7-11-2011

Comparison of Clustered RDF Data Stores

Venkata Patchigolla
Purdue University, patchigolla.rama@gmail.com

Patchigolla, Venkata, "Comparison of Clustered RDF Data Stores" (2011). *College of Technology Masters Theses.* Paper 43.
http://docs.lib.purdue.edu/techmasters/43

COMPARISON OF CLUSTERED RDF DATA STORES

A Thesis
Submitted to the Faculty
of
Purdue University
by
Venkata N. Ramarekha Patchigolla

In Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

August 2011
Purdue University
West Lafayette, Indiana

To my parents, sister and friends for their love, guidance and support.

ACKNOWLEDGMENTS

I feel fortunate to be a part of the Computer and Information Technology department at Purdue University. I am thankful to my chair, advisor and guide Prof. John Springer for his full support and guidance during the research. Prof. Springer was a constant source of motivation throughout. I am grateful for his valuable comments and advise during the course of project.

I am also thankful to my committee members Prof. Raymond Hansen and Prof. Kari Clase for their insightful comments and feedbacks which helped me refine my work.

I am grateful to Cummins College of Engineering, Pune for providing me the opportunity to study Master’s in Computer and Information Technology department at Purdue University. I offer my gratitude to Prof. Gail Farnsley, for helping me and guiding me throughout my Master’s degree.

I am thankful to my parents and sister for their everlasting encouragement and support.

TABLE OF CONTENTS

Page

Page

APPENDICES

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Patchigolla, Venkata Naga Ramarekha. M.S., Purdue University, August 2011. Comparison of clustered RDF data stores. Major Professor: John Springer.

Storing data in RDF format helps in simpler data interchange among different researchers compared to present approaches. There has been tremendous increase in the applications that use RDF data. The nature of RDF data is such that it tends to increase explosively. This makes it necessary to consider the time for retrieval and scalability of data while selecting a suitable RDF data store for developing applications. The research concentrates on comparing BigOWLIM. Bigdata, 4store and Virtuoso RDF stores on basis of their scalability and performance of storing and retrieving cancer proteomics and mass spectrometry data using SPARQL queries. In this research the author compares RDF data stores on a single machine as baseline and extends 4store and BigOWLIM data stores on a cluster for comparison. The author uncovers that Virtuoso has the best performance on data consisting of less than 250,000 triples whereas 4store has better scalability and performance for the larger data.

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 Semantic Web (Berners-Lee et al., 2001) includes various technologies that allow machines to understand and infer the information present in the World Wide Web. These technologies help machines to communicate with each other, regardless of the format in which data is stored. Resource Description Framework (RDF) and Web Ontology Language (OWL) are some of the technologies included in the Semantic Web as recommended by World Wide Web Consortium (W3C). RDF consists of triples of the form subject-predicate-object. OWL is a layer on top of RDF, which is used to process RDF data. OWL is designed to be interpreted by computers. OWL has inference power which empowers machines for logical analysis. OWL is a stronger language and it has larger vocabulary, for describing properties and classes, than RDF. OWL adds semantics to the schema. SPARQL Protocol and RDF Query Language (SPARQL) is the query language used for querying RDF data. Few relational databases support RDF data. RDF is a structured language that computer applications can use for understanding semantics of data. There has been an increase in the use of applications that use RDF data. Many custom built RDF stores have been developed and are available for use. The scope of the thesis is to compare a few of these custom built RDF data stores for storing and retrieving data using SPARQL.

1.2. Significance

RDF is a method of expressing knowledge. RDF is a structured language used to store data along with its semantics. Computer applications can use this RDF to understand semantics of this data. RDF is very useful for integrating data from different sources. RDF can be used for simpler data interchange and reuse by other researchers. These factors have increased the importance of using and storing data in RDF format. RDF data was initially stored in relational databases. However, there has been increase in the availability of RDF data, and this has led to development of different custom based solutions for storing RDF data. The nature of RDF data is that it tends to increase explosively. Therefore, it is necessary to consider the scalability of data while selecting a suitable RDF data store for developing applications. Different RDF stores available today use different mechanisms to enable the scalability of data. While using RDF data for an application, it is necessary to select a suitable RDF data store. These RDF stores need to be compared on various factors such as performance, scalability, etc. that will help in making the right selection of RDF stores depending on the nature of RDF data that an application has. Thus, it becomes very important to compare and evaluate the various RDF stores that are available.

1.3. Research Question

To compare and understand various RDF stores for their scalability and retrieval using SPARQL queries on cancer proteomics data.

1.4. Assumptions

The assumptions of this research study include:

1. The RDF data generated from Clinical Proteomic Technology Assessment for Cancer (CPTAC) data set is assumed to be a true representation of the data in an RDF store.
2. The proposed system is assumed to be a standalone system (i.e., there do not exist multiple users querying the data store simultaneously).
3. All the network delays are assumed to be constant for all the clustered stores.
4. Network transfer time is negligible.
5. The setup of clusters, CPU and network is not favorable to any of the clustered stores in comparison.
6. The queries used for data manipulation are a subset that covers in general all the queries that could be executed on the data.
7. The incremental load process did not impact the query response times.

1.5. Delimitations

The delimitations of this research study include:

1. Only two clustered RDF stores and 4 single machine RDF stores are considered for comparison among various RDF stores available.
2. The clusters built consist of only 4, 6 and 9 nodes.
3. The data used has maximum input of 1,000,000 triples only.
4. Data was loaded in incremental fashion into the repository for comparison.

1.6. Limitations

The limitations of this research study include:

1. The author considers data generated for the cancer proteomics research as the input data for the RDF stores (i.e., the data used may not be generalized RDF data).
2. The author tests the RDF data store for their performance and correctness of data retrieval. There are other characteristics of the system which are not being considered for comparison.
3. The author uses SPARQL queries for querying RDF data as the standard for the World Wide Web. There is no attempt look for any other RDF data querying language.

1.7. Definition

Metadata – It is data about data. It describes the data.

Resource Description Framework (RDF) – Triples having the form (subject, object, predicate) and primarily used for storing data on the World Wide Web (Groppe, Groppe, Ebers, & Linnemann, 2009)

Semantic Web – It is about giving meaning to the information available on the web such that computers and machine can understand and use the data meaningfully.

SPARQL Protocol and RDF Query Language (SPARQL) – It is the query language that is primarily used for querying the RDF data. (Neumann & Weikum, 2008)

Schema – It is a way to define the structure, content and semantics of data.

Sesame – It is a standard framework for storing, inferencing and querying RDF data and RDF schema information. (Kampman, Harmelen, & Broekstra, 2002)

Web Ontology Language (OWL) – It describes the relationships between the three RDF components. (Laborda & Conrad, 2005)

1.8. Summary

This chapter described the motivation behind the research work. It presented the scope and research question. It also provided assumptions, delimitations and limitations in the study. It also gave a definition of the key terms used in the proposal.

CHAPTER 2. LITERATURE REVIEW

The World Wide Web is the biggest repository of information available today. The web content available today is designed for the humans to read and understand. Searching and sorting through the enormous amount of data on the web to get the relevant information is becoming seemingly difficult. Hence, the need arises to organize the data on web such that it is machine understandable.

2.1 Background

This section explains the Semantic Web and various technologies that are being utilized in the Semantic Web.

2.1.1. Semantic Web

The Semantic Web is a group of technologies used to give meaning to the information available on the web so that computers and machine can understand and use the data meaningfully. Using the Semantic Web structured and meaningful web pages will be generated which are used by software agents to understand and create inference. Consider a scenario where you want to schedule an appointment with your dentist. You activate your Semantic Web agent to schedule an appointment. The agent will synchronize your daily calendar and dentist’s timing. Then agent would infer and suggest a date and time for a suitable appointment. Then with one click the agent will go ahead and schedule an appointment for you with the dentist. Thus, the Semantic Web is “A new form of Web content that is meaningful to computers and will unleash a revolution of new possibilities”. (Tim, James, & Ora, 2001). The Semantic Web is becoming very popular. It is being used in various applications in the fields of searching the web, biological research and electronic commerce to name a few.

2.1.2. RDF

Resource Description Framework (RDF) is the tool that provides a way of storing the representation of the metadata. Metadata is a description of the data. RDF is a data format for representing information on the web. The data in RDF is stored in form of triples and directed graphs and is expressed as a triple: <subject, predicate, and object>. RDF triples have Uniform Resource Identifiers (URIs). URIs are identifiers that give the location of the description of data. RDF offers a great deal of flexibility when the schema is not known. RDF data is stored in RDF repositories that can be queried by using languages such as SPARQL. (Selcuk, Huan, & Reshma, 2001).

2.1.3. SPARQL

Researchers such as Prud’hommeaux, and Seaborne (2008) state that “SPARQL query language for RDF (SPARQL) is the language used to query RDF data.” SPARQL queries are used to query RDF data stores to obtain results. This makes it easy for machines and humans to connect to the store and get the relevant data.

2.2. RDF Data Store

Performance and scalability are very important issues to be addressed while storing RDF data. Finding solutions for efficient storage & retrieval of RDF data is very important. Researchers Abadi, Marcus, Madden, and Hollenbach

(2007) investigated the issue of providing scalable RDF data store. Initially RDF was stored in Relational DBMS. They provided two approaches

a) Vertically partition the database and

b) Column-Oriented database to improve the scalability of the system.

The vertically partitioned store will contain various two column tables based on their properties. Each property table has a subject and an object. The column- oriented store will store tuples in columns instead of rows. These two approaches were evaluated. It was found that both of them improve the performance and scalability of system. While vertical partitioning was better than column based approach it could be used only for subset of RDF data and thus cannot generalize it.

Another solution was proposed by Weiss, Karras, and Bernstein (2008), which treats RDF data as triples and stores them in a relational DBMS. Instead of treating triples differently, they indexed the data by creating a Hexastore that indexed the data in 6 different ways. They evaluated this and found that performance is improved as compared to vertical partitioning. But the storage memory required for this store increased rapidly as compared to vertical partitioning.

One of the ways used to address scalability of RDF data is the use of clustered RDF data stores. Weave and Williams (2009) built a clustered store for storing RDF data without any preprocessing. They used Beowulf clusters and an IBM Blue Gene/L supercomputer to generate a system for answering basic graph pattern queries over large RDF data sets on clusters. Since then, various clustered stores have become available. A cluster’s parallelism is utilized to load and query the data in much faster way as compared to sequential approaches. Thus, performance and scalability of the RDF data store is improved.

