Analysis of Semantic Web Databases

By

Anila Sahar Butt

2007-NUST-MS-PhD IT-22

Supervisor

Dr. Sharifullah Khan

A thesis submitted in partial fulfillment of the requirements for the degree of

Masters of Science in Information Technology (MSIT)

In

NUST School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST), Islamabad, Pakistan

(September 2010)

APPROVAL

It is certified that the contents and form of thesis entitled **"Analysis of Semantic Web Databases"** submitted by **Anila Sahar Butt** have been found satisfactory for the requirement of the degree.

Advisor: _____Dr. Sharifullah Khan______

Signature: ___________________________

Date: ___________________________

Committee Member 1: _Dr. Khalid Latif___

Signature_____________________________

Date:________________________________

Committee Member 2: _Dr. Zia ul Qayyum_

Signature _____________________________

Date: ________________________________

Committee Member 3: _Mr. Owais Malik__

Signature _____________________________

Date: ________________________________

In the Name of Allah, the Most Gracious, the Most Merciful

Say: "And He has subjected to you the night and the day, the sun and the moon; and the stars are subjected by His Command. Surely, in this are proofs for people who understand. And whatsoever He has created for you on this earth of varying colors. Verily! In this is a sign for people who remember".

-----------Holy Quran

To my Parents, for their love and support

CERTIFICATE OF ORIGINALITY

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Anila Sahar Butt

Signature: ___________________

ACKNOWLEDGEMENTS

First and foremost, I am immensely thankful to Almighty Allah for letting me pursue and fulfill my dreams. Nothing could have been possible without His blessings.

More than anything, I would like to thank my mom and dad who have endured me through this period of research that has been extremely demanding and challenging. Thanks to my mom for always being so motherly in everything she does. Especial thanks to my dad for not letting me worry about anything but work. These last years of research could not have been possible without the unending support from my parents. They have always supported and encouraged me to do my best in all matters of life. To them I dedicate this thesis.

I would also like to thank my Grandmother, brothers and sister for all the care and love they showered on me during all these years. I am immensely grateful to Ali Naqi for always being appreciative of my hard work, for his unending support and for encouraging me during the hardest of times.

Finally, this thesis would not have been possible without the expert guidance of my esteemed advisor, Dr. Sharifullah Khan, who has been a great source of inspiration for me during these years of research. Not only has he been readily available for me, as he so generously is for all of his students, but he always responded to any queries that I might have, read through my draft copies, listened to my moaning and complaining and supported me every step of the way.

My heartfelt thanks to everyone at DELSA Lab, my committee members and all others who contributed in any way towards the successful completion of this thesis.

Anila Sahar Butt

Table of Contents

LIST OF ABBREVIATIONS

API	Application Programming Interface
C_X	Cost of X
Jena$_M$	Jena (In-Memory Store)
LT	Load Time
MM	Main Memory
OWL	Ontology Web Language
OWL-DL	Web Ontology Language – Description Logic
QRT	Query Response Time
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RS	Repository Size
Sesame$_M$	Sesame (In-Memory Store)
Sesame$_N$	Sesame (Native Store)
Sesame$_{RDB}$	Sesame (RDBMS Store)
URI	Uniform (Universal) Resource Identifier
URL	Uniform (Universal) Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

LIST OF FIGURES

LIST OF TABLES

Abstract

The popularity of Semantic Web has given rise to the development of Semantic Web databases with improved performance. Benchmarks are being performed to validate performance claim made by developers of Semantic Web databases. However, detailed information regarding the strengths and shortcomings of these databases is limited due to the fact that the existing benchmarks provide little depth in scalability analysis. They measure the Semantic Web databases' performance in terms of time and do not cover resource utilization during data manipulation operations. The research literature available on Semantic Web databases does not provide details of their internal architecture. In this research, we aim to evaluate the existing Semantic Web databases to discover their comparative behavior and scalability trends for a newly proposed evaluation methodology, and to analyze their architectures particularly with respect to their storage schemas and access methods.

To cope with the deficiencies of existing evaluation methodologies, we have proposed a new evaluation methodology to perform comparative analysis and scalability performance study of Semantic Web databases. Our evaluation methodology comprises test cases for the data access methods and query optimization techniques to analyze the performance of Semantic Web databases. We defined new metrics for query cost estimation. As a part of this work, we also evaluated the performance of seven prominent open-source Semantic Web databases. These Semantic Web databases were evaluated on our proposed evaluation methodology using Barton Library dataset.

Based upon our experiments and proposed methodology, we highlighted the key strengths and weaknesses of these Semantic Web databases, and discovered their scalability behavior. Storage schemas and access mechanism of the Semantic Web databases are identified in this thesis. We conclude that overall native Semantic Web databases perform better than others i.e. in-memory and non-memory non native Semantic Web databases. We also conclude that the requirements of in-memory stores for time and resource usage do not increase as rapidly as in other two categories of Semantic Web databases. The evaluation results show that the proposed evaluation methodology provides better scalability behavior and performance estimation of Semantic Web database than the existing evaluation studies.

INTRODUCTION

This chapter introduces the research work that has been taken in this thesis. First it describes the motivation for research then it goes on describing our research contribution that includes problem definition, goals and objectives and the approach to achieve these goals.

1.1. Motivation

Semantic Web provides interoperability in integrating information from multiple resources and makes computers intelligent to work on their own behalf for unseen situations [1]. Semantic Web databases are designed to hold massive Semantic Web data in such a manner that the information they encode can be retrieved efficiently. Many stores have been introduced with the popularity and increased used of Semantic Web applications.

