.1. Scope 1
1.2. Significance 2
1.3. Research Question 3
1.4. Assumptions 3
1.5. Limitations 4
1.6. Delimitations 4
1.7. Definitions 5
1.8. Summary 6
CHAPTER 2. LITERATURE REVIEW 7
2.1. Motivation and existing techniques 7
2.2. Summary 16
CHAPTER 3. FRAMEWORK AND EVALUATION 17
3.1. Framework 17
3.2. Evaluation 18
3.3. Summary 27
CHAPTER 4. DATA ANALYSIS 28
4.1. Graphical representation 28
4.2. Analysis and explanation 35
4.3. Summary 36
CHAPTER 5. CONCLUSIONS, DISCUSSIONS AND FUTURE DIRECTIONS 37
5.1. Conclusions 37
5.2. Discussion 38
5.3. Future Directions 39
5.4. Summary 39
LIST OF REFERENCES 40

LIST OF TABLES

Table Page
Table 3.1 Query response time for LUBM(1,0) 23
Table 3.2 Query response time for LUBM(5,0) 24
Table 3.3 Query response time for LUBM(10,0) 24
Table 3.4 Query response time for LUBM(20,0) 25
Table 3.5 Query response time for LUBM(50,0) 26
Table 3.6 Time taken to load the dataset 27

LIST OF FIGURES

Figure Page
Figure 4.1 Scatter plot for query 1 for LUBM(1,0) 28
Figure 4.2 Query response time for LUBM(1,0) 29
Figure 4.3 Query response time for LUBM(5,0) 30
Figure 4.4 Query response time for LUBM(10,0) 31
Figure 4.5 Query response time for LUBM(20,0) 32
Figure 4.6 Query response time for LUBM(50,0) 33
Figure 4.7 Time taken to load the dataset 34

ABSTRACT

Wagle, Mihir S. M.S., Purdue University, August, 2010. Efficient Storage of Semantic Web Data. Major Professor: Jeffrey Brewer.

With the adoption of RDF (Resource Description Framework), OWL (Web Ontology Language) and SPARQL (SPARQL Protocol And RDF Query Language) as standards for the semantic web, it has become essential to look into datawarehousing systems that are dedicated to working with the RDF data (World Wide Web Consortium). Traditional datawarehouses have focused on relational databases and have been optimized to work with the relational data. However, working with RDF data involves exploiting the triple nature of the data. As the size of the database increases, the time required to evaluate the queries on the database increases as well (Rohloff & Dean, et al., 2007). However, not only do the users need access to information as soon as possible, but also the information that is presented to them needs to be relevant to their search (Spink & Wolfram, et al., 2000). Through this project, the author looked into the different storage techniques for RDF data and attempted to strike a balance between the access time for information retrieval and parameters such as the storage space needed for the data and the complexity of the queries. BigOWLIM and Pellet which are built around open source frameworks such as Jena and Sesame

respectively were used for this study. The work done in this project is of significance mainly to small and medium enterprises since small datasets having about a million triples have been considered.

CHAPTER 1. INTRODUCTION

This chapter introduces the study with the scope, significance, research question and the definition of key terms. The assumptions, limitations and delimitations of the work are also stated thereafter.

1.1. Scope

The Resource Description Framework (RDF) schema is primarily used for storing and working with information on the World Wide Web. RDF is primarily made up of triples having a specific form (subject, object, predicate). The Web Ontology Language (OWL) provides a layer of abstraction and describes the relationships between these three RDF components. OWL enables one to query data from heterogeneous sources. The SPARQL Protocol and RDF Query Language (SPARQL) are an implementation of OWL and are similar to the Structured Query Language (SQL) that is used for relational databases. Efforts have been made to exploit the similarities between SQL and SPARQL while designing datawarehouses. In fact, the current implementations of many datawarehouses support both – SQL as well as SPARQL. This project looked into the different storage techniques for RDF data and attempted to strike a

balance between the access time for information retrieval and parameters such as storage space needed and the complexity of the queries. The focus was primarily on the RDF data and not on the relational data that one comes across in general datawarehouses.

1.2. Significance

Organizations have traditionally used relational databases to store data. In October 2009, the World Wide Web Consortium (W3C) accepted RDF and OWL as the standard for the storage of the World Wide Web data (World Wide Web Consortium). Since RDF and OWL have been accepted as the standard for the World Wide Web, it becomes important to look into systems that are dedicated to working with the RDF data. Although, the RDF format has been accepted as the format for the World Wide Web, essentially it could even be used for storing large amounts of data that is related to a particular corporation or enterprise (Konopnicki & Shmueli, et al., 2005).

The key fields on which the search terms are based in datawarehouses are usually indexed. Indices have been implemented in datawarehouses for faster information retrieval. However, storing these indices becomes an additional overhead. Prior work has focused on index compression techniques for datawarehouses in order to reduce the disk space (Ferragina & Gonzalez, et al., 2009). However, compressing and decompressing these indices in real-time can lead to a time delay in information retrieval. General purpose datawarehouses use horizontal partitioning in order to store the separate tuples

of information. Abadi and Marcus et al. (2009) attempted to partition the data on the basis of indexed columns. While vertical partitioning of the data speeds up the retrieval process, it is only applicable for a subset of RDF data that makes use of property tables. As the size of the database increases, the time required to evaluate the queries on the database increases as well (Rohloff & Dean, et al., 2007). However, users need answers to their queries as fast as possible and the time required for information retrieval is of prime importance to them (Spink & Wolfram, et al., 2000). Thus, there is a need to find a general approach that is applicable across the different types of datasets. Hence, it becomes important to determine and work on a trade-off between access time and storage space.

1.3. Research Question

What is the impact on the query response time of RDF data due to parameters such as the input size of the data and the complexity of the queries?

1.4. Assumptions

The following are the assumptions in the study:

1. The LUBM (Lehigh University BenchMark) dataset was used to generate the RDF data and was assumed to be a true representation of the homogeneous data in an RDF store (Guo & Pan, et al., 2004).
2. The system was assumed to be a standalone system (i.e., there did not exist multiple users querying the data store simultaneously).

1.5. Limitations

The following are the limitations of the study:

1. The focus was primarily on the RDF data and not on the relational data that one comes across in general data warehouses (i.e., the author did not take into account a general purpose database).
2. The author considered the RDF data and the Web Ontology Language for querying the RDF data as the standard for the World Wide Web. The author did not attempt to look into any alternate methods for the World Wide Web.
3. Although the author varied the storage space that was needed for the data, the focus of this study was essentially in terms of the complexity of the queries.
4. Stand-alone systems have been used for this project i.e., the systems do not take into account any network related problems.

1.6. Delimitations

The following are the delimitations of the study:

1. The author did not take into account datasets other than the LUBM dataset.
2. The editor, Eclipse was used for the system involving Pellet. Similarly the Sesame workbench was used for the system involving BigOWLIM. The author did not take into account the impact that these systems had on the test results of the study.