Harris, Lamb and Shadbolt (2009) described 4store which is a clustered RDF store. This was built as a backend for application called garlik. A RID integer is calculated for the subject of any given triple. A triple is then put in a segment which is calculated as a function on Resource ID (RID) of a given triple. In this store RDF data is stored in quad format with each RDF triple having a model associated with it. 4store is queried using SPARQL queries.

Clustered TDB is another approach to store RDF triples in a clustered form. This forms a clustered backend for Jena. It has a query coordinator and data nodes. Query coordinator decides the node to which data is to be sent. It distributes each triple three times based on its three indexes of subject, property, and object. Thus, clustered TDB does partitioning of data. It is one more approach proposed to store large volumes of RDF data (Alisdair, Andy, & Nick, 2008). Other approaches for clustered stores include YARS2 (Andreas, J¨urgen, Aidan, & Stefan, 2007). It is an end to end semantic search engine that stores RDF data as graphs and uses distributed indexing and parallel query methods on the data stored in the cluster.

2.3. Evaluation of RDF data stores

Various researchers have tried to develop benchmarks and other ways to evaluate RDF data stores.

2.3.1. LUBM

It is a benchmark developed for evaluating large scale knowledge based systems. This was developed to evaluate RDF storage mechanisms. LUBM uses synthetic data for evaluation. This data consisted of an ontology developed for university data. They designed test queries taking into account input size, selectivity, complexity, hierarchy information and assumed logical inference.

From the queries they formulated they evaluated load time, query response time, completeness, and repository size required to store the data which was synthetically generated. (Yuanbo, Zhengxiang, & Jeff, 2005) LUBM is an important benchmark for evaluating Semantic Web data stores. The queries mentioned have been used for different evaluations.

2.3.2. Other Approaches

Ma, Yang, Qiu, Xie and Pan (2006) noted that LUBM was developed for specific types of ontologies. It did not consider OWL lite and OWL DL when benchmark was being developed. They tried to derive a complete ontology benchmark. In their system data generated could be of type OWL lite or OWL DL. They evaluated their systems based on this data. They further discussed native storage and DBMS based approaches and came to conclusion that native storage improved the performance as compared to DBMS approaches.

Similar results were found when Liu and Hu (2005) performed evaluation of seven large scale data storage systems with respect to data loading time and query response time. They used LUBM queries for comparisons of data stores. They used memory based RDF stores, persistent RDBMS stores that could store RDF data, and three native RDF systems. They concluded that the performance of native RDF systems is better.

Alisdair O (2009) performed an investigation in improving the performance of RDF data stores. He described various benchmarks used for performance evaluation. He designed a new RDF based test cases which offer a wider variety of tests and clarity as compared to LUBM. Also user has designed use case based test benchmarks.

2.4. Summary

This chapter provided the motivation for RDF and Semantic Web technologies to be developed and their existence. This chapter gave information on various attempts on evaluating these RDF stores and various benchmarks thus evolved. Though various benchmarks have been developed and used for comparing these systems and many evaluations have been made, most of these consisted of synthetically generated data. Also it is to be noted there has been no comparison of clustered RDF stores. With the increase of RDF applications being made there needs to be more evaluations done with the real data.

CHAPTER 3. METHODOLOGY

This chapter describes the framework that has been used to evaluate the performance of RDF data stores for proteomics data. It includes the overview of the ontology and description of the RDF data stores used. The author has evaluated the performance of 4 data stores on a standalone machine and 2 data stores on clusters of 4, 6 and 9 nodes.

3.1. Framework for Evaluation

The evaluation framework consists of an evaluation of the performance and scalability of 4 RDF data stores on a standalone machine and extending 2 RDF data stores to clusters with 4, 6 and 9 nodes respectively. It is a quantitative research consisting of data of different sizes loaded in the data stores and queried using SPARQL queries to measure the data stores’ scalability and performance.

3.1.1. System Setup

The framework for evaluation is shown in figure 3.1. :

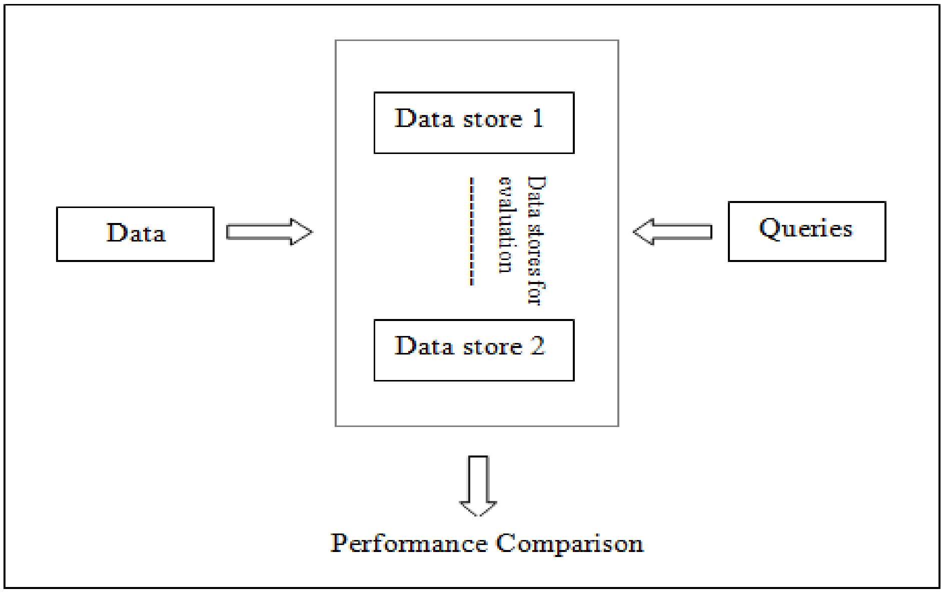


Figure 3.1. Framework for performance comparison

The hardware specifications of each machine used are:

- 1000 Mhz Dual core AMD Opteron(tm) Processor 180
- 2 GB RAM
- 1 MB Cache
- 145 GB Hard disk

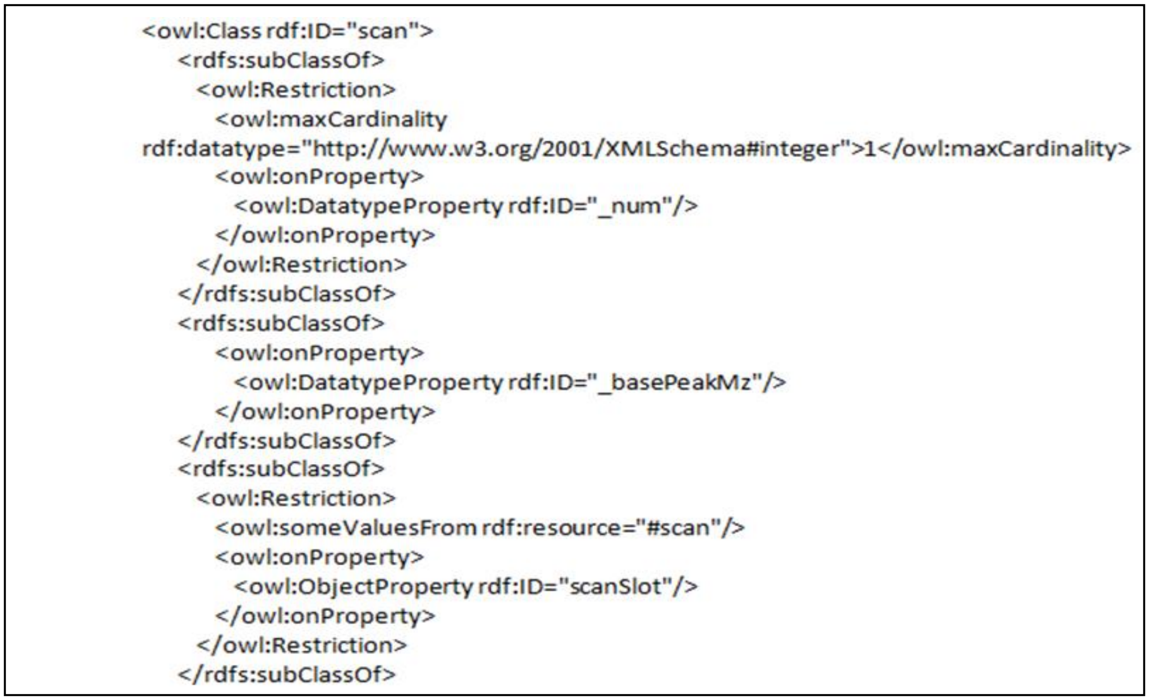
The software specifications used are:

- Operating System: Linux Fedora Core 12 x86_64
- Java JDK 1.6.0_18 with Tomcat 6
- Sesame 2.3.2
- BigOWLIM 3.5
- Bigdata
- 4store 1.1.3
- Virtuoso-Opensource 6.1.1

3.2. Ontology used – Proteomics data

The ontology used for comparison consists of cancer proteomics data. The data was generated from a mass spectrometry tool used for the evaluation of proteins in a biological system. This data is in mzXML format. The ontology was then extracted from the mzXML file using XSL (Extensible Stylesheet Language). OWL files generated were then loaded into the data stores for evaluation.

The ontology used describes the results from mass spectrometry instruments. It also includes metadata such as the type of instrument used, data processing techniques, and the software used along with the information about each scan and peaks observed in the scan. The ontology also consists of points “mz” and “intensity” for the graphs generated during each scan.



```
<owl:Class rdf:ID="scan">
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1</owl:maxCardinality>
       <owl:onProperty>
         <owl:DatatypeProperty rdf:ID="_num"/>
       </owl:onProperty>
     </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
       <owl:onProperty>
         <owl:DatatypeProperty rdf:ID="_basePeakMz"/>
       </owl:onProperty>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:someValuesFrom rdf:resource="#scan"/>
       <owl:onProperty>
         <owl:ObjectProperty rdf:ID="scanSlot"/>
       </owl:onProperty>
     </owl:Restriction>
   </rdfs:subClassOf>
```

Figure 3.2. Snapshot of proteomics ontology

This ontology defines all the tags in the mzXML file along with the properties of each tag. It has 19 classes and 44 properties.