The main challenge of Semantic Web databases is to execute different data management tasks in a user interactive time by utilizing few system resources. Moreover, scalability of these databases is a major issue because when procuring or designing a Semantic Web database, users and designers are often requires that it must be scalable because of number of triples increases rapidly to model a piece of information in Semantic Web. In view of vast research on efficient storage and retrieval of Semantic Web data, it is important to survey and evaluate the existing Semantic Web databases as developers of each one claims to perform the best. While a performance evaluation of Semantic Web databases facilitates the community in verifying developers' performance claim of Semantic Web databases and in assessing the performance of a newly created or updated Semantic Web database against existing once, more importantly they should reveal the strengths and shortcomings of each existing Semantic Web database.

While comparative evaluation facilitates the users to select from a library of Semantic Web databases, it is important for the developers of new Semantic Web database to know about the architectural details of existing Semantic Web databases. The Semantic Web database developers provide the API's along with the user documentation but they do not provide their internal architecture and/or working. It is worth describing the storage layouts and data retrieval mechanism used by existing Semantic Web databases.

1.2. Research Contributions

In this research we aim to evaluate the existing Semantic Web databases to learn about their *comparative behavior* and *scalability trends* for our *proposed evaluation methodology* and to analyze their *storage schemas.*

Research objectives set to follow above stated problem statement are given below. After that we present a summary of our contribution to achieve these objects.

- To design a new evaluation methodology for Semantic Web databases in order to overcome the limitations of existing methodologies.
- To provide detailed analysis of a dataset that best represents the Semantic Web data in order to make it understandable for testing Semantic Web databases.
- To quantify and compare the performance of some of the prominent Semantic Web databases in order to provide better understanding of their key strengths and weaknesses.
- To investigate the scalability trends of these Semantic Web databases for analyzing their scalability (time scalability & space-time scalability).
- To document the Semantic Web databases storage layouts for helping the designers of new Semantic Web database

In light of these objectives we initially propose a new evaluation methodology that comprises of (1) *test cases* and (2) *performance and scalability metrics*. Our test cases are classified on the basis of CRUD operations (i.e. Create, Read, Update and Delete). *Read* test cases check different parameters that affect the performance of a Semantic Web database; it includes selectivity estimation, result size, query complexity, indexing mechanism and storage organization. Moreover, we propose the potential performance and scalability metrics that capture different aspects of evaluation process and provide better insight of the Semantic Web data by providing detailed results than present literature about Semantic Web databases' benchmarks. We introduce two core performance metrics i.e. "Resource Utilization" and "Success Ratio", and one derived metric i.e. "Cumulative Query Performance".

In order to provide detailed analysis of any real Semantic Web dataset, we made tradeoff on dataset size and dataset structure and selected Barton Library dataset for our evaluation. We performed detailed analysis of Barton dataset and designed a data model for its better understanding, because it is frequently used in Semantic Web analysis. Secondly we provided the Barton datasets characteristics i.e. its number of triples, number of nodes, type of instances, total unique properties, single-valued properties and multi-valued properties. In addition to this, we cleaned this dataset to remove illegal URIs.

For quantifying and comparing the performance of prominent Semantic Web databases, we selected several popular Semantic Web databases. Semantic Web databases compared in this work are Jena_{M} [57], Sesame_{M} [60], TDB [59], SDB [58], Sesame_{N} [60], Sesame_{RDB} [60], and AllegroGraph [37]. These Semantic Web databases are selected because all these are open source java APIs and they all together represent all three categories of Semantic Web databases. Moreover, most of these are frequently used in performance benchmarking in the Semantic Web database research literature [19], [22], and [24]-[28]. We compared all these Semantic Web databases and on the basis of which we provide the strengths and weakness of each store. The comparative analysis of the Semantic Web database concludes read and write optimized store of each category (i.e. in-memory, native, and non-memory non-native) with respect to time and resource utilization.

Based upon results obtained from our experiments for comparative evaluation, we performed scalability analysis to check the scalability behavior of each store. Scalability analysis presents a clear idea that how the time and resource utilization of each store increases with the increase number of triples to manage. We concluded our scalability analysis by presenting the comparative scalability behavior of these stores.

For describing the Semantic Web databases' storage layouts we surveyed the past approaches used by selected Semantic Web databases and presented their common approaches to store and retrieve the data e.g. id-based vs. value-bases approach, schema oblivious vs. schema aware approach, prefix compression, type and number of indexes, value-to-Id and Id-to-value mapping techniques. This survey helps the designers of new Semantic Web databases and clarifies the performance level of each particular storage layout.

1.3. Thesis Organization

This thesis is organized as follows:

- Chapter 2 provides pre-requisite knowledge required to understand the research, presented in this thesis. It describes Semantic Web and its languages followed by Semantic Web databases. A brief comparative analysis between other database systems and Semantic Web databases is presented to establish understanding why Semantic Web databases are necessary. By the end of this chapter, some eminent benchmarks are discussed and analyzed critically to rationalize the research work.