1.7. Definitions

- Resource Description Framework (RDF) – RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link. This is usually referred to as a triple having the form (subject, object, predicate). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications (Groppe & Ebers, et al., 2009).
- Web Ontology Language (OWL) – OWL builds on RDF and RDF Schema and adds more vocabulary for describing properties and classes. It incorporates features such as relations between classes (e.g., disjointness), cardinality (e.g., "exactly one"), equality, characteristics of properties (e.g. symmetry), and enumerated classes. It essentially describes the relationships between the three RDF components (Laborda & Conrad, 2005).
- SPARQL Protocol and RDF Query Language (SPARQL) – It is the query language that is primarily used for querying the RDF data. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns

along with their conjunctions and disjunctions. (Neumann & Weikum, 2008).

- Volume of a query: A low-volume query is one where the number of query results is very small (less than 5%) relative to the number of triples in the triple-store (Guo & Pan, et al., 2004). Conversely, a high-volume query is one that returns a large portion of the stored triples in response to a query.
- Complexity of a query: A low-complexity query is one that requires very little processing power to complete, while a high-complexity query is one that requires substantial computing power to complete.

1.8. Summary

This chapter provided an overview to the research work, including scope, significance, research question and definitions. The next chapter outlines the motivations for using RDF data. Also, it provides an overview of the current techniques used for storing the RDF data.

CHAPTER 2. LITERATURE REVIEW

This chapter talks about the work done by other researchers in this field. It provides the background for the work being done by the author.

2.1. Motivation and existing techniques

The study conducted by Spink, et al. in 2000 looked into the querying habits of users over the World Wide Web – their preferences and the search query terms entered by them. The study involved surveys of users using the Internet for finding information from popular search engines like Google, MSN, Yahoo, etc. The survey showed that people, especially those without a technical background rarely went beyond the top 10 results that the search engine provided. It clearly showed that the users were more concerned with getting the correct top few results rather than going through all the links that the search engines provided. This introductory paper, clearly demonstrating the user focus on precision over recall, showed the need to delve deeper into the field of information retrieval in order to get the top results correct without making the users wait for a long time to get to the information that they are looking for (Spink & Wolfram, et al., 2000). Their work clearly demonstrates the importance of rapidly getting the accurate results.

The study carried out by Dong and Halevy in 2007 looked into the basic data storage methods that are predominantly used for the World Wide Web. It delved deeper into the inverted list structure for the Semantic Web as well as extensions to it that could help in efficient retrieval of data. Dong and Halevy indexed heterogeneous data from multiple sources through a central (virtual) triple store, so as to support queries that combine keywords and structural specifications. The study talked about research methods that were designed to support flexible querying over databases. It showed that incorporating structure into inverted lists could considerably speed up query answering. It also, showed methods that not only allow the users to specify query structure when they can, but also allows them to fall back on keywords in the absence of a fixed framework, could potentially be of prime importance. Dong and Halevy proposed a hybrid index that combined the strengths of the following two approaches:

- Dup-ATIL: duplicating a row that includes an attribute name for each of its ancestors in the hierarchy
- Hier-ATIL: keyword in each row includes the entire hierarchy path

The main contribution of their study was that it underscored the importance of inverted lists, even if in the modified form, when it comes to efficient querying over heterogeneous data sources. The author of this study has built further upon this work and looked into the different ways for storage of data structures that support efficient querying over heterogeneous data sources.

Groppe and Ebers et al. (2009) looked into existing work for languages that are used to query for information over the World Wide Web. While there is

SQL (Structured Query Language) for structured databases, there was a need to identify querying languages for unstructured and semi-structured data. The author came across the RDF (Resource Description Framework) for working with unstructured data. RDF represents the basic support to write metadata on Web resources and to grant interoperability among heterogeneous applications when exchanging these metadata. The author then focused on the similarities and differences between SQL and SPARQL (SPARQL Protocol and RDF Query Language) when it comes to querying the RDF data. While there are many similarities between SQL and SPARQL, SPARQL has its own characteristics different from SQL, that could be exploited for optimizing the SPARQL queries. The approach of Groppe & Ebers, et al., of dynamically restricting the triples and working with indices can help to efficiently perform computations on the RDF data.

The use of RDF can be in a controlled environment such as an enterprise or an uncontrolled environment such as the World Wide Web. The author then looked into existing research that talked about the use of search indices to aggregate data from all kinds of applications and servers (Konopnicki & Shmueli, et al., 2005). Their study suggested that it was important to integrate information from a variety of sources including but not limited to objects, documents, semantic information, XML and other text data. The study by Konopnicki & Shmueli focused on the requirements of a query language in order to harness this unstructured, heterogeneous data. This study demonstrated that RDF data

could be used for enterprises and did not necessarily have to be restricted to the World Wide Web.

Relational OWL (Web Ontology Language) provides a layer of abstraction for querying data from heterogeneous sources. Laborda and Conrad (2005) looked into the representation format for both, schema and data information based on the Web Ontology Language. Their aim was to enable seamless integration of databases from different formats that could provide for scalable processing of join operations over the heterogeneous data formats. The use of relational OWL enables us to write formal conceptualizations of domain models (i.e., the ontology). After creating an ontology, the researchers were able to encode knowledge about things and their inter-relationships within their specific domain into a machine-understandable format, which could later be decoded and interpreted. One of the primary advantages of using relational OWL is the simple interconnectivity of existing ontologies. Two communities using different ontologies could easily collaborate, as soon as a semantic mapping is created between these two ontologies. This has potential applications in the field of peer-to-peer databases. For applications where the recall value is not so important as compared to the precision value (e.g., searching over the World Wide Web), multiple, peer to peer databases could be used. This could drastically reduce the access time.

The author then focused on the general compression techniques used in databases. While structured data is different from RDF data, the underlying index compression techniques for data storage are essentially the same. Also, there

has been a lot of work in the field of inverted list storage techniques (Ferragina & Gonzalez, et al., 2009). Indices have been implemented in datawarehouses for faster information retrieval. However, storing these indices becomes an additional overhead. Prior work has focused on index compression techniques for data warehouses in order to reduce the disk space. The author looked into existing research in the field of index compression in the form of prevalent compression algorithms such as the suffix array, Lempel Ziv index and full-text compressed indices (Ferragina & Gonzalez, et al., 2009). The ratio of access time to storage space provides an insight into the efficacy of the different algorithms. However, the author observed that compressing and decompressing these indices in real-time can lead to a performance delay in information retrieval on account of the overhead associated with these tasks.

Web documents contain a lot of links to other documents. The Uniform Resource Indicators (URIs) cover a significant portion of the RDF documents. Storage space could be saved by making use of the relative paths of these documents. General purpose compressors such as gzip neither take into account the format of the RDF data nor the XML links that accompany the RDF data on the World Wide Web. XML compressors can provide very high compression rates. However, these compressors are not equipped with query processing capabilities. Lee and Kim et al. (2008) proposed a compression mechanism that consists of two levels based on the dictionary based encoding. The first level is to find an URI index of an URI reference to be compressed in the URI dictionary. The second level is to find an URI reference index and replace the URI reference

with URI reference index. The two level dictionary based encoding approach: one for compressing the URI parts of URI references and the other for compressing whole URI references looks to be quite promising. The work done by Lee and Kim et al. (2008) focused on achieving the maximum possible compression for RDF data by making use of the XML links that inherently accompany the data on the World Wide Web. However, there is still a significant amount of work remaining when it comes to compressing and de-compressing the data in real-time for faster information retrieval. Also, their study focuses on compressing the links in the XML data that essentially accompany the RDF data on the World Wide Web. It does not make any attempt to look into factors that could have a bearing on the RDF data itself.