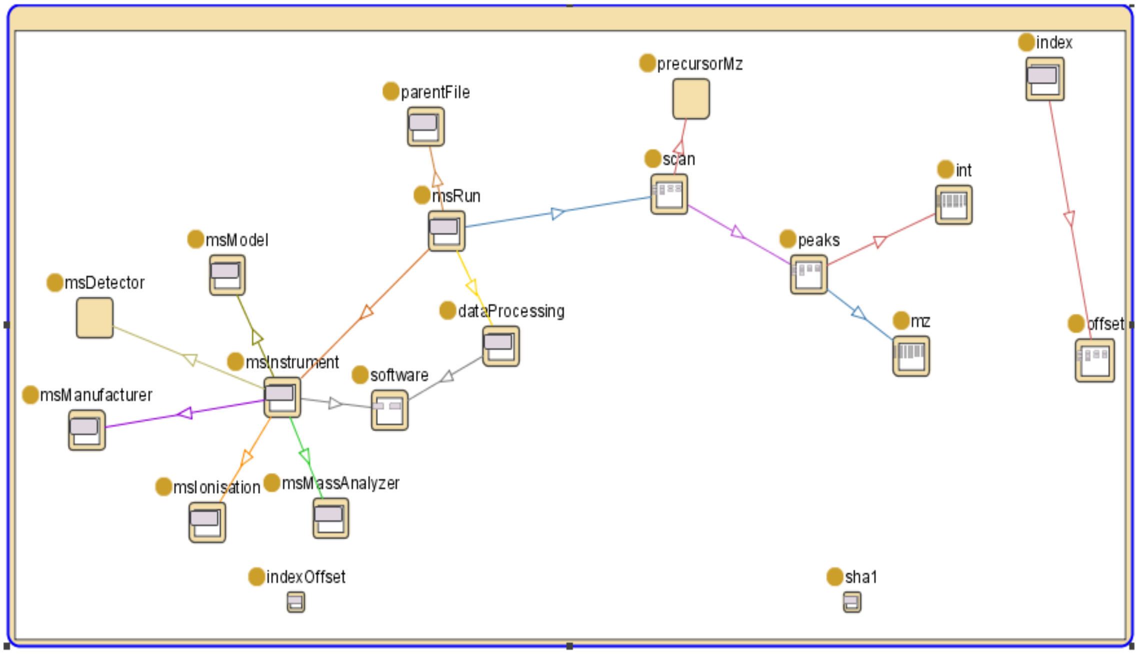


Figure 3.3. Hierarchy of the owl files used.

3.3. Variables

The independent variables that were manipulated during this study were: the size of the data set (that is, the number of triples) and the number of nodes in a cluster. The number of triples was varied as follows to test the performance of the systems:

- 10,000
- 50,000
- 100,000
- 250,000
- 500,000
- 750,000
- 1,000,000 triples

The number of nodes in a cluster was varied to use 4, 6 and 9 nodes. Additionally 4 types of queries were used to get the results. The dependent variable was the mean time taken for the query execution in milliseconds with removal of outliers.

3.3.1. Test Queries

The queries were formulated to be run against the data loaded. Using LUBM queries as a basis, four types of test queries were generated. LUBM is a widely used benchmark for comparing semantic web databases. These queries mainly take into account Input size, Selectivity and Complexity. The queries have been briefly described as follows:

Query 1:

This query is similar to LUBM Query 1. It has large input and high selectivity:

```
PREFIX owl:<http://www.owl-ontologies.com/mzxml.owl#>

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

PREFIX xml:<http://www.w3.org/2001/XMLSchema#>

select ?x where {?s rdf:type owl:scan. ?s owl:_basePeakMz
"444.97509766"^^xsd:string. ?s owl:_num ?x}
```

This query returns the scan numbers of all the scans having peakscount basePeakMz 444.97509766.

Query2:

This query is similar to LUBM Query 4. It is a complex query that queries subclasses.

```
PREFIX owl:<http://www.owl-ontologies.com/mzxml.owl#>

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

select ?w ?x ?y ?z where {?s rdf:type owl:scan. ?s owl:_num ?x. ?s
owl:_peaksCount ?w. ?s owl:_retentionTime ?y. ?s owl:_polarity ?z}
```

This query returns the properties- num, retention time and polarity and peakCount of scans.

Query 3:

This query is similar to the LUBM Query 2. This query has an hierarchical relationship. In the data set peakslot has a child relationship with scan and mzslot has a child relationship with peak slot. In addition, the mzslots elements have m/z values contained within m/z-int pairs.

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

PREFIX xml:<http://www.w3.org/2001/XMLSchema#>

select DISTINCT?x ?y ?z ?w where {?x rdf:type owl:scan. ?y rdf:type owl:peaks.
?z rdf:type owl:mz. ?x owl:peaksslot ?y. ?y owl:mzslot ?z. ?x owl:_num ?w.
FILTER(?w < 53).}
```

This query returns scan id, peak id and mz ids of scan numbers less than 53. In this query the evaluation of results is also important since three relations must be satisfied along with a condition.

Query 4:

This query uses SPARQL features: FILTER and DISTINCT

```
PREFIX owl:<http://www.owl-ontologies.com/mzxml.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX xml:<http://www.w3.org/2001/XMLSchema#>