- Chapter 3 gives the detail of Semantic Web databases architecture that we have evaluated in our research. It describes the storage schemas used for storage and management of the triples of each Semantic Web database. It goes on to describe the storage layouts of In-memory, native and non-memory non-native categories of Semantic Web databases.
- Chapter 4 details the Semantic Web databases evaluation framework in detail along with introduction to some prevalent Semantic Web databases that are being evaluated. Moreover, it presents the logical schema of Barton dataset in detail to evaluate previously mentioned Semantic Web databases. The last section provides details of our proposed evaluation methodology that includes test cases description and proposed performance and scalability metrics.
- Chapter 5 presents the detailed results and analysis obtained during research. It describes the test configurations, both for machine and Semantic Web databases. It then provides a detailed C*omparative Evaluation* of tested Semantic Web databases, on the basis of which strengths or weaknesses of each store are discussed. In addition to this, *Scalability Analysis* for these stores is provided.
- Chapter 6 concludes with the summary of points made and describes the directions that have been decided upon for future research with justification and references to a set of work packages that would be required to complete the research.

BACK GROUND AND RESEARCH MOTIVATION

This chapter is divided into two parts. The first part focuses on background literature to facilitate the understanding of the context of this research while the second part presents literature survey of related work to rationalize the research work being done.

2.1. Semantic Web

The *Semantic Web (Web3.0)* is aimed at providing a common framework for data sharing and reuse across Semantic Web applications, enterprises and communities [1][5][39][40]. The current web (Web2.0) is the web of documents where the information has been tailored for human understanding and not for computers. As shown in Figure 1, the documents are related by unlabeled links. Computers can realize that a document is related to another document, but cannot exactly understand the nature of the relation that exists between documents and their contents. All these resources are missing semantic information and homogenous for computers [40].

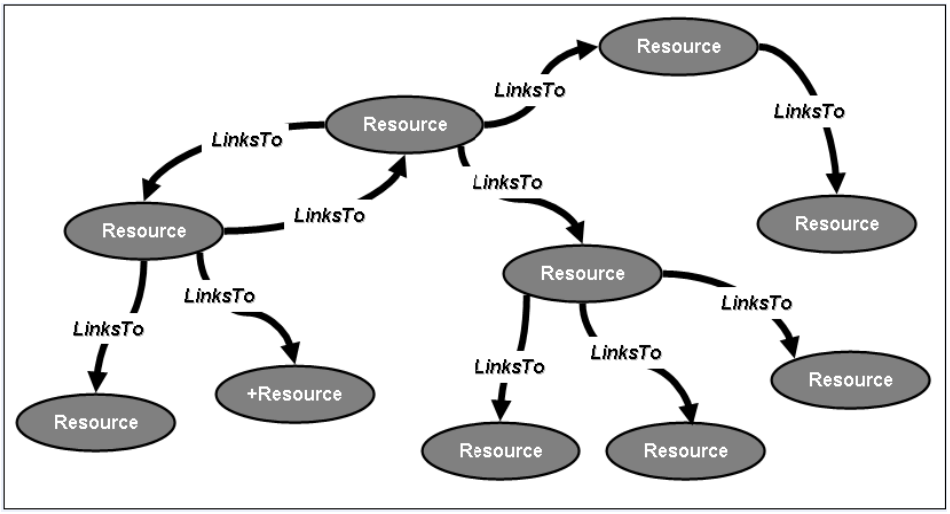


Figure 1: World Wide Web

Semantic Web is defined as an extension of the current web where information is presented with well defined meanings. It enables computers and people to work in cooperation [1][39][41]. Semantic Web augments but do not replace the current web. Its design goal is to enable computers to behave or think like humans in order to make the World Wide Web (WWW) data machine-processable. In Semantic Web resources are named and links between them are labeled as shown in Figure 2. Semantic Web is the web of things rather than being the web of documents. It defines the relationship among things and properties of these things. So that computers can easily identify the nature and relationships of things through their properties. Consequently Semantic Web helps in retrieval of knowledge rather than data from web [41].

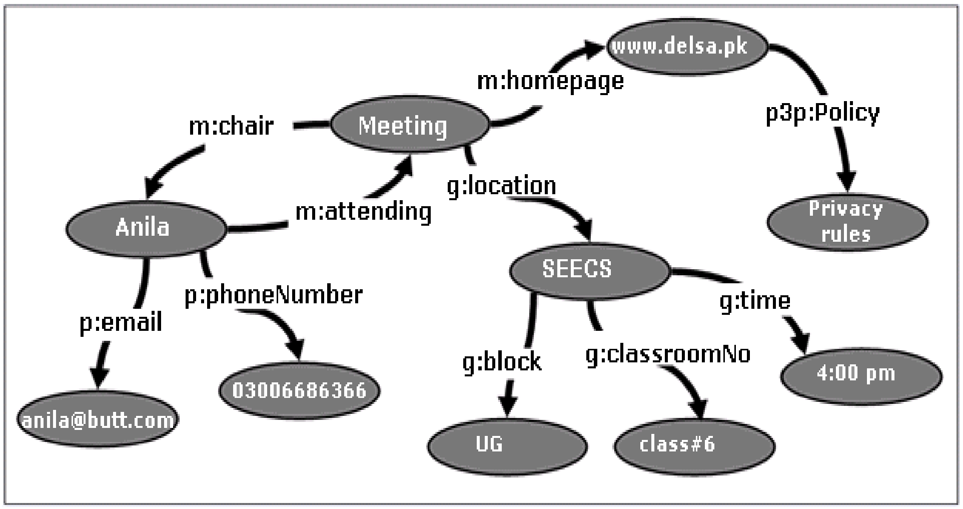


Figure 2: Semantic Web

The difference between Semantic Web and current web can be explained with an example of cooking analogy as shown in Figure 3. Assume a website or web page is a big cooking pot where a user puts his information contents. A cook can put rice in one pot and tag it *RICE* and put chicken in another pot and tag it *CHICKEN*. Similarly web users put their information contents on web pages in the form of documents and tag them. In the current web, computers understand only these tags and cannot figure out that the rice is a grain and full of carbohydrates. They also do not know that the cook has another pot containing chicken and is cooking chicken rice. This type of judgment necessitates context analysis and the capability to glance the whole scenario, i.e. to identify what is in each pot and to develop the understanding that the cook is making foodstuff. In the context of web, computers must know that what types of contents are on each page and how they are related to each other. This is the objective of the Semantic Web. Semantic Web can be visualized as "*the web of meaning*" or " *the contextual web*" [1].