Abadi and Marcus et al. (2009) explored the scalability issues with respect to current data management solutions for RDF data. They primarily focused on two approaches in order to store the RDF data: a) the use of property tables and b) vertical partitioning of the RDF data. The property table technique denormalizes RDF tables by physically storing them in a wider, flattened representation similar to traditional relational schemas. Flattening the data involves finding sets of properties that tend to be defined together. The flattened property table representation requires fewer joins to access, because self joins on the subject column are eliminated. However, there are several limitations of this approach:

- Nulls: Because few properties are defined for all subjects in the subject cluster, the resulting join tables have many null values.

- Multi-valued attributes: Attributes having multiple values and many-to-many relationships are difficult to express in a flattened representation.
- Proliferation of Union Clauses and joins: Most of the queries are not restricted to a single property table. Querying multiple flattened tables leads to complex union clauses and joins.

Abadi and Marcus et al. (2009) then proposed an alternative approach of vertically partitioning the RDF data. It involved creating a two column table for each unique property in the RDF dataset. The first column contained subjects that defined the property. The second column contained the object values for the subjects. In order to evaluate the performance of vertical partitioning, they executed queries generated by a Web-based RDF browser over a large scale catalog of library data. Further, it was observed that if a column-oriented DBMS (a database architected specially for the vertically partitioned case) was used instead of a row oriented DBMS, a significant performance improvement was observed, with querying time dropping from minutes to seconds. While vertical partitioning speeds up the retrieval process, it is only applicable for RDF data that makes use of property tables. The author of this project observed that there was a need to find a general approach that was applicable across the different types of datasets.

From the standpoint of a relational database, the constraints on scalability and efficiency are derived from the very nature of the RDF data model, which is based on a triple format. Weiss and Karras et al. (2008) studied the schemes that utilize the triple nature of RDF data by indexing the RDF data in six possible

ways, one for each possible ordering of the three RDF elements. They created a Hexastore with six indices because for the RDF triple of (subject, object, predicate), 3! = 6 different orderings are possible. Each index structure in the Hexastore was centered on one RDF element and defined a prioritization between the other two elements. This approach exploits the triple nature of RDF data. The vertical partitioning approach would appear as a special case of the Hexastore where the index would be centered on the subject or object. While this method overcomes the problem of accessing data without property tables, it also leads to an increase in the index storage space. The author of this project observed that a single update or insert operation would affect all six indices, thereby slowing down the performance. This project has attempted to determine and work on a trade-off between the access time and parameters such as storage space and complexity of queries.

Neumann and Weikum (2008) provided an implementation of RDF data storage that used the six index approach of the hexastore. They studied the existing solutions that store and index the RDF triples while completely eliminating the need for physical design tuning. Instead of making any changes to the physical design, they focused on scalable join processing. Additionally, Neumann and Weikum developed light weight methods for information passing between separate joins at query run-time. These provided a highly effective filter on the input streams of joins. Also, their work involved improving upon the previously proposed algorithms for join-order optimization by making accurate selectivity estimations for very large RDF graphs. The use of very fast merge

joins greatly improved the information retrieval time. However, their approach did not provide a complete SPARQL implementation.

Neumann and Weikum (2008) also developed the RDF-3X engine, an implementation of SPARQL that pursues a RISC-style architecture. RDF-3X provided a generic solution for storing and indexing RDF triples that completely eliminated the need for physical design tuning. It leveraged the work done by them with respect to the fast merge join operations. Also, RDF-3X made use of a query optimizer for choosing optimal join orders using a cost model based on statistical synopses for entire join paths. A selectivity estimator based on statistics for frequent paths acted as the input for the query optimizer. The author of this project proposes to evaluate and independently test the efficacy of RDF-3X on different datasets as a part of the further development of this work.

Guo and Pan et al. (2004) developed the LUBM (Lehigh University Benchmark) in order to benchmark different OWL Knowledge Base Systems. The LUBM featured an ontology for the university domain and synthetic OWL data scalable to an arbitrary size. The benchmark helps to evaluate knowledge base systems with respect to extensional queries over a large dataset that commits to a single realistic ontology. Based on the benchmark, their work was essentially focused on the scalability of systems working with RDF data. On the other hand, this project has attempted to focus on the impact of change in query complexity on the different OWL Knowledge Base Systems.

Rohloff and Dean et al. (2007) compared the performance of different triple store technologies using the LUBM framework. Their work dealt with

different deployment scenarios where the triple store needs to load data and respond to queries over a very large knowledge base (on the order of hundreds of millions of triples). While their work focused on the scalability of the triple store systems, they used proprietary technologies such as AllegroGraph and Virtuoso for their study. Their work is useful for the large enterprises that have access to such high performance systems.

2.2. Summary

This chapter provided a brief overview of the motivations for the focus on RDF data for the World Wide Web and few data storage techniques. Though there have been widely proposed methods for the storage of RDF data in literature, none have found widespread use in any commercial applications due to various factors such as the access time for information retrieval and the scalability factor on account of the overhead on storage space. The next chapter focuses on the specific methodology and the framework developed for this study.

CHAPTER 3. FRAMEWORK AND EVALUATION

This chapter discusses the framework used for this project done by the author. It also talks about the experiments conducted in order to evaluate the performance of the system.

3.1. Framework

The author compared the following two systems:

a. Pellet (Clark & Parsia) which is based on top of the Jena framework (Sourceforge) and
b. BigOWLIM (BigOWLIM Corporation) which is based on top of the Sesame framework (Aduna Corporation).

Pellet does not provide for persistent storage and performs the computations in-memory. On the other hand BigOWLIM provides persistent storage and well as implements disk-based reasoning. Initially the author of this project started out with SwiftOWLIM (SwiftOWLIM Corporation) instead of BigOWLIM. The SwiftOWLIM system provides persistent storage just like the BigOWLIM system as they both make use of the Sesame framework. However, unlike BigOWLIM, SwiftOWLIM performs the computations in-memory. This makes it a system that is more comparable to Pellet. However, the author of this project observed that

SwiftOWLIM scaled very poorly and failed to execute even the simplest of the queries on the most basic dataset of LUBM(1,0). Since SwiftOWLIM was not scalable, the author of this project switched over to BigOWLIM instead.