select DISTINCT?x ?y where {?x rdf:type owl:scan. ?x owl:_num ?y  FILTER (?y
< 150)} order by ?y
```

This query returns scan id and number of the scans less than 150. Here the performance of a store will depend not only on number of results but time taken to evaluate and output the results in increasing order.

3.4. Target Systems

Following 4 systems were used for evaluation:

1. 4store: 4store (Garlik) is a scalable and stable RDF database. In 4store RDF triples are stored in Quad format. (Harris, Lamb, & Shadbolt, 2009)
2. BigOWLIM: BigOWLIM (BigOWLIM Corporation) is a high performance semantic repository available as storage and inference layer on top of the Sesame framework (Aduna).
3. Bigdata: Bigdata (Systap) is a high performance database designed for large scale semantic data. The stand-alone system has the Sesame framework as the SPARQL endpoint. There is presently no SPARQL endpoint developed for the clustered version of Bigdata.
4. Virtuoso: Virtuoso (Erling, & Mikhailov, 2007) is a multi model data server with a RDF triple store. RDF triple store on a stand-alone machine is open source whereas the clustered version is a commercial edition.

Initially the author had decided to compare performance of all the 4 systems on stand alone and cluster architecture. Due to the above mentioned difficulties experienced for Bigdata and Virtuoso systems, only BigOWLIM and 4store were used for comparison on cluster architecture.

3.5. Summary

This chapter focused on framework and methodology used for comparison of RDF stores. Data used as the input to system was discussed along with the types of queries that were executed for evaluation.

CHAPTER 4. EVALUATION AND DATA ANALYSIS

This chapter presents the data analysis performed on the results. It presents the comparison of performances of target system with respect to different sizes and queries.

4.1. Evaluation

Each test query was executed 100 times and the mean performance time was noted. This gives better estimate and neutralizes anomalies that might occur during a run. The averages of the results were then calculated. All the tests on different target data stores were performed on the same machine to reduce the error due to a change in environment. Bash script was used to run the queries 100 times.

4.1.1. Single Machine

On a single machine each query was executed and its results were noted as follows:

Table 4.1.

Query response time for Query 1

No. of Triples	No. of Scans	No. of Results	Bigdata	BigOWLIM	4store	Virtuoso
10,000	6	2	52.66	44.48	39.54	3.57
50,000	13	2	48.63	41.2	39.36	3.72
100,000	21	2	46.96	45.38	39.18	3.54
250,000	45	2	45.34	43.79	39.09	3.65
500,000	72	2	46.21	46.81	39.22	3.87
750,000	123	2	46.45	47.26	39.19	3.8
1,000,000	264	2	41.59	43.23	38.67	4.17

Table 4.2.

Query response time for Query 2

No. of Triples	No. of Scans	No. of Results	Bigdata	BigOWLIM	4store	Virtuoso
10,000	6	6	57.8	46.41	49.09	7.1
50,000	13	13	59.03	44.36	48.47	13.78
100,000	21	21	58.94	49.18	49.11	9.2
250,000	45	45	74.83	63.44	50.96	21.54
500,000	72	72	94.55	82.23	51.59	14.93
750,000	123	123	114.11	100.47	54.82	22.36
1,000,000	264	264	153.69	144.81	71.64	35.85

Table 4.3.

Query response time for Query 3

No. of Triples	No. of mzslots	No. of Results	Bigdata	BigOWLIM	4store	Virtuoso
10,000	1588	386	171.46	158.33	71.24	94.61
50,000	9799	3868	675.28	615.35	107.19	652.78
100,000	22439	10287	1248.84	1183.32	174.71	1527.11
250,000	50091	20060	1914.19	2024.19	322.62	3345.71
500,000	98770	68322	6032.02	4867.61	710.82	10805.47
750,000	144278	113830	10584.32	7698.55	1068.82	17176.79
1,000,000	188817	150025	14426.2	13971.92	1279.67	22060.76

Table 4.4

Query response time for Query 4

No. of Triples	No. of Scans	No. of Results	Bigdata	BigOWLIM	4store	Virtuoso
10,000	6	6	47.331	44.32	58.35	3.59
50,000	13	13	47.78	44.01	57.73	4.7
100,000	21	21	49.97	46.29	58.6	3.83
250,000	45	45	52.84	56.24	60.25	9.66
500,000	72	72	63.54	63.87	64	7.68
750,000	123	99	70.12	71.61	64.72	16.73
1,000,000	264	132	83.46	84.2	65.73	12.3

4.1.2. Cluster Machine

Clusters of 4, 6 and 9 nodes (including master nodes) were used for evaluation. Each query was executed for different data sets on the cluster model and its performance was noted down.

4.1.2.1. 4store

In order to edit the number of nodes in 4store cluster the author edited /etc/4store to give the host names of the machines to be used. All the machines in the cluster had same configuration and 4store was installed on each of them. The master node was Achilles with an IP address 10.112.42.10. Data sets and queries used for evaluation were same.

Table 4.5

Query response time for Query 1 for clustered 4store

No. of Triples	No. of Scans	No. of Results	Single Machine	4Nodes	6Nodes	9Nodes
10,000	6	2	39.54	45.94	46.34	48.12
50,000	13	2	39.36	45.91	48	48.13
100,000	21	2	39.18	45.5	47.55	48.86
250,000	45	2	39.09	45.45	47.20	48.73
500,000	72	2	39.22	46.02	45.99	48.14
750,000	123	2	39.19	46.01	48.47	48.48
1,000,000	264	2	38.67	46.21	47.78	48.39

Table 4.6.

Query response time for Query 2 for clustered 4store

No. of Triples	No. of Scans	No. of Results	Single Machine	4Nodes	6Nodes	9Nodes
10,000	6	6	49.09	55.96	55.85	58.52
50,000	13	13	48.47	59.19	60.06	60.33
100,000	21	21	49.11	58.38	59.61	62.83
250,000	45	45	50.96	61.55	62.89	64.56
500,000	72	72	51.59	64.97	65.89	67.31
750,000	123	123	54.82	68.14	69.89	70.60
1,000,000	264	264	71.64	93.5	94.18	94.67

Table 4.7.

Query response time for Query 3 for clustered 4store

No. of Triples	No. of mzslots	No. of Results	Single Machine	4Nodes	6Nodes	9Nodes
10,000	1588	386	71.24	114.83	117.50	119.55
50,000	9799	3868	107.19	197.6	189.75	192.60
100,000	22439	10287	174.71	276.02	259.98	262.43
250,000	50091	20060	322.62	440.34	429.94	417.46
500,000	98770	68322	710.82	925	847.63	831.96
750,000	144278	113830	1068.82	1365.64	1269.24	1230.15
1,000,000	188817	150025	1279.67	1688.44	1537.28	1494.30

Table 4.8.

Query response time for Query 4 for clustered 4store

No. of Triples	No. of Scans	No. of Results	Single Machine	4Nodes	6Nodes	9Nodes
10,000	6	6	58.35	64.97	66.48	66.15
50,000	13	13	57.73	65.57	67.70	68
100,000	21	21	58.6	65.91	66.56	68.69
250,000	45	45	60.25	69.03	69.98	69.99
500,000	72	72	64	71.52	72.47	74.82
750,000	123	99	64.72	74.44	75.37	75.70
1,000,000	264	132	65.73	75.47	76.04	76.48

4.1.2.2. BigOWLIM

In order to edit the number of nodes in BigOWLIM cluster the author copied the cluster template file to sesame repository directory on the master. The data store built for single machine and that for cluster configuration is different in the master. All the other machines in the cluster had same configuration. BigOWLIM and Sesame were installed along with a stand-alone repositories were created. Master node was Achilles with IP address 10.112.42.10. The author used Jconsole (JMX interface) to connect the master node with the worker nodes. Data sets and queries used for evaluation were same. All the commands were executed from the master. The master was configured writable to allow loads to be executed.