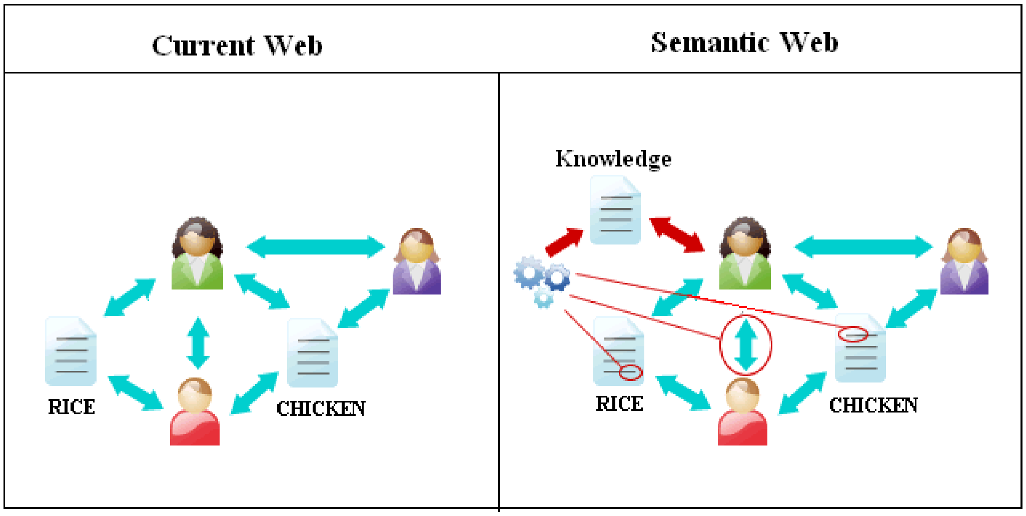


Figure 3: Semantic Web vs. Current Web

The key idea behind Semantic Web is to provide sufficient structure around the data to transform it into information, so that it can be worked upon to extract knowledge [40][41]. There are *universal information formats* to provide structure to data as well as metadata on Semantic Web. These formats are known as Semantic Web languages e.g. *Resource Description Framework* (RDF), *Resource Description Framework Schema* (RDFS) and *Web Ontology Language* (OWL). Semantic Web languages are further explained in section 2.2.

Semantic Web data must be stored in some repository for knowledge extraction. Specialized databases are available to work with Semantic Web data because of its specialized structure. These databases are known as *Semantic Web databases* that are further explained in section 2.3.

Different query languages are supported by different Semantic Web databases. One of the most common and evolved query languages is SPARQL [35]. Semantic query languages are further explained in section 2.4.

2.2. Semantic Web Languages

Semantic Web has its own special languages to introduce semantics into data. These are also named *Semantic Web standards*. These standards make data integration and interoperability possible on Semantic Web. The current Semantic Web standards consist of two layers *data layer* and *ontology layer*. Both these layers are built on top of URI, XML and XML namespaces that provide the foundations for these Semantic Web layers. An overview of these layers is given in Figure 4.

- **Ontology Layer:** Ontology is the representation of application domain information modeled though some Semantic Web language [42]. Ontology layer of Semantic Web standards contains those Semantic Web languages that are used to model the application domain semantics through ontology. RDFS and OWL are ontology layer modeling languages.

- **Data Layer:** Data layer of Semantic Web standards contains those languages that are used to define the resources (data) of a particular application domain that has been modeled through ontology layer. Most widely used data layer standard/language is RDF.

In the subsequent subsections we will explain these standards in detail.

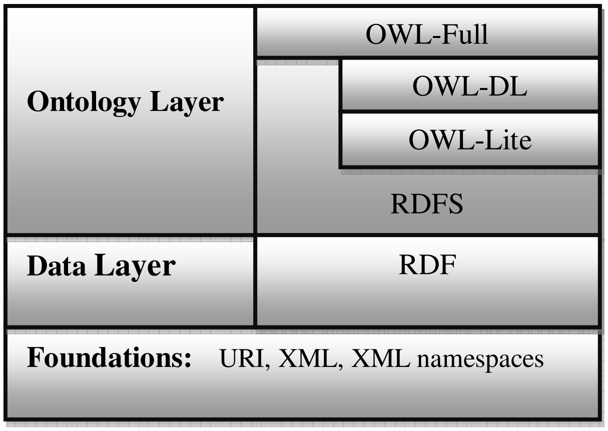


Figure 4: Semantic Web Standards

2.2.1. RDF: A Data Modeling Language

Resource Description Framework (RDF) is an underpinning language for information representation in Semantic Web [49][3][4][5]. Its three fundamental concepts are:

1. Resources
2. Properties
3. Statements

Resources are the '*things*' that are being described by the RDF expressions. A resource may be a part of web page e.g. an HTML element within a document, a whole web page e.g. an HTML document or a collection of web pages e.g. a complete website. Each resource is assigned a unique identifier that is called *Universal Resource Identifier (URI)*.

Properties are special kind of resources. A property describes the relationship between two resources.