The hardware specifications used for this project are as follows:

- 2.67 GHz Intel Core i7 – 920 Processor
- 12 GB RAM
- 8 MB Cache
- 1TB hard disk

The software specifications used for this project are as follows:

- Windows Vista Home Premium
- Java SDK 1.6 with Eclipse SDK 3.4
- Pellet 2.0.2 with Jena 2.6.2
- BigOWLIM 3.0 with Sesame 2.0

3.2. Evaluation

The author used the LUBM dataset and compared the query run-times of the two systems mentioned above. The LUBM dataset is the standard benchmark that has widely been adopted by major companies like Oracle to measure the performance of Knowledge Base Systems (Oracle Corporation, 2009). Data stores like MySQL and PostgreSQL have already been tried out as

alternatives and have found to be wanting – both in terms of performance as well as scalability (Rohloff & Dean, et al., 2007).

The author merged the relevant input files and combined them into a single input file in order to simplify the loading process. Also, the author tested the standard LUBM queries for both the systems. The author grouped the queries into the following four classes:

- Class 1: Low volume, low complexity
- Class 2: Low volume, high complexity
- Class 3: High volume, low complexity
- Class 4: High volume, high complexity

The description of volume and complexity with respect to query types was taken from the LUBM documentation and has been briefly described in section 1.7 of chapter 1. The author ran a query of each of the four above mentioned sets as a representative for that type. Each query was executed fifty times and the response time was noted in order to mitigate statistical sampling errors. For the queries, the author included the geometric mean of the query set, because it was often used as the workload-average measure in benchmarks and was more resilient to extreme outliers than the arithmetic average (Neumann & Weikum, 2008). Also, the cache memory of the system was flushed every time in order to ensure that the results were not affected by the level of cache memory optimization. The queries have been described briefly as follows:

- LUBM Query 1 was used as a low volume, low complexity query.
- LUBM Query 2 was used as a low volume, high complexity query.

- LUBM Query 14 was used as a high volume, low complexity query.
- LUBM Query 9 was used as a high volume, high complexity query.

Class 1 - LUBM Query 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE
{?X rdf:type ub:GraduateStudent .
?X ub:takesCourse
<http://www.Department0.University0.edu/GraduateCourse0>}
```

This query asks for the number of graduate students at a particular university at a particular course.

Class 2 – LUBM Query 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X ?Y ?Z
WHERE
{?X rdf:type ub:GraduateStudent .
?Y rdf:type ub:University .
?Z rdf:type ub:Department .
?X ub:memberOf ?Z .
```

?Z ub:subOrganizationOf ?Y .

?X ub:undergraduateDegreeFrom ?Y}

This query is fairly complex and involves a triangular relationship between the GraduateStudent, the Department and the University.

Class 3 – LUBM Query 14:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>

SELECT ?X

WHERE {?X rdf:type ub:UndergraduateStudent}

This query simply lists out all the undergraduate students in the department. A correct response for this query is a large fraction of the number of triples stored in the triple-store.

Class 4 – LUBM Query 9:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>

SELECT ?X ?Y ?Z

WHERE

{?X rdf:type ub:Student .

?Y rdf:type ub:Faculty .

?Z rdf:type ub:Course .

?X ub:advisor ?Y .

?Y ub:teacherOf ?Z .

?X ub:takesCourse ?Z}

This query is fairly complex and involves a triangular relationship between the Student, the Faculty and the Course. Also, a correct response for this query is a large fraction of the number of triples stored in the triple-store. Although, this is a high volume query like query 14, the number of results returned is much smaller than that of query 14.

The individual metrics initially used by the LUBM were used as a starting point for the data collection in this evaluation study. Data was collected on the following parameters:

- Number of files: The number of files that were merged in order to form the input file gave us this parameter.
- Input size: The size of the input file used to load the evaluation data was noted in order to provide an idea about the disk space requirements.
- Number of triples: Although the number of triples is usually proportional to the input size, this parameter provided an accurate measure of the size of the datawarehouse.
- Query response time: Query response time was calculated as the geometric mean of the execution time for each of the four classes of queries. Time was measured with the help of a stop-watch and not in terms of the number of CPU cycles involved.

The input size of the data was varied along the following lines in order to test the performance and scalability of the two systems:

- LUBM(1,0)
- LUBM(5,0)
- LUBM(10,0)
- LUBM(20,0)
- LUBM(50,0)

Table 3.1.

Query response time for LUBM(1,0)

	Geometric Mean query response time for Pellet-Jena system	Arithmetic Mean query response time for Pellet-Jena system	Geometric Mean query response time for BigOWLIM-Sesame system	Arithmetic Mean query response time for BigOWLIM-Sesame system
Class 1	3183 ms	3259 ms	3046 ms	3254 ms
Class 2	3338 ms	3583 ms	3229 ms	3371 ms
Class 3	3513 ms	3842 ms	3443 ms	3552 ms
Class 4	3415 ms	3672 ms	3338 ms	3501 ms

Table 3.2.

Query response time for LUBM(5,0)

	Geometric Mean query response time for Pellet-Jena system	Arithmetic Mean query response time for Pellet-Jena system	Geometric Mean query response time for BigOWLIM-Sesame system	Arithmetic Mean query response time for BigOWLIM-Sesame system
Class 1	16724 ms	17102 ms	16538 ms	16816 ms
Class 2	17321 ms	17619 ms	17119 ms	17454 ms
Class 3	18898 ms	18999 ms	18795 ms	18904 ms
Class 4	19839 ms	20012 ms	19753 ms	19954 ms

Table 3.3.

Query response time for LUBM(10,0)

	Geometric Mean query response time for Pellet-Jena system	Arithmetic Mean query response time for Pellet-Jena system	Geometric Mean query response time for BigOWLIM-Sesame system	Arithmetic Mean query response time for BigOWLIM-Sesame system
Class 1	37378 ms	38153 ms	37032 ms	37398 ms
Class 2	39307 ms	40702 ms	39017 ms	39423 ms

Table 3.3. continued

Query response time for LUBM(10,0)

Class 3	40358 ms	40721 ms	39998 ms	40543 ms
Class 4	42576 ms	42884 ms	42254 ms	42657 ms

Table 3.4.

Query response time for LUBM(20,0)

	Geometric Mean query response time for Pellet-Jena system	Arithmetic Mean query response time for Pellet-Jena system	Geometric Mean query response time for BigOWLIM-Sesame system	Arithmetic Mean query response time for BigOWLIM-Sesame system
Class 1	-	-	74153 ms	74578 ms
Class 2	-	-	78157 ms	78724 ms
Class 3	-	-	80107 ms	80601 ms
Class 4	-	-	84575 ms	84903 ms

Table 3.5.

Query response time for LUBM(50,0)

	Geometric Mean query response time for Pellet-Jena system	Arithmetic Mean query response time for Pellet-Jena system	Geometric Mean query response time for BigOWLIM-Sesame system	Arithmetic Mean query response time for BigOWLIM-Sesame system
Class 1	-	-	185002 ms	185563 ms
Class 2	-	-	194973 ms	195347 ms
Class 3	-	-	199956 ms	200397 ms
Class 4	-	-	210913 ms	211463 ms

Although loading the dataset is a one-time operation for most enterprises, periodic back-ups need to be performed in order to maintain the consistency of the data. Thus, it is important to have an estimate of the time taken to load the data. Hence, the author also noted the time taken to load the data into the two systems respectively.