BigOWLIM parses the RDF data in heap memory and then processing is performed to store the data onto worker nodes. The maximum heap space for java on master given was 1GB. Thus, with small heap space BigOWLIM was able to load and query data on a 4 nodes cluster but the loading data failed for 100,000 triples and more in 6 nodes and failed for 9 nodes cluster. Hence the queries on only 4 nodes cluster were executed. The results are as follow:

Table 4.9.
Query response time for Query 1 for clustered BigOWLIM

No. of Triples	No. of Scans	No. of Results	Single Machine	4Nodes	6Nodes
10,000	6	2	44.48	72.46	49.87
50,000	13	2	41.2	59.36	54.88
100,000	21	2	45.38	55.25	50.18
250,000	45	2	43.79	54.72	
500,000	72	2	46.81	51.3	
750,000	123	2	47.26	51.65	
1,000,000	264	2	43.23	56.02	

Table 4.10.

Query response time for Query 2 for clustered BigOWLIM

No. of Triples	No. of Scans	No. of Results	Single Machine	4Nodes	6Nodes
10,000	6	6	46.41	68.01	59.05
50,000	13	13	44.36	59.96	56.71
100,000	21	21	49.18	64.99	60.87
250,000	45	45	63.44	75.67	
500,000	72	72	82.23	90.45	
750,000	123	123	100.47	107.62	
1,000,000	264	264	144.81	152.67	

Table 4.11.

Query *response time for Query 3 for clustered BigOWLIM*

No. of Triples	No. of mzslots	No. of Results	Single Machine	4Nodes	6Nodes
10,000	1588	386	158.33	201.76	176.7
50,000	9799	3868	615.35	802.8	798.13
100,000	22439	10287	1183.32	1446.32	1455.34
250,000	50091	20060	2024.19	2015.73	
500,000	98770	68322	4867.61	5148.81	
750,000	144278	113830	7698.55	8583.63	
1,000,000	188817	150025	13971.92	16291.39	

Table 4.12.

Query response time for Query 4 for clustered BigOWLIM

No. of Triples	No. of Scans	No. of Results	Single Machine	4Nodes	6Nodes
10,000	6	6	44.32	62.27	55.69
50,000	13	13	44.01	58.3	56.01
100,000	21	21	46.29	58.18	57.24
250,000	45	45	56.24	66.91	
500,000	72	72	63.87	73.62	
750,000	123	99	71.61	89.53	
1,000,000	264	132	84.2	97.79	

4.2. Graphical Representation

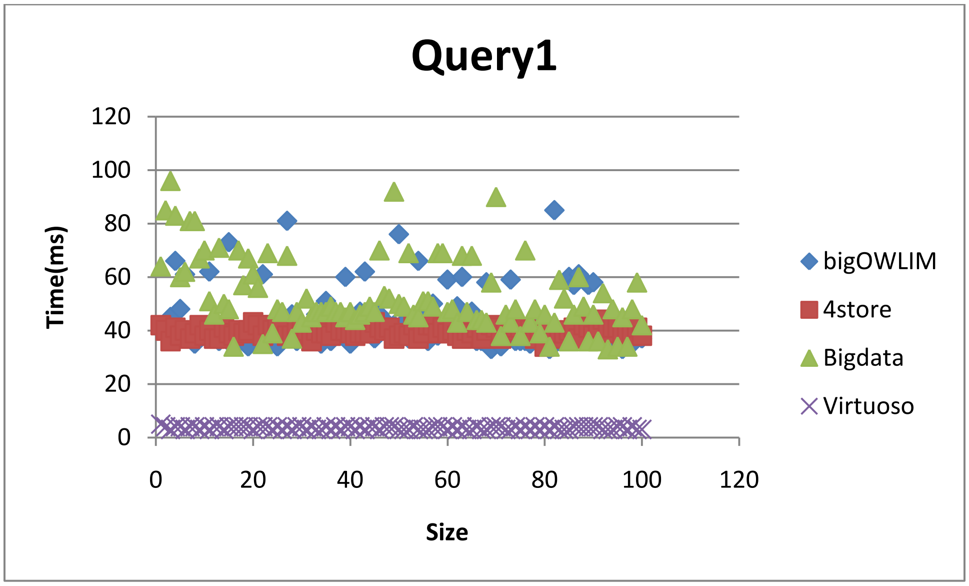


Figure 4.1: Scatter plot for data set1 and query1

A scatter plot was drawn to check the consistency of the data. The above figure shows the query response time query 1 with 10,000 triples. The data has similar variations for other data sets and query combinations. Hence, only one scatter plot has been included. The author observed that data was randomly distributed. It was observed that time taken by Virtuoso to process the query was much less than other systems.

4.3. Performance

The study was focused on evaluating the performance of 4 target systems for proteomics type data in order to get efficient retrieval when queried. This study interpreted performance as consistently fast retrieval of data when various types of queries were executed on the system. Performance was measured based on data query executed, number of triples/size of data.

4.4. Scalability

The study focused on scalability of the target systems for proteomics data. This study interpreted scalability as ability to handle increase in data in a graceful manner. Also the study interpreted scalability for clustered system as ability to execute queries in a faster manner when the number of nodes of a cluster was increased.

4.5. Comparison

Results obtained from running the queries were recorded and the means of each query execution were calculated. The data was graphically represented to ease the process of comparison of the target systems with respect to performance and scalability.

4.5.1. Single Machines

The red line shows the response time of BigOWLIM, the green line is for response time of 4store, the purple is for Bigdata and blue is for Virtuoso. The horizontal axis gives the data set and the vertical axis gives the time taken for execution in milliseconds.

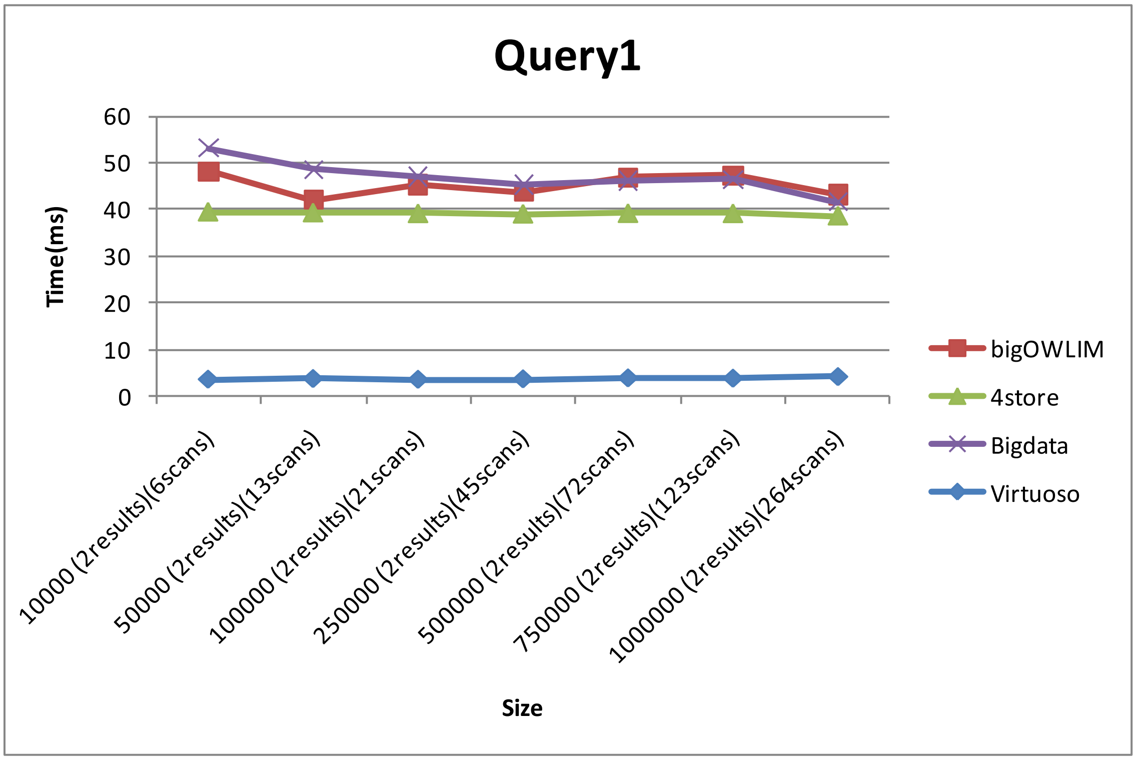


Figure 4.2: Query Response for query 1

Query 1 is a simple query with large input and high selectivity. The resulting data set for this query is very small. In this query scans containing certain property were selected. From the graph it can be deduced that this query has high selectivity and also there is a direct correlation between the number of triples and the number of scans. It is observed that performance of Virtuoso is much better than any other system for such kinds of queries. Also if we take a closer look at individual systems for scalability, we can see that the trend is almost the same as the number of triples in the data set increase. Since the number of results is constant and small we do not observe any significant change in retrieval times although number of triples and scans increase significantly.

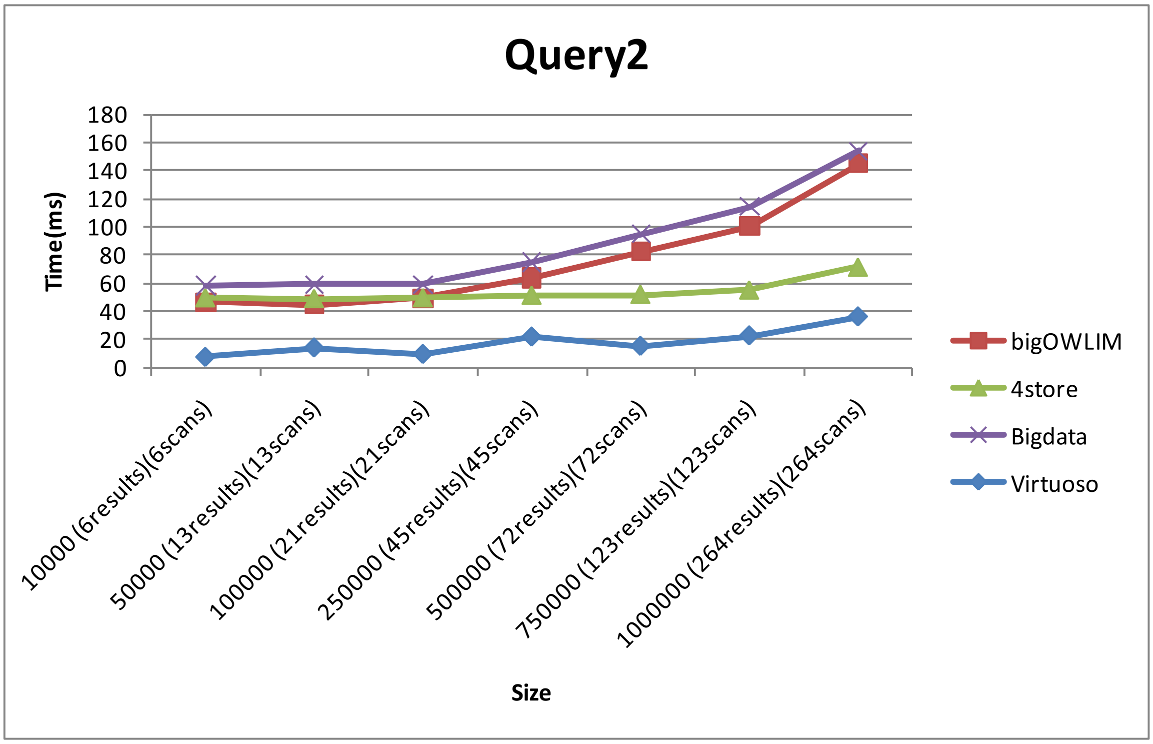


Figure 4.3: Query Response for query 2

Query 2 queries subclasses of class scan. As the data set size increases the number of scan objects also increase. Hence the resulting data set for this query increases with the number of scans. It is observed that here also performance of Virtuoso is better than the other systems. But as the number of triples in data set increase, the time taken by Virtuoso for execution increases rapidly as compared to the other systems. For the last data set it is observed that there is decrease in the gap between Virtuoso and 4store curves. Also it can be observed that BigOWLIM and Bigdata have steeper curves as compared to Virtuoso and 4store.

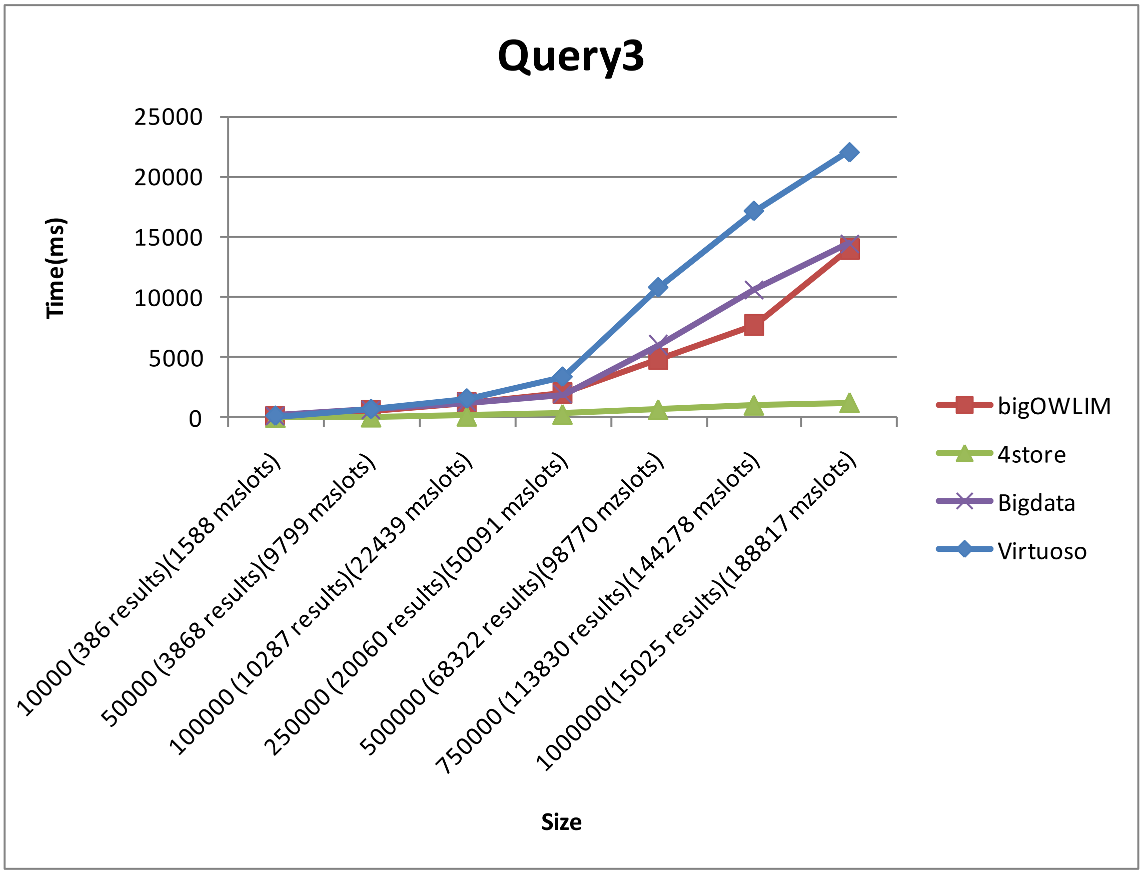


Figure 4.4: Query Response for query 3

Query 3 is a correlation query. This query has a hierarchical relationship where peakslot has a child relationship with scan and mzslot has a child relationship with peakslot. Here there is more emphasis on evaluation of query. Also it can be observed that there is a large increase in time taken by Virtuoso when the number of results increases. For this particular query it is observed that 4store gives a better query performance. It can also be seen that Virtuoso has a very poor scalability, whereas 4store has the best scalability for this kind of queries. BigOWLIM and Bigdata perform similarly, which is better than Virtuoso as the number of results increases.

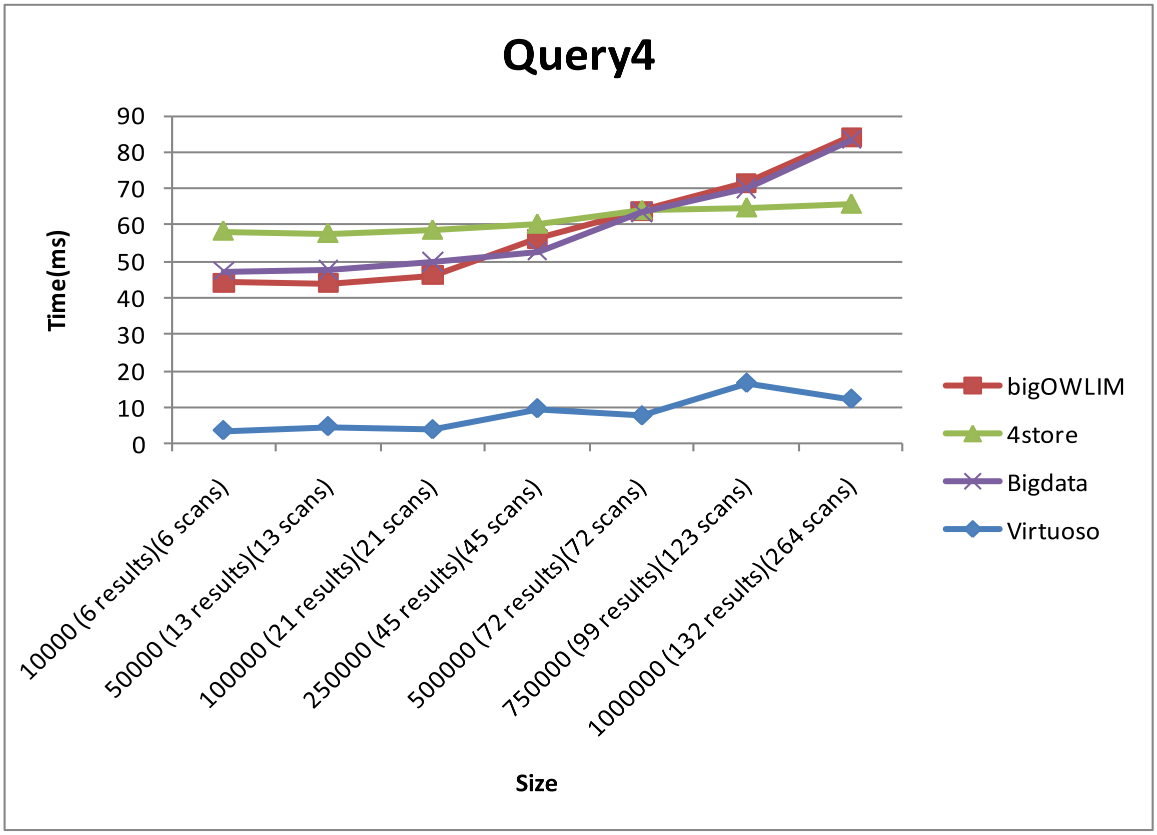


Figure 4.5: Query Response for query 4