Statement is composed of a resource along with its associated property and value for that property as shown in figure below.

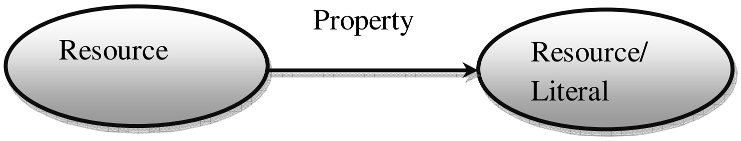


Figure 5: RDF Statement

An RDF statement is represented in a triple format also called '*Triple*'. Triples are composed of three different parts that are called *subject*, *predicate* and *object* respectively. In an RDF statement, subjects and predicates are guaranteed to be resources but object or value part may be a resource or a literal, where literals are atomic values i.e. without URIs. A detail study of RDF can be found at [49][3][4]**.** Consider an example of a simple sentence "Anila Sahar is a member of DELSA". This sentence in terms of RDF statement has three parts.

Subject: Anila Sahar (Resource)

Predicate: memberOf (Property)

Object: DELSA (Resource)

Pictorial representation of this example is shown in Figure 6.

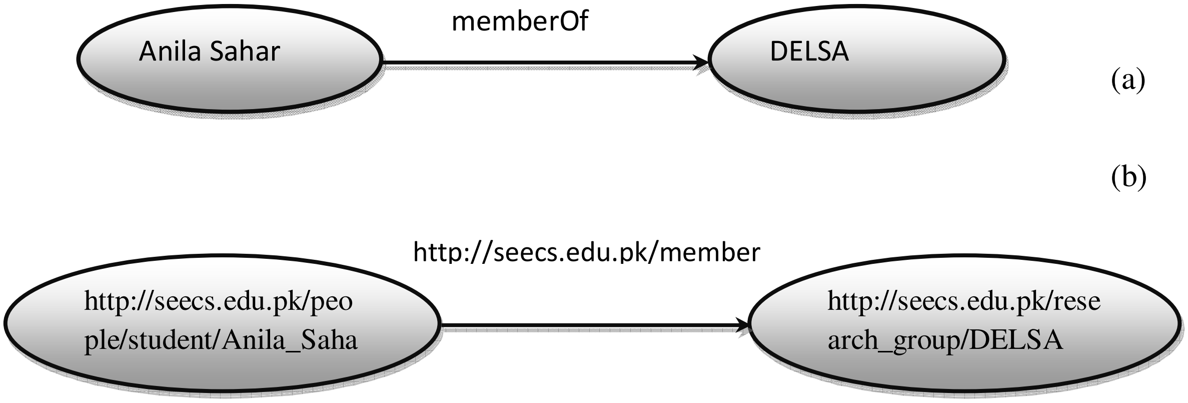


Figure 6: Triple Concept

Each resource is represented with its URI. These URIs can be replaced with prefixes as shown below

Prefix	URIs
Seecs	http://seecs.edu.pk/

Table 1: URI with Prefix

So above triple pattern replaced with prefix can be as shown in Figure 7

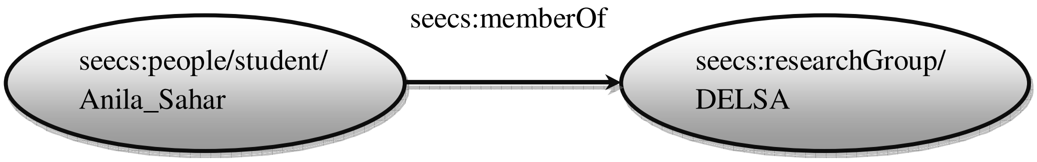


Figure 7: Triple with Replaced URIs

A set of RDF triples is known as an *RDF model* as shown in Figure 8. An RDF model is basically a directed graph of information that offers great deal of power and flexibility. There is no limit to extend this graph structure. It tenders the ability to represent different concepts and their relationships semantically into an unlimited large graph of information.

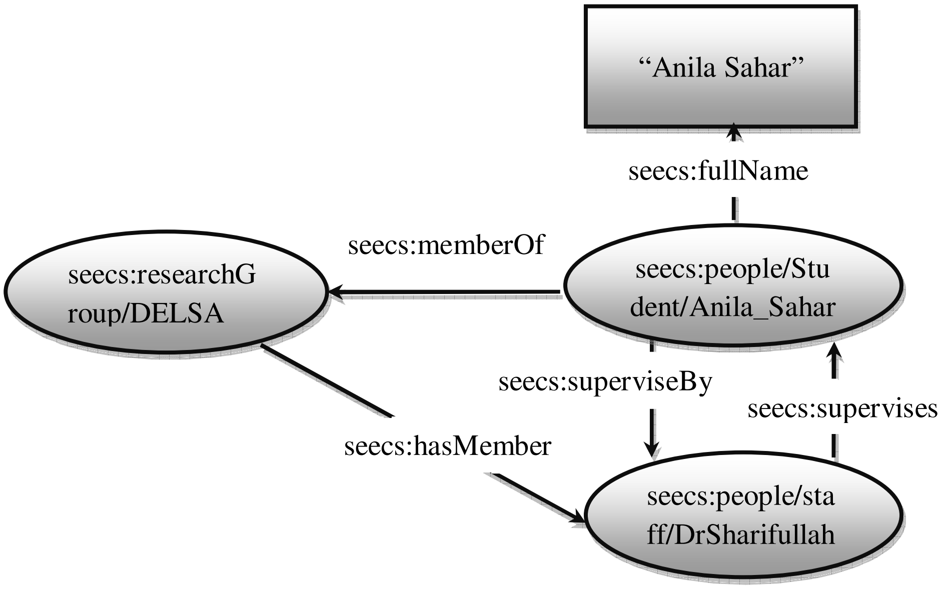

"Anila Sahar"
seecs:fullName
seecs:memberOf
seecs:researchGroup/DELSA
seecs:people/Student/Anila_Sahar
seecs:superviseBy
seecs:supervises
seecs:hasMember
seecs:people/staff/DrSharifullah