Table 3.6.

Time taken to load the dataset

	Number of files	Input size	Number of triples	Load time (Pellet-Jena)	Load time (BigOWLIM-Sesame)
LUBM(1,0)	15	7.82 MB	103074	3641 ms	3518 ms
LUBM(5,0)	93	49 MB	645649	13234 ms	12576 ms
LUBM(10,0)	189	99.9 MB	1316322	24107 ms	22185 ms
LUBM(20,0)	402	212 MB	2781322	46426 ms	41653 ms
LUBM(50,0)	999	529 MB	6888642	117328 ms	116987 ms

3.3. Summary

This chapter focused on the framework and the evaluation methodology developed for this study. The chapter also discussed the experimental setup and the process that was followed.

CHAPTER 4. DATA ANALYSIS

This chapter presents the analysis of data. It presents the findings for different metrics used to evaluate the efficiency of the semantic web systems.

4.1. Graphical representation

A scatter plot was drawn in order to check for the consistency of the data.

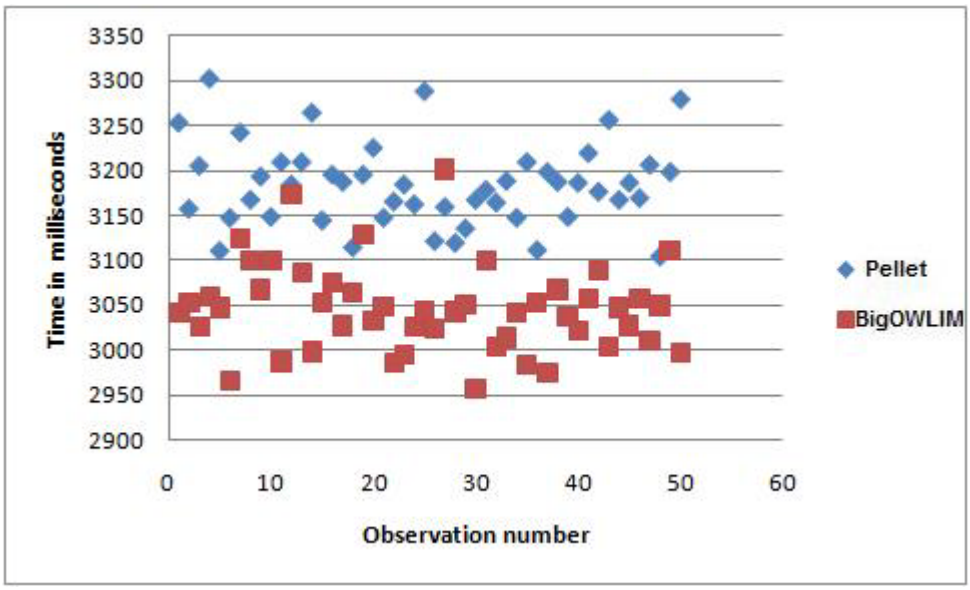

Figure 4.1 Scatter plot for query 1 for LUBM(1,0)

The above figure shows the query response time for the fifty data points for the two systems. The author observed that the data was randomly distributed. Although, the query response time for the BigOWLIM system is greater than the

query response time for the Pellet system for some of the observations, the geometric mean of the fifty data points for the BigOWLIM system is smaller than that of the Pellet system.

The graphical representation of the query response time against the class of queries is as follows:

The blue line shows the response time for the Pellet – Jena system. The red line stands for the BigOWLIM – Sesame system. The vertical axis shows the time in milliseconds. The horizontal axis gives the class of queries.

LUBM(1,0):

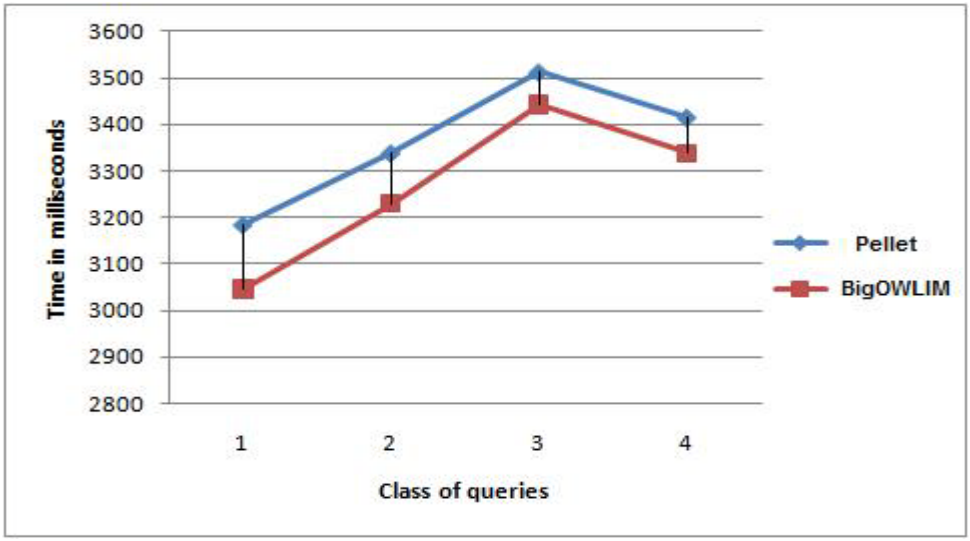

Figure 4.2 Query response time for LUBM(1,0)

One can clearly see that the Pellet - Jena system is slower than the BigOWLIM – Sesame system for all queries for LUBM(1,0). Also, one can see that LUBM query 14 gave results faster than LUBM query 9 for LUBM(1,0) for both the systems. This can be attributed to the fact that LUBM(1,0) has only 15 files and

about 0.1 million triples. Thus, a highly complex query like query 9 is executed much faster as compared to a high volume query like query 14.

LUBM(5,0):

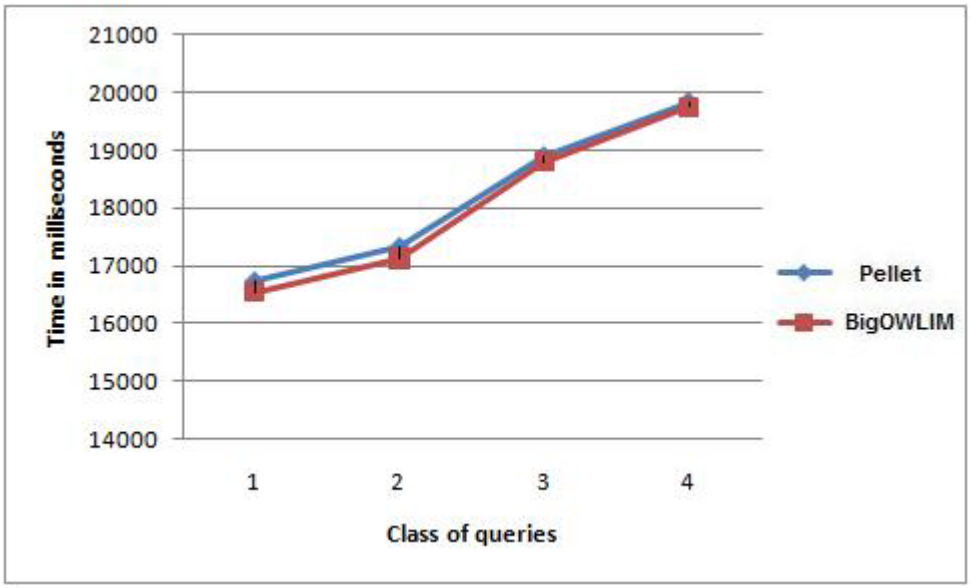

Figure 4.3 Query response time for LUBM(5,0)