Query 4 is a simple query which tries to check the functionalities of SPARQL: Distinct, Filter and Order By. The output of this query has few results. This query is similar to query 1. As already observed for such kinds of queries, Virtuoso performs better than any of the other system. Also it can be observed that although the time taken by Bigdata and BigOWLIM for the initial data set is less than 4store, 4store performs better as the size of the data set increases. Moreover, BigOWLIM performs better than Bigdata for smaller output and data sets.

4.5.2. Cluster Stores

4.5.2.1. 4store

4store is one the RDF data store used in the scalability comparison. The following graphs represent the time taken for the execution of each query on the single machine and the clusters of 4, 6 and 9 nodes, respectively.

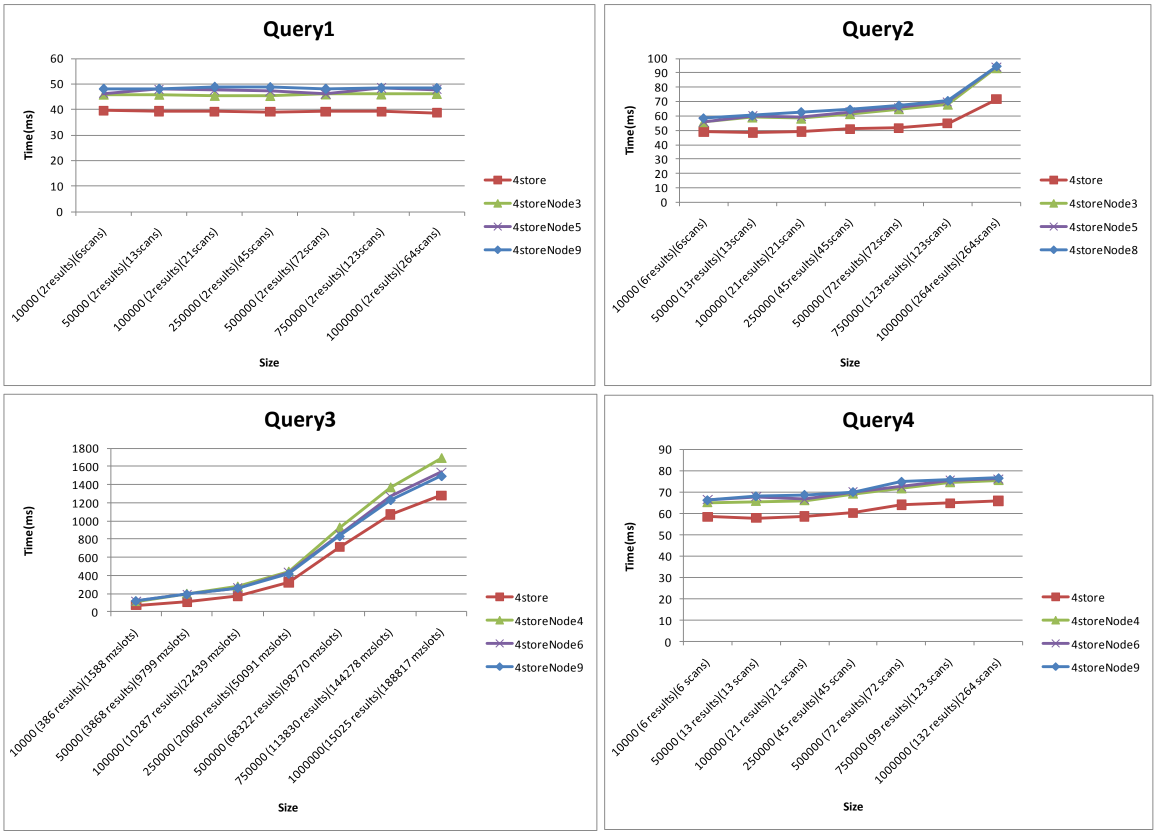


Figure 4.6: Query Response for 4store cluster

It is clear from the graphs that there is no performance improvement on the execution of the query when queried on clusters of varying sizes. One of the reasons for this trend can be due to the fact that the largest number of data set used consists of only 1Million triples.

4.5.2.2. <u>BigOWLIM</u>

BigOWLIM is another data store used for evaluating scalability. Due to the limitation of cluster writes the author could evaluate BigOWLIM only on a single machine, a 4 node cluster and 6 node clusters only till 100,000 triples.

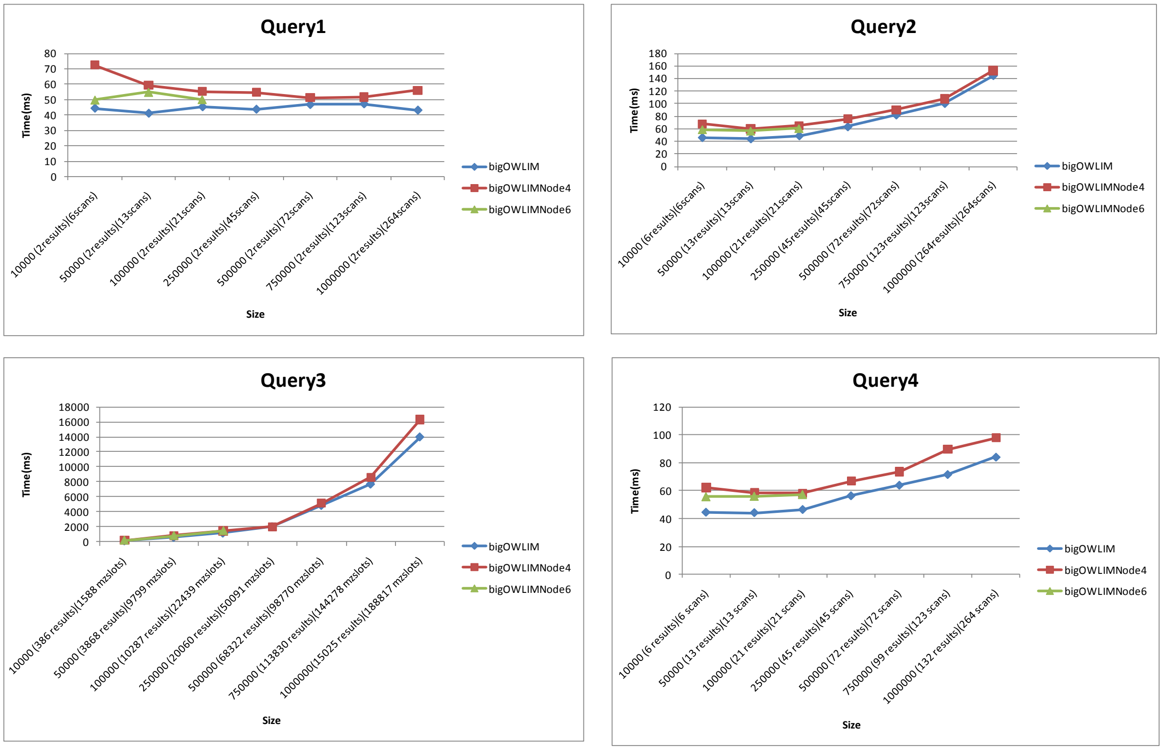


Figure 4.7: Query Response for BigOWLIM cluster

BigOWLIM cluster reaffirms the nature of query performance as seen with the 4store cluster. There is an increase in the time taken to execute queries on a 4 node cluster compared to a single machine.

4.6. <u>Analysis</u>

Based on the results it is observed that query retrieval time for Virtuoso is much smaller as compared to BigOWLIM – Sesame, Bigdata –Sesame and 4store. However it should be noted that in Query 3 Virtuoso performs poorly.

Query 3 is an important query as it is the query most likely to be encountered in the context of Proteomics data. The results further show that the 4store is the more consistent with different types of queries and output as compared to others.

To better understand the performance of data stores the author has ranked each data store for each query based on the data points. The data point with the shortest retrieval time among the four is considered to have the least weight. Based on these weightings, this is the ranking of data stores performance in single machine for each query.

Table 4.13.

Ranking of single machine data stores

Query 1	Query 2	Query 3	Query 4
1. Virtuoso (7)	1. Virtuoso (7)	1. 4store (7)	1. Virtuoso (7)
2. 4store (14)	2. 4store (17)	2. BigOWLIM(16)	2. Bigdata (19)
3. BigOWLIM (24)	3. BigOWLIM (18)	3. Bigdata (22)	3. BigOWLIM (20)
4. Bigdata (25)	4. Bigdata (28)	4. Virtuoso (25)	4. 4store (24)

Based on the weightings and rankings it can be said Virtuoso performs better than rest of the data stores.

A significant observation was made while loading the data. As the size of the data increases Virtuoso has trouble loading more than 100,000 triples at one time. As a result the input data must be split into files containing no more than 100,000 triples and loaded into the same data store individually.

For clustered data stores 4store gives better performance than BigOWLIM. Results of performance evaluation of queries on clustered RDF stores such as 4store and BigOWLIM have indicated increases in the time taken to execute the query. The limitation of the comparison is that the size of the data was limited to 1Million triples.

One of the problems the author came across while loading data in BigOWLIM was that the heap size requirement for data to be loaded on a single machine is much less than that of the 4 nodes. Thus files of size 1million triples can be loaded onto single machine BigOWLIM whereas they generated “Java™ heap space error” for the cluster.

Also creating and using clustered RDF stores for 4store was simpler than doing so for a BigOWLIM cluster.

4.7. Summary

This chapter provided the graphical representation and the analysis of the data gathered in this research.

CHAPTER 5. CONCLUSION, DISCUSSION AND FUTURE RECOMMENDATIONS

This chapter summarizes the conclusions made at the end of the study. It provides the discussion of the results obtained and the future recommendations for the research.

5.1. Conclusion

The author successfully set up and evaluated 4 different RDF stores on a single machine and 2 different clustered RDF stores for their performance and scalability. The study focused on coming up with a suitable RDF data store for storage and efficient retrieval of cancer proteomics data in the RDF format.

Proteomics data present with the author was in an mzXML format. The author successfully converted the mzXML data into OWL format using an extensible stylesheet. This data was then loaded into 4 data stores – Virtuoso, Bigdata, BigOWLIM and 4store for further evaluation.

The author then generated SPARQL queries based on the data and LUBM query specifications. These queries were executed on the 4 data stores for the seven different data sets (varying with respect to the number of triples) to get the better understanding of performance and scalability of these data stores.

Virtuoso data store was found to be most efficient for loading and querying small amounts of data. It was also observed that the performance of 4store data store was consistent with increase in the data size and query complexity.

During the study it was observed that there is no performance improvement of query evaluation with increase in the number of nodes. The study shows higher evaluation times with greater number of nodes.

The numbers of triples that are generated from an mzXML file ranges from 10,000 to 1 billion. Scalability thus is an important factor while considering storage for proteomics data. Hence, in conclusion 4store can be recommended as one of the most suitable data store for storage and evaluation of cancer proteomics data.

5.2. Discussions

The data was loaded in an incremental manner in all the data stores. First 10,000 triples were loaded, and then 40,000 triples were loaded on top of it to make it 50,000 triples. This was done since all the data stores do not accept large files consisting of more than 250,000 files in one load. So to obtain a consistent approach throughout incremental loading process was used.

The results obtained in query 3 for single machine show a different trend as compared to results of other queries. In query 3 Virtuoso’s performance is poor as compared to other queries. This can be attributed to the fact that the Virtuoso data server also supports relational data. Virtuoso’s triple store has features implemented similar to relational data store. Query 3 explores the hierarchical and exponential nature of the RDF data. Hence, Virtuoso may perform poorly for these types of queries.