Figure 8: RDF Graph Model

2.2.2. RDFS & OWL: Domain Modeling Languages

Resource Description Framework Schema (RDFS) and *Web Ontology Language* (OWL) are the Semantic Web languages that model an application domain to offer the facility to perform interoperability and reasoning on the Semantic Web.

RDFS basically augments the RDF by adding some basis constructs [49]. These constructs include classes, properties, class hierarchies, property hierarchies, domain and range. Vocabulary used to model these extended constructs are rdfs:Class, rdfs:Property, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range. A set of individuals that share same properties belong to a single class e.g., *"Student"* and *"Research group"*. All those individuals that belong to a particular class are referred as instances of that class e.g. "Anila Sahar" is an instance of class "Student". These constructs allow statements about something's type by making new statements from existing statements such as 'Anila Sahar is a type of student' and with an additional statement that 'Student is a subclass of people', one can infer that Anila Sahar is a type of people as shown in Figure 7. Domain and range of a property describe the class of things that can be declared as a subject or an object part of a property e.g., Property hasMemeber can have an instance of class research group as a subject and an instance of class people as an object in an RDF statement as shown in Figure 9.

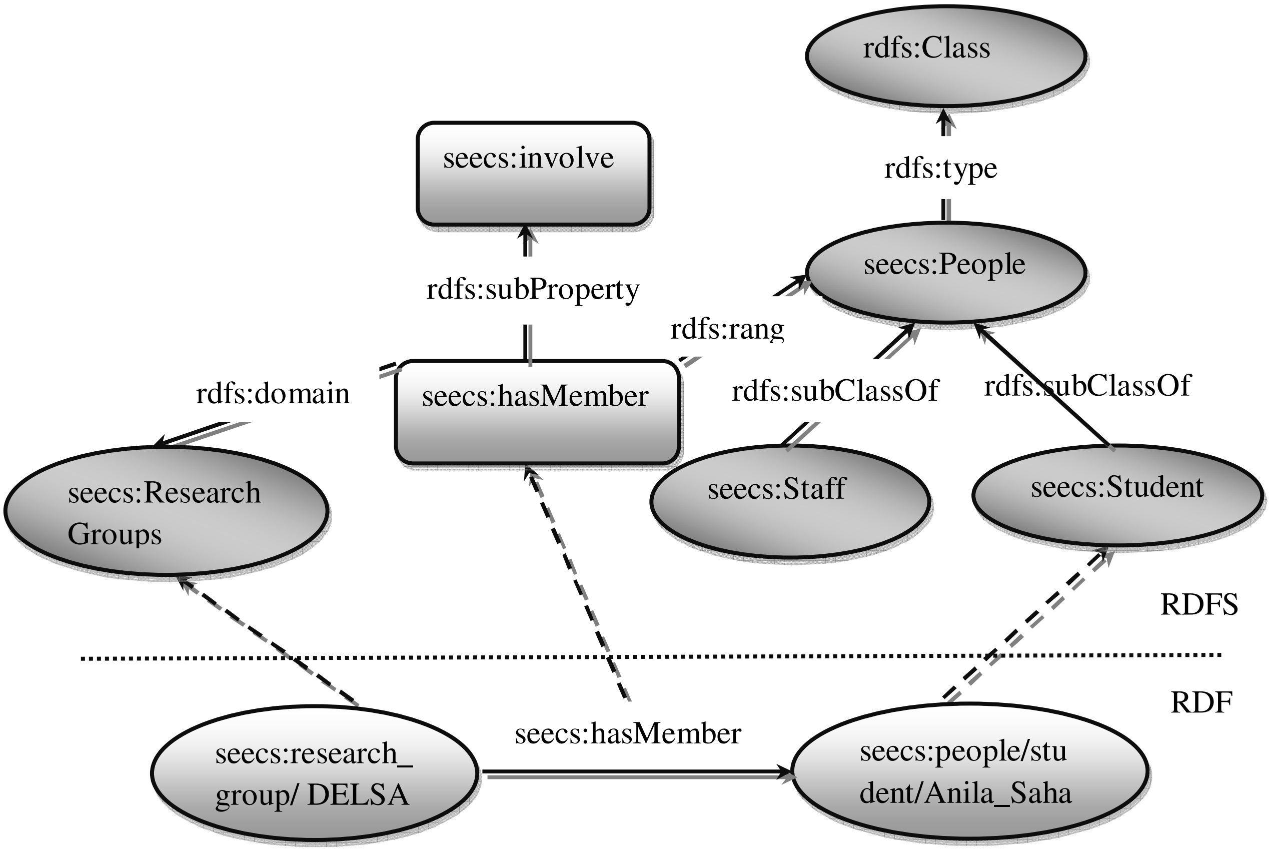


Figure 9: RDF layer vs. RDFS layer

As mentioned above, RDFS defines semantics of the application domain but *Web Ontology Working Group* of W3C identified a number of characteristics for ontology on the web which would require much more expressiveness than that provided by RDF and RDF Schema [6]. Some of the identified limitations of RDFS by W3C are given below.