For LUBM(5,0) the response time for both the systems is nearly the same. However, the Pellet – Jena system is still slightly slower than the BigOWLIM – Sesame system. Also, one observes that for LUBM(5,0) both the systems are able to execute a high volume query like query 14 faster than a highly complex query like query 9. LUBM(5,0) has 93 files and 0.6 million triples. Thus, one observes that as the size of the dataset increases, it takes more time to execute a highly complex query as compared to a high volume query.

LUBM(10,0):

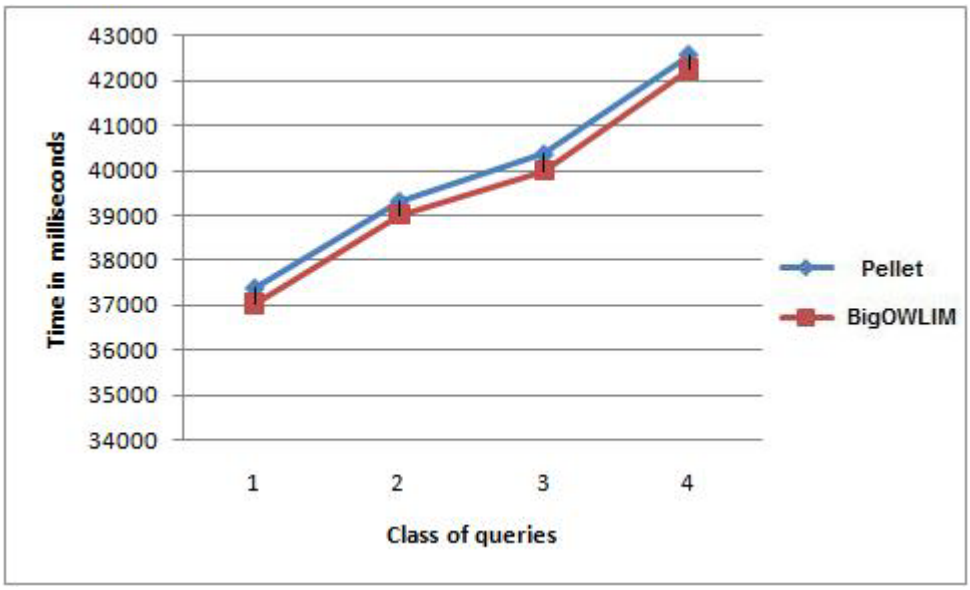

Figure 4.4 Query response time for LUBM(10,0)

For LUBM(10,0) the pattern of query response time is similar to that of LUBM(5,0). However, one observes that the time gap between these two systems has increased slightly as the size of the dataset has increased from about 0.6 million triples to about 1.3 million triples i.e., the BigOWLIM – Sesame system appears to have improved its performance as compared to the Pellet – Jena system as the dataset has scaled up in size.

LUBM(20,0):

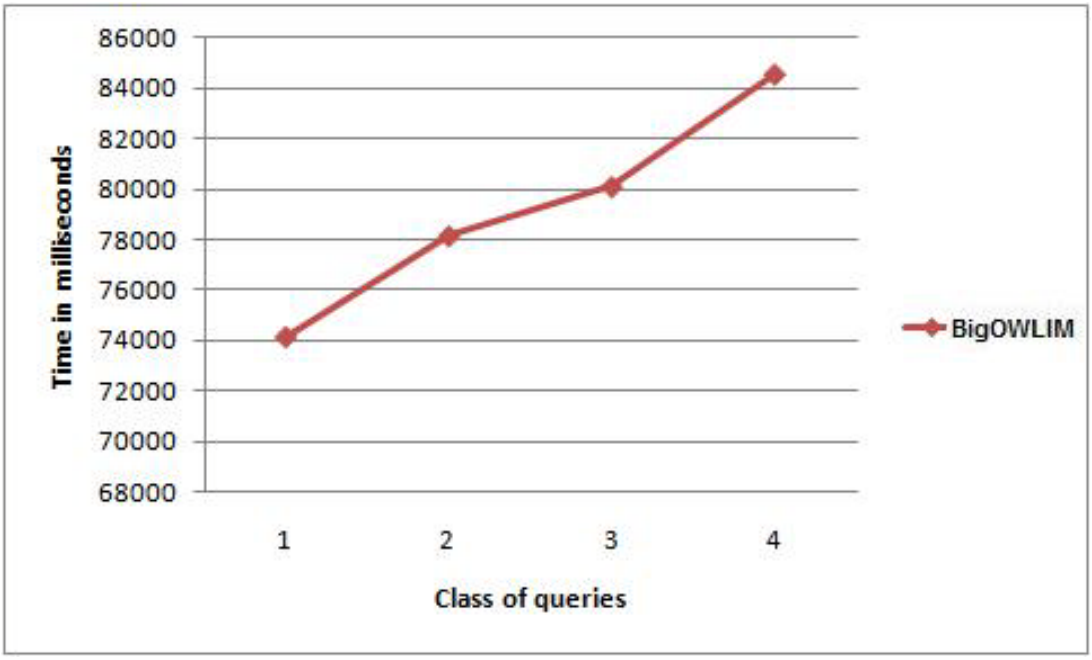

Figure 4.5 Query response time for LUBM(20,0)

Since the Pellet system failed to execute the queries for LUBM(20,0), the author progressively increased the input size of the database in order to find the exact input data size at which Pellet stops working for the given system configuration. The author observed that the Pellet system fails to execute queries beyond LUBM(17,0). LUBM(17,0) has 333 files with 2299693 triples and an input size of 180 MB. The author tried to increase the size of the JVM (Java Virtual Machine). However, the Pellet system still gave the error "java.lang.OutOfMemoryError: PermGen space". Thus, Pellet fails because it performs its computations in-memory.