Large sized data set were not loaded on clustered BigOWLIM but were loaded on the single machine. This created a bit of uncertainty about implementation of clustered BigOWLIM. But after discussion with the developers of BigOWLIM and a look at forum it was identified to be a known existing problem in BigOWLIM.

Comparing single machine with clusters did not give any improvement in query performance. This may be attributed to the fact that the maximum size of data measure was 1 million triples. The performance cluster may prove useful if the data was in order 100 billion triples. It was noted that load timings for clustered 4store were significantly smaller than the time required for loading the same data on single machine.

For comparison OWL files were generated from mzXML files. These are mass spectrometry files provided from an ongoing research. There are different softwares which generate mzXML files. Converting mzXML files into semantic language enables simpler exchange of data among researchers. Additional semantic layer enables richer queries and link to other RDF data. This comparison of data stores is valid for all the instances of mzXML ontology. Thus depending on the nature of the mzXML files data and the analysis and conclusion done in this research, one can decide which RDF data store is most suitable.

5.3. Future Recommendations

In this study the data was compared only on the basis of loading and retrieval of data. Inference power of these RDF data stores was not evaluated. The data was compared using the queries similar to the LUBM queries. One could expand this study by considering other benchmarks such as the Berlin SPARQL benchmark. (Bizer & Schulz., 2008).

The comparison between the stores was done where the data was loaded in an incremental format due to the limitations of few stores. In future the data consisting of increasing sizes can be loaded separately to check for performance.

Also only two clustered systems were considered. One could expand the study by considering clustered systems like clustered TDB, Yars2, Virtuoso, etc. Finally the study could be expanded by studying the effects of query execution of trillions of triples on a cloud based environment.

The XML Stylesheet developed to convert mzXML to OWL file does not support 4 element tags in the mzXML schema. It was tested on mzXML file generated from CompassXport. It can be extended for generic mzXML to OWL file conversion.

5.4. Summary

This chapter included the major findings in the research and addressed the problem statement stated in chapter 1. It also discussed some of the conclusions in the study and gave recommendations about the extension of the research.

LIST OF REFERENCES

LIST OF REFERENCES

Aduna Corporation. (n.d.). *OpenRDF Corporation*. Retrieved June 15, 2011, from Sesame Web site: http://www.openrdf.org/about.jsp

Andreas H., J¨urgen U., Aidan H., & Stefan D. (2007). Yars2: A federated repository for querying graph structured data from the web. *6th International Semantic Web Conference and 2nd Asian Semantic Web Conference* (pp. 211-224). Busan, South Korea: Springer-Verlag.

Alisdair O. (2009). An Investigation into Improving RDF Store Performance. Unpublished thesis, University of Southampton, Southampton, United Kingdom.

Alisdair O., Andy S., & Nick G. (2008). Clustered TDB: A clustered triple store for Jena (Tech. Rep.) Southampton, United Kingdom: University of Southampton, Electronics and Computer Science.

BigOWLIM Corporation. (n.d.). *BigOWLIM Corporation*. Retrieved June10, 2011, from BigOWLIM Corporation Web site: http://www.ontotext.com/owlim/

Chris Bizer and Andreas Schulz. Berlin SPARQL Benchmark (BSBM). http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/, 2008. Website seen on 2008-06-01.

Fredinand, M., Zirpins, C., & Trastour, D. (2004). Lifting XML schema to OWL. N. Koch, P. Fraternali, and M. Wirsing (Eds.), 4th *International Conference, ICWE*, (pp. 354-358). Munich, Germany: Springer

Guo, Y., Pan, Z., Heflin, J. (2004). An Evaluation of Knowledge Base Systems for Large OWL Datasets. (Report No. LU-CSE-04-012). Retrieved from http://swat.cse.lehigh.edu/pubs/guo04d.pdf

Harris S., Lamb N., & Shadbolt N. (2009). 4store: The design and implementation of a clustered rdf store. In A. Fokoue, Y. Guo, and T. Liebig (Eds.), proceedings of the *5th International Workshop on Scalable Semantic Web Knowledge Base Systems* (pp. 94-109). Washington DC, USA: CEUR-WS.

Kampman A., Harmelen F., & Broekstra J. (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. Proceedings of The International Semantic Web Conference.

Liu B., & Hu B. (2005). An evaluation of RDF storage systems for large data applications. In Proc. the *1st International Conference on Semantics, Knowledge and Grid* (p.59). Tokyo, Japan: IEEE Computer Society.

Ma, L., Yang, Y., Qiu, Z., Xie, G., & Pan, Y. (2006). Towards a complete owl ontology benchmark. In Y. Sure, and J. Domingue (Eds.) Proceedings of the *third European Semantic Web Conference* (pp. 124–139). New York, USA : Springer.

Erling, O., & Mikhailov I. (2007). RDF Support in the Virtuoso DBMS. In Franconi et al. (Eds), Proceedings of the 1st Conference on Social Semantic Web, Leipzig, Germany.

Prud’hommeaux, E., & Seaborne, A. (2008). SPARQL query language for RDF. Retrieved October 4, 2010, from HTTP://WWW.W3.ORG/TR/RDF-SPARQL-QUERY/

Selcuk C., Huan L., & Reshma S. (2001). Resource Description Framework: Metadata and its applications. SIGKDD Explorations Newsletter, 3(1), 6–19.

SYSTAP, LLC. Bigdata. (n.d.). *Bigdata*. Retrieved June10, 2011, from SYSTAP Web site: http://www.systap.com/bigdata.htm

Tim B., James H., & Ora L. (2001, May). The Semantic Web. Scientific American, pp.28-37.

Weaver, J., & Williams, G. T. (2009). Scalable RDF query processing on clusters and supercomputers. In A. Fokoue, Y. Guo, and T. Liebig (Eds.), proceedings of the *5th International Workshop on Scalable Semantic Web Knowledge Base Systems* (pp. 81-93). Washington DC, USA: CEUR-WS.

Weiss, C., Karras, P., & Bernstein, A. (2008). Hexastore: sextuple indexing for semantic web data management. Proceedings of the VLDB Endowment, 1, 1008 – 1019.

Yuanbo, G., Zhengxiang, P., & Jeff, H. (2005). LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semantics: Science, Services and Agents on the World Wide Web, 3(2–3), 158–182.

APPENDICES

Appendix A.

Host configurations of nodes in cluster

IP address	Host Name
10.112.42.21	heracles
10.112.42.19	perseus
10.112.42.17	pegasus
10.112.42.16	odysseus
10.112.42.15	cadmus
10.112.42.14	bellerophon
10.112.42.13	orion
10.112.42.12	theseus
10.112.42.10	achilles(Master Node)

Appendix B.

Bash scripts used to run the queries:

4store:

```
#!/bin/sh

for i in {1..100}
do
time 4s-query -f text store1 'PREFIX owl:<http://www.owl
        ontologies.com/mzxml.owl#> select * where {?s  ?p ?o}'
done
exit
```

BigOWLIM and Bigdata:

```
#!/bin/sh

for i in {1..100}
do
openrdf-sesame-2.3.2/bin/console.sh -s http://localhost:8080/openrdf-sesame
        load4 << EOF
sparql PREFIX owl:<http://www.owl-ontologies.com/mzxml.owl#> PREFIX
        rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> select ?x where {?s
        rdf:type  ?x}.
exit.
EOF
done
exit
```

Virtuoso:

```
#!/bin/sh
```

```
for i in {1..100}
do
/usr/local/virtuoso-opensource/bin/isql << EOF
sparql PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> select ?x from
        <http://load2> where {?s rdf:type ?x};
exit;
EOF
done
exit
```

Appendix C.

Commands to use the target systems:

4store:

1. 4s-backend-setup: To create database on single machine
2. 4s-backend: To start database on single machine
3. 4s-cluster-create: To create database on a cluster
4. 4s-cluster-start: To start database on cluster
5. 4s-import: To import data into data
6. 4s-query: To query the database

Bigdata:

1. openrdf-sesame-2.3.2/bin/console.sh -s http://localhost:8080/openrdf-sesame: To open sesame console
2. create bigdata. :To create Bigdata repository
   Properties to be specified:
   Repository ID [bigdata]:
   Repository title [Bigdata store]:
   Properties:
3. open <Repository ID>. : To open the repository
4. load <file to be loaded>: To load the data into the repository
5. SPARQL <query>: To query the repository
6. close <Repository ID>. : To close the repository

BigOWLIM:

1. openrdf-sesame-2.3.2/bin/console.sh -s http://localhost:8080/openrdf-sesame: To open sesame console
2. create Bigdata. :To create BigOWLIM repository
   Properties to be specified:
   Repository ID [BigOWLIMTest]:
   Repository title [BigOWLIM Test store]:
   Set of rules [owl-horst-optimized]:
   Storage folder [owlimTest-storage]:
   entity index size [200000]:
   imports(';' delimited):
   defaultNS(';' delimited):
   open <Repository ID>. : To open the repository
3. create cluster: To create BigOWLIM cluster
   Repository ID [cluster]:
   Repository description [BigOWLIM Replication Cluster master node]:
4. load <file to be loaded>: To load the data into the repository
5. SPARQL <query>: To query the repository
6. close <Repository ID>. : To close the repository

Virtuoso:

1. virtuoso-opensource/bin/virtuoso-t -f &: To start virtuoso server
2. virtuoso-opensource/bin/isql: To start console
3. DB.DBA.RDF_LOAD_RDFXML (file_to_string ('<location of file>'),'','<name to be loaded>'); : To load the data
4. SPARQL <query> : To query the database
5. Exit : To exit from the console

Appendix D.

To create cluster setup:

4store:

1. Edit /etc/4s-cluster to write the hostnames of the nodes of the cluster.
2. Use 4s-cluster-create & 4s-cluster-destroy to create and destroy the cluster.

BigOWLIM:

1. In sesame console use “create cluster” command.
2. In /etc/tomcat6/tomcat6.conf added these lines to start remote port:
   ```
   JAVA_OPTS="-Dcom.sun.management.jmxremote.port=8089 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun/management.jmxremote.ssl=false"
   ```
3. Start a JMX client jconsole: jconsole localhost:8089
4. Go to mbeans tab and press on the replication cluster option to add worker nodes to the master node using the addClusterNode operation.
5. Configure the master node to be writeable.

Appendix E.

Stylesheet to convert mzXML into OWL format.

```
<xsl:stylesheet version="2.0"
             xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
             xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:x="http://sashimi.sourceforge.net/schema_revision/mzXML_3.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://sashimi.sourceforge.net/schema_revision/mzXML_3.0
http://sashimi.sourceforge.net/schema_revision/mzXML_3.0/mzXML_idx_3.0.xsd"
>
        <xsl:output media-type="text/xml" version="1.0" encoding="UTF-8"
indent="yes" use-character-maps="owl"/>

        <xsl:strip-space elements="*"/>

        <xsl:character-map name="owl">
                <xsl:output-character character="&amp;" string="&amp;"/>
        </xsl:character-map>


<xsl:template match="/">
             <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
             xmlns:rdfs="http://www.w3.org/2001/01/rdf-schema#"
             xmlns:owl="http://www.w3.org/2002/07/owl#"
             xmlns:mz="http://www.owl-ontologies.com/Ontology1267448365.owl"
             xml:base="http://www.owl-ontologies.com/Ontology1267448365.owl">
                    <owl:ontology rdf:about="">
                           <owl:imports/>
                    </owl:ontology>
                    <xsl:apply-templates/>
             </rdf:RDF>
       </xsl:template>


<xsl:variable name="number">0</xsl:variable>

<xsl:template match="x:mzXML">


       <xsl:for-each select="//x:msRun">
<msRun rdf:ID="msRun_0">
       <_scanCount rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
<xsl:value-of select ="@scanCount"> </xsl:value-of>
                    </_scanCount>

       <_startTime rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="@startTime"> </xsl:value-of>
                    </_startTime>

       <_endTime rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="@endTime"> </xsl:value-of>
                    </_endTime>


       <xsl:for-each select="//x:parentFile">
```

```
<parentFileslot>
<parentFile rdf:ID="{generate-id()}">
<_fileName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="@fileName"> </xsl:value-of>
                    </_fileName>

<_fileType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="@fileType"> </xsl:value-of>
                    </_fileType>

<_fileSha1 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="@fileSha1"> </xsl:value-of>
                    </_fileSha1>
</parentFile>
</parentFileslot>
</xsl:for-each>

<xsl:for-each select="//x:msInstrument">
<msInstrumentslot>

       <msInstrument rdf:ID="{generate-id()}">
       <xsl:for-each select="//x:msInstrument/x:msManufacturer">
<msManufacturerslot>
             <msManufacturer rdf:ID="{generate-id()}">
                    <_category
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@category"> </xsl:value-of>
                    </_category>

                    <_value
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@value"> </xsl:value-of>
                    </_value>
             </msManufacturer>
</msManufacturerslot>
       </xsl:for-each>


       <xsl:for-each select="//x:msInstrument/x:msModel">

<msModelslot>
             <msModel rdf:ID="{generate-id()}">
                    <_category
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@category"> </xsl:value-of>
                    </_category>

                    <_value
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@value"> </xsl:value-of>
                    </_value>
             </msModel>
</msModelslot>


       </xsl:for-each>


       <xsl:for-each select="//x:msInstrument/x:msIonisation">
```

```
<msIonisationslot>
              <msIonisation rdf:ID="{generate-id()}">
                     <_category
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select ="@category"> </xsl:value-of>
                     </_category>

                     <_value
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select ="@value"> </xsl:value-of>
                     </_value>
              </msIonisation>
</msIonisationslot>

       </xsl:for-each>


       <xsl:for-each select="//x:msInstrument/x:msMassAnalyzer">
<msMassAnalyzerslot>
              <msMassAnalyzer rdf:ID="{generate-id()}">
                     <_category
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select ="@category"> </xsl:value-of>
                     </_category>

                     <_value
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select ="@value"> </xsl:value-of>
                     </_value>
              </msMassAnalyzer>
</msMassAnalyzerslot>

       </xsl:for-each>

       <xsl:for-each select="//x:msInstrument/x:software">
<softwareslot>
              <software rdf:ID="{generate-id()}">
                     <_type
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select ="@type"> </xsl:value-of>
                     </_type>

                     <_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select ="@name"> </xsl:value-of>
                     </_name>

                     <_version
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select ="@version"> </xsl:value-of>
                     </_version>

              </software>
</softwareslot>

       </xsl:for-each>
       </msInstrument>
</msInstrumentslot>

</xsl:for-each>
```

```
<xsl:for-each select="//x:dataProcessing">
<dataProcessingslot>

        <dataProcessing rdf:ID="{generate-id()}">
        <_centroided rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@centroided"> </xsl:value-of>
        </_centroided>

        <xsl:for-each select="//x:dataProcessing/x:software">
<softwareslot>

                <software rdf:ID="{generate-id()}">
                        <_type
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@type"> </xsl:value-of>
                        </_type>

                        <_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@name"> </xsl:value-of>
                        </_name>

                        <_version
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@version"> </xsl:value-of>
                        </_version>

                </software>
</softwareslot>

        </xsl:for-each>
        </dataProcessing>
</dataProcessingslot>

</xsl:for-each>




<xsl:for-each select="//x:msRun/x:scan">
<xsl:variable name="number" select="$number + 1"/>
<scanslot>
        <scan rdf:ID="{generate-id()}">

        <_num rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@num"> </xsl:value-of>
                        </_num>

        <_peaksCount rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@peaksCount"> </xsl:value-of>
                        </_peaksCount>

        <_polarity rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@polarity"> </xsl:value-of>
                        </_polarity>

        <_scanType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                        <xsl:value-of select ="@scanType"> </xsl:value-of>
                        </_scanType>
```

```
       <_filterLine rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@filterLine"> </xsl:value-of>
                    </_filterLine>

       <_retentionTime rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@retentionTime"> </xsl:value-of>
                    </_retentionTime>

       <_lowMz rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@lowMz"> </xsl:value-of>
                    </_lowMz>

       <_highMz rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@highMz"> </xsl:value-of>
                    </_highMz>

       <_basePeakMz rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@basePeakMz"> </xsl:value-of>
                    </_basePeakMz>

       <_basePeakIntensity
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@basePeakIntensity"> </xsl:value-of>
                    </_basePeakIntensity>

       <_totIonCurrent rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@totIonCurrent"> </xsl:value-of>
                    </_totIonCurrent>


<xsl:for-each select="child::x:peaks">
<peaksslot>
       <peaks rdf:ID="{generate-id()}">

       <_precision rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@precision"> </xsl:value-of>
                    </_precision>

       <_byteOrder rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@byteOrder"> </xsl:value-of>
                    </_byteOrder>

<_pairOrder rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                    <xsl:value-of select ="@pairOrder"> </xsl:value-of>
                    </_pairOrder>

<text rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="self::x:peaks"> </xsl:value-of>
       </text>

       </peaks>
</peaksslot>

       </xsl:for-each>

<xsl:for-each select="//x:index">

<indexslot>
<index rdf:ID="{generate-id()}">
```

```
       <xsl:for-each select="x:offset">
<offsetslot>
<offset rdf:ID="{generate-id()}">


<_id rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="@id"> </xsl:value-of>
       </_id>

<text rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="self::x:offset"> </xsl:value-of>
       </text>

</offset>
</offsetslot>

</xsl:for-each>
</index>
</indexslot>

       </xsl:for-each>



       <xsl:for-each select="//x:indexOffset">
<indexOffsetslot>
<indexOffset rdf:ID="{generate-id()}">
<text rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="//x:indexOffset"> </xsl:value-of>
       </text>
</indexOffset>
</indexOffsetslot>

       </xsl:for-each>

       <xsl:for-each select="//x:sha1">

<sha1slot>

<sha1 rdf:ID="{generate-id()}">
<text rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
<xsl:value-of select ="//x:sha1"> </xsl:value-of>
       </text>
</sha1>
</sha1slot>

       </xsl:for-each>


</msRun>
       </xsl:for-each>

</xsl:template>


</xsl:stylesheet>
```