a. **Local scope of a class:** In RDFS rdfs:range defines that a class is a range of a property. We cannot declare range restrictions that apply to some subclasses only and not to all the subclasses of a class e.g. using rdfs:range we can define a property “hasMember” for research groups as “Research groups have only people as its members” but we cannot say that DELSA (a subclass of research group) can have only staff as its members while other research groups may have student members as well.

b. **Disjointness of classes:** Some application domains necessitate the declaration that classes are disjoint, e.g. undergraduates and postgraduates are two disjoint classes in university domain. RDFS is unable to define disjointness of classes.

c. **Boolean combination of classes:** Some application domains are required to built new classes by combining existing classes using union, intersection and complement e.g. class students is a union of two disjoint classes, undergraduate class and postgraduate class. RDFS does not provide the facility of creating new classes from existing once.

d. **Cardinality restrictions:** An ontology developer may want to restrict the number of instances that a particular property can have as its range e.g. There should be five members at minimum and twenty members at maximum in a research group. This is called cardinality restriction. RDFS do not provide the feature of cardinality restriction.

e. **Special characteristics of properties:** Some special characteristics of properties may require to model an application domain, e.g. transitive property (like ‘older than’), unique property (like ‘supervisedBy’) and inverse property (like ‘supervisedBy’ and ‘supervises’) while modeling a domain. This feature is also missing in RDFS.

Web Ontology Language (OWL) extends the RDF schema in the sense that OWL uses RDFS vocabulary like rdfs:Class, rdfs:subClassOf and add some language primitives that support more expressiveness as highlighted above. Extended constructs of OWL includes owl class, property (object property and data type properties), property restrictions, special properties, boolean combinations, enumerations, instance, data type and versioning information (For further detail see [7]). As we have seen that OWL provides a higher level of expressiveness than RDFS but the richer the language is, the more inefficient reasoning support becomes. We need tradeoff on the selection of a language supported by reasonably efficient reasoners and a language that can express large classes of ontologies and knowledge. Depending upon a tradeoff between language expressiveness and efficient reasoning, OWL has three variants i.e. OWL Full, OWL Lite and OWL DL. All these variants use RDF for their syntax.

a. **OWL Full:** The complete OWL language is called OWL Full. It includes all OWL constructs and combination of these constructs in arbitrary way with RDF and RDFS. It is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no reasoning guarantee.

b. **OWL Lite:** A sublanguage of OWL which provides constructs to users for classification hierarchy and simple constraints. The advantage of OWL Lite is both easier to grasp for users and easier to implement for tool builders. The disadvantage is of course a restricted expressivity.

c. **OWL DL:** OWL DL, is a subset of OWL Full. It supports maximum expressiveness while retaining the efficient reasoning facility. It is less expressive than OWL full but more expressive than OWL Lite.

2.3. Semantic Web Databases

Semantic Web databases are the databases of the Semantic Web world, designed to hold massive numbers of triples in such a manner that the information they encode can be simply retrieved. This section is further divided into two sections. Section 2.3.1 presents a brief introduction of Semantic Web databases and section 2.3.2 gives an overview of the role of the Semantic Web databases even in the presence of exiting XML stores and/or database systems i.e. relational DBMS, object oriented DBMS, and object relational DBMS.

2.3.1. Semantic Databases[1]

Semantic databases are RDF databases where semantic data can be conveniently stored, operated upon and retrieved [12]. A triple store can be defined as "A system to provide a mechanism for persistent storage and access of Semantic Web graphs." Its main functions include storing, reasoning and querying Semantic Web data. They employ index structures, algorithms for buffering, join, concurrency control for optimal query processing and reasoning. An intelligent query optimizer in a triple store strives to save resources in terms of time and memory space in query processing and reasoning.

2.3.1.1.Design goals of Semantic Web databases:

Some of the eminent design goals of Semantic Web databases are:

a. **Scalability:** Resources are described on the Semantic Web in terms of triples. A resource may need many triples for its perfect description. It is therefore necessary for Semantic Web databases to deal with a large number of triples in an elegant manner.

[1] Semantic storage systems also known as *Triple Stores*, RDF stores, Knowledge bases and Semantic Web databases

b. **Dynamism and Network Distribution:** Data on the Semantic Web is dynamic as it belongs to different sources because of network distribution. Semantic Web databases may be used as a server or a client to handle timeouts, network failure, bandwidth use and deal with denial-of-services (DoS). Semantic Web databases should be able to manage the network resources in both when a server or a client and deal with dynamic data in a graceful way.

c. **Unpredictability:** Semantic data is highly unpredictable in its nature. A Large number of triples, dissimilar terms used to describe resources, rate of triples exchange over the network and effect of network provide a high degree of unpredictability to the data. Semantic Web databases need to handle this unpredictable nature of data in an efficient and accurate manner.

d. **Provenance:** As described earlier, the Semantic Web data comes across different sources that may necessitate keeping track of original location or context of the data. This part of information is called *provenance*. Semantic Web databases may need to store the context, along with the original information.

e. **Data Processing:** Data in the Semantic Web databases need to be processed[2]. This necessitates Semantic Web databases to provide some mechanism for accessing the RDF graphs, identifying triples, storing triples in data stores, merging data from multiple sources into single store, and querying as well as administering the data stores. These operations are performed by applications many times thus require Semantic Web databases to provide lightweight, fast, easy to use and understandable APIs to carry out

[2] Triples addition, modification, removal and retrieval

these tasks. Many of the existing Semantic Web applications interact with human. So, they must perform as fast and accurate as possible in order to provide shield against frustration. Interactive level performance is one of the key requirements of these stores for human friendliness.

f. **Reasoning:** A final clear issue is support for inference. Semantic Web databases support different degree of reasoning (RDF/RDFS/OWL) while RDFS support is common. Currently only a very few stores support OWL features, and they do not provide the performance measures while using these features whereas simpler systems do.