LUBM(50,0):

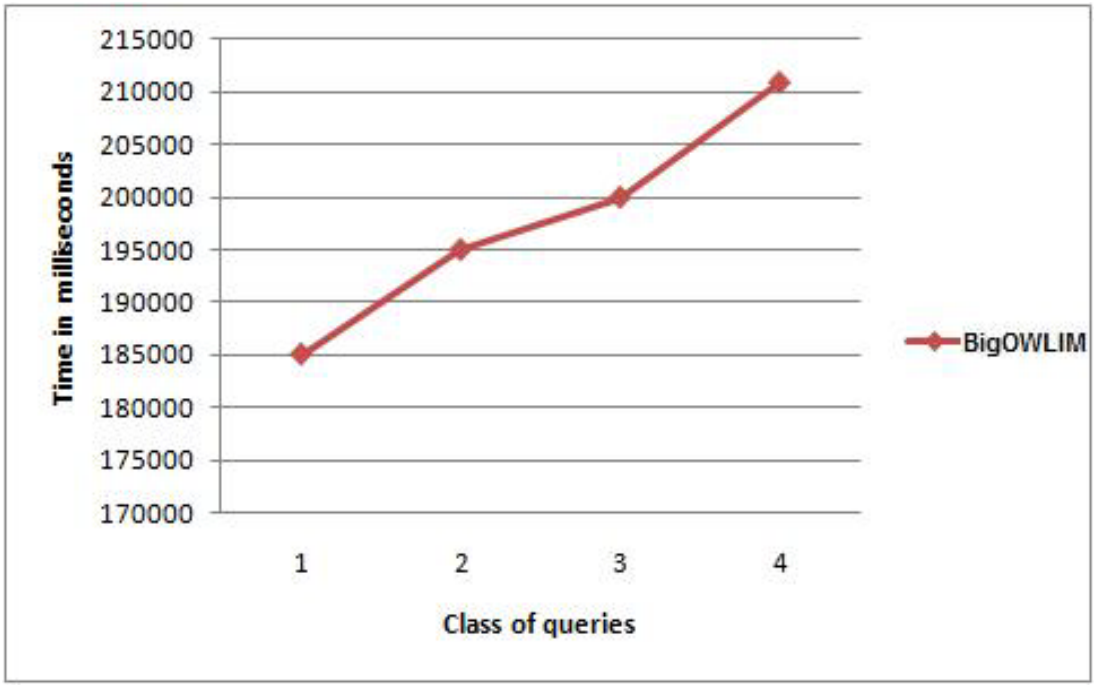

Figure 4.6 Query response time for LUBM(50,0)

The Pellet – Jena system failed to execute the queries for LUBM(20,0) and LUBM(50,0) despite increasing the memory allotted to the Java Virtual Machine (JVM) to about 12 GB. The Pellet – Jena system successfully managed to load the dataset. However, during query execution it failed to answer even the most basic class of queries (Class 1). For the BigOWLIM – Sesame system, the query response time patterns for both LUBM(20,0) and LUBM(50,0) were almost identical. Thus, one observes that unlike the Pellet – Jena system, the BigOWLIM – Sesame system is scalable. Also, one can see that the BigOWLIM – Sesame system is faster than the Pellet – Jena system.

Load time:

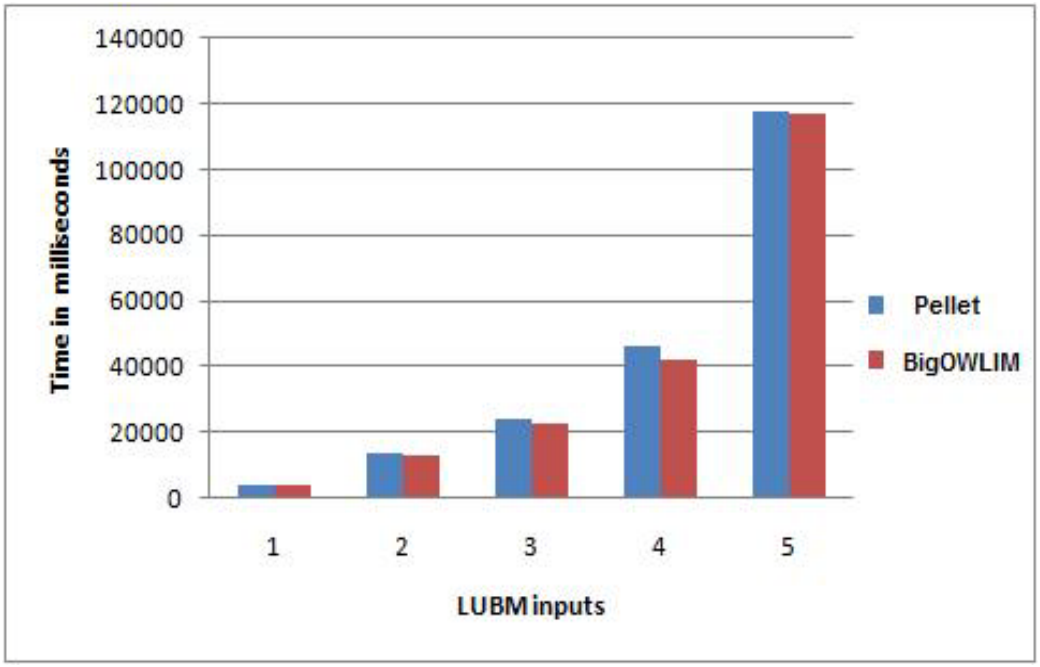

Figure 4.6 Time taken to load the dataset

At first glance, the time needed to load the datasets appears to increase exponentially. However, if one takes into account that the input data size, given on the horizontal axis, is also increasing, then one observes the growth is not exponential but rather close to linear. Also, one observes that the time taken to load the dataset is slightly greater for the Pellet – Jena system as compared to the BigOWLIM – Sesame system. This time difference as a percentage of the total time taken to load the dataset progressively decreases as one moves from LUBM(1,0) to LUBM(50,0).

4.2. Analysis and explanation

Based on the results, the author observes that the BigOWLIM – Sesame system is faster and more scalable as compared to the Pellet – Jena system. However, using the Pellet – Jena system too has its share of benefits. An additional, but significant observation is that the BigOWLIM – Sesame system accepts the input even if it is not formatted according to the specified RDF tags. Thus, one of the most significant advantages of the Pellet – Jena system is that it does a strong type checking of the input. The Pellet – Jena system throws a runtime exception in case the input is not in the correct format. This is a very significant advantage of the Pellet – Jena system particularly when it comes to working with large datasets.

There are a few reasons that could provide an explanation for the poor performance of the Pellet – Jena system as compared to the BigOWLIM – Sesame system:

- The Pellet – Jena system does an error checking of the input files. Hence, it needs some additional time to perform the validation as compared to the BigOWLIM – Sesame system.
- The Pellet – Jena system uses the tableau algorithm for evaluating the queries (Haarslev & Moller, 2001). On the other hand, the BigOWLIM – Sesame system uses the forward chaining algorithm for evaluating the queries (Bacchus & Winter, 2001). This probably explains why the Pellet – Jena system is not able to execute the queries for LUBM(20,0) and above although it manages to load the dataset.

4.3. Summary

This chapter presented the analysis of the data gathered in this research. The next chapter presents the conclusions and recommendations for future directions of the research.

CHAPTER 5. CONCLUSIONS, DISCUSSIONS AND FUTURE DIRECTIONS

This chapter summarizes the findings in this research. It further provides a general discussion and directions for further extension of this research.

5.1. Conclusions

The author evaluated the efficiency of two systems for the storage and retrieval of semantic web data – Pellet, which is based on top of the Jena framework and BigOWLIM, which is based on top of the Sesame framework.

The BigOWLIM – Sesame system is faster than the Pellet – Jena system. The performance of the BigOWLIM – Sesame system is better than that of the Pellet – Jena system across the different classes of queries for a given value of input data. The queries have been classified on the basis of their complexity as well as volume.

The author then varied the size of the input data. The performance of the BigOWLIM – Sesame system was better than that of the Pellet – Jena system for data of different input size. As the size of the input data increases, the author observed that the Pellet – Jena system failed to meet the requirement of scalability.

Although the BigOWLIM – Sesame system is faster than the Pellet – Jena system and meets the requirements of scalability as well, one of its significant drawbacks is that it does not perform a strong type checking of the input data. This can prove to be a major limitation as the size of the input data increases. The Pellet – Jena system, although a bit slower than the BigOWLIM – Sesame system throws a runtime exception in case the input is not in the correct format.

5.2. Discussion

RDF has been accepted as the standard for the storage of semantic web data by the World Wide Web Consortium. Efforts are on to develop systems that are capable of efficient storage and retrieval of RDF data. While the goal is to build systems that are fast and scalable, other factors such as type checking of the input data that affect the adoption and implementation of any system should also be considered.

Based on the results, the author recommends the use of Pellet – Jena system for datasets with less than a million triples. Although the Pellet – Jena system is a bit slower than the BigOWLIM – Sesame system, one does not have to worry about the quality of the input data since it automatically does the type checking. For datasets with more than a million triples, one has to use the BigOWLIM – Sesame system since the Pellet – Jena system is not scalable beyond that for the system configuration used in this project.

5.3. Future Directions

The LUBM dataset was used as the standard for evaluating the performance of the systems. One could expand this study by considering other benchmarks such as the University Ontology Benchmark. (Li & Yang, et al., 2006).

Also, only two of the systems have been considered in this work. There are other systems such RDF – 3X (Neumann & Weikum, 2008), that have been recently developed and should be evaluated thoroughly in order to check for their feasibility in terms of parameters such as their scalability as well as their response time for responding to queries. Also, Pellet has now become the first system to integrate itself with a backend that would be based on Oracle instead of relying on an open source system such as Jena. Initial reports point to much improved performance of such a system.

Finally, one could expand the study by studying the effect of concurrent users on the performance of the system.

5.4. Summary

The essence of this study is to compare the performance of systems such as Pellet and BigOWLIM, built around open source frameworks such as Jena and Sesame respectively, for small and medium enterprises. This chapter summed up the findings in this research. It also presented a general discussion and recommendations for future extensions of the current research.

LIST OF REFERENCES

LIST OF REFERENCES

Abadi, D., Marcus, A., Madden, S., & Hollenbach, K. (2009). SW-Store: a vertically partitioned DBMS for SemanticWeb data. *VLDB Journal , 18*, 385 - 406.

Aduna Corporation. (n.d.). *OpenRDF Corporation*. Retrieved January 10, 2010, from Sesame Web site: http://www.openrdf.org/about.jsp

Bacchus, F., & Winter, M. A. (2001). Planning with Resources and Concurrency A Forward Chaining Approach.

BigOWLIM Corporation. (n.d.). *BigOWLIM Corporation*. Retrieved January 10, 2010, from BigOWLIM Corporation Web site: http://www.ontotext.com/owlim/

Clark & Parsia. (n.d.). *Clark and Parsia Corporation*. Retrieved April 4, 2010, from Clark and Parsia Corporation Web site: http://clarkparsia.com/pellet/

Dong, X., & Halevy, A. (2007). Indexing Dataspaces. *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (pp. 43 - 54). Beijing, China: ACM.

Ferragina, P., Gonzalez, R., Navarro, G., & Venturini, R. (2009). Compressed text indexes: From theory to practice. *ACM Computing Research Repository , 13*, 2.

Groppe, J., Groppe, S., Ebers, S., & Linnemann, V. (2009). Efficient Processing of SPARQL Joins in Memory by Dynamically Restricting Triple Patterns. *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing* (pp. 1231 - 1238). Honolulu, Hawaii: ACM.

Guo, Y., Pan, Z., & Heflin, J. (2004). LUBM: A Benchmark for OWL Knowledge Base Systems. *Third International Semantic Web Conference* (pp. 274 - 288). Hiroshima, Japan: Springerlink.

Haarslev, V., & Moller, R. (2001). Racer System Description. (pp. 701 - 705). Springer - Verlag.

Konopnicki, D., & Shmueli, O. (2005). Database-Inspired Search. *VLDB: Proceedings of the 31st international conference on Very Large DataBases* (pp. 2 - 12). Trondheim, Norway: VLDB Endowment.

Laborda, C., & Conrad, S. (2005). Relational OWL - A Data and Schema Representation Format. *APCCM '05: Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling. 43*, pp. 89 - 96. Newcastle, New South Wales, Australia: Australian Computer Society, Inc.

Lee, K., Kim, G.-W., Son, J., & Kim, M. (2008). Web Document Compaction by Compressing URI references in RDF and OWL Data. *ICUIMC '08: Proceedings of the 2nd international conference on Ubiquitous information management and communication* (pp. 163 - 168). Suwon, Korea: ACM.

Neumann, T., & Weikum, G. (2008). RDF - 3X: a RISC - style engine for RDF. *Proc. VLDB Endowment , 1* (1), 647 - 659.

Neumann, T., & Weikum, G. (2009). Scalable join processing on very large RDF graphs. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data* (pp. 627 - 640). Providence, Rhode Island, USA: ACM.

Rohloff, K., Dean, M., Emmons, I., Ryder, D., & Summer, J. (2007). An Evaluation of Triple-Store Technologies for Large Data Stores. In *On the Move to Meaningful Internet Systems* (pp. 1105 - 1114). Berlin: Springerlink.

Sourceforge. (n.d.). *Sourceforge Corporation*. Retrieved April 4, 2010, from Sourceforge Corporation Web site: http://jena.sourceforge.net/

Spink, A., Wolfram, D., Jansen, B., & Saracevic, T. (2000). Searching The Web: The Public and Their Queries. *Journal of American Society for Information Science and technology , 52* (3), 226 - 234.

SwiftOWLIM Corporation. (n.d.). *SwiftOWLIM Corporation*. Retrieved January 10, 2010, from SwiftOWLIM Corporation Web site: http://www.ontotext.com/owlim/

Weiss, C., Karras, P., & Bernstein, A. (2008). Hexastore: Sextuple Indexing for Semantic Web Data Management. *Proc. VLDB Endowment , 1* (1), 1008 - 1019.

World Wide Web Consortium. (n.d.). *World Wide Web Consortium*. Retrieved April 6, 2010, from World Wide Web Consortium Web site: HTTP://WWW.W3.ORG/2004/OWL/