2.3.1.2.Types of Semantic Web databases:

Different Semantic Web databases have different architectures thus result in varying performance levels. Based on their storage structure and medium, Semantic Web databases can be divided into three broad categories, *In-memory, Native, Non-native Non-memory.*

a. In Memory Stores: Triples are stored in main memory e.g. storing an RDF graph using BRAHM [9].

b. Native Store: Persistent storage systems that are disk resident with their own implementation of databases e.g. Virtuoso [38] and AllegroGraph [37].

c. Non-memory non-native Stores: Non-memory non-native stores are disk resident and employ the existing database management systems such as Microsoft SQL, MySQL, and Oracle for storing triples. 3Store [36] is an example of this type of Semantic Web databases.

Hybrid of three classes is also available, for example Jena [34] and Sesame [33]. Triple stores have their own query languages to query store's data. List of existing large triple stores include [8] BigOWLIM, Bigdata(R), Garlik, 4store ,YARS2, Virtuoso, Jena TDB, AllegroGraph, Jena SDB, Mulgara, RDF gateway, Jena with PostgreSQL, Kowari, 3store with MySQL, Sesame.

2.3.2. Need of Semantic Web databases

RDF is characterized by a property centric, extremely flexible and dynamic data model. Resources can acquire properties and types at any time, regardless of the type of the resource or property. This flexibility makes RDF an attractive technology for the specification and exchange of arbitrary metadata. The challenge is thus how to provide persistent storage for the new RDF data model in an efficient and flexible manner.

2.3.2.1.Semantic Web databases vs. Existing database systems:

Purpose of the Semantic Web databases is somewhat similar to the existing database systems i.e. management of stored data**.** RDF documents' storage necessitates special type of data stores because of two fundamental differences between RDF graph model and other data models e.g. relational data model [31] and object data model [32] that demand some special kind of data stores to manage RDF data. These two differences are:

1. Unpredictable structure of the data stored in RDF graph model
2. Unpredictable query patterns over this data in Semantic Web

All existing database systems require that structure of data (i.e. schema) must be defined before inserting that data [10]. Predefined structure of data helps in data integrity by constraining the incorrect data to be used by any organization or application. However in Semantic Web, where the interoperation between heterogeneous data sources is permissible, structure of data is unknown and changes continuously. Existing database systems are unable handle unstructured data. This gives rise to a data storage system that does not need any prior definition of the structure of data.

Existing database management systems are used by known set of applications. Such databases can be optimized on the basis of metadata i.e. indexes and estimated statistical knowledge [10] for most anticipated query patterns for these applications. Access for all other patterns is comparatively slower than these anticipated patterns. But RDF data can be accessed and manipulated by any node on the Semantic Web that requires RDF data stores to handle queries of unpredictable patterns.

Both the reasons concluded above necessitate the proposal of some new storage systems that can better handle the complexity of RDF data and query.

2.3.2.2.Semantic Web databases vs. XML Stores:

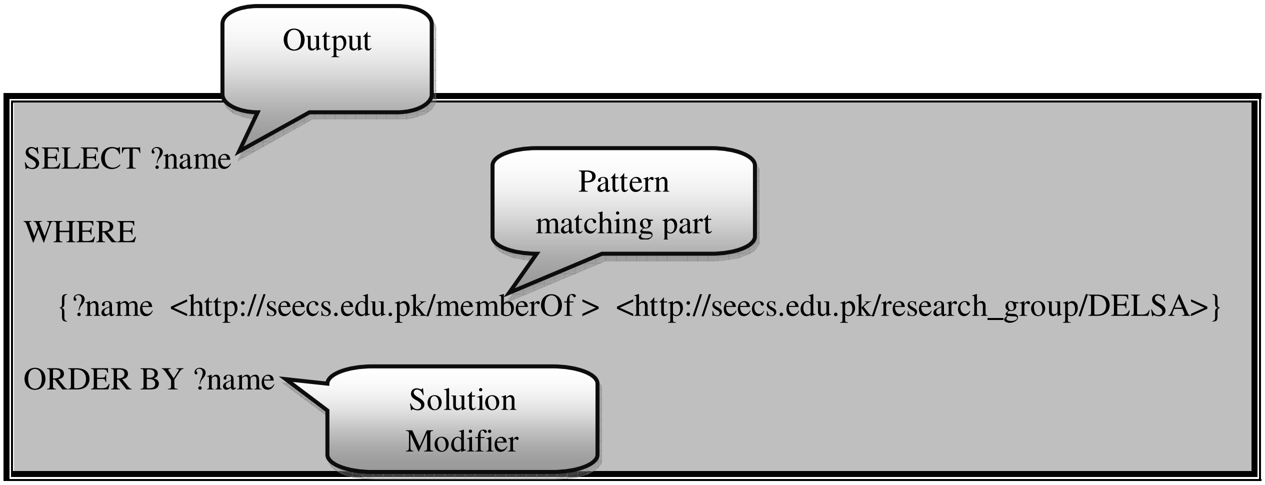
One approach for RDF storage might be to map the RDF data to XML and simply leverage prior work on the efficient storage of XML. However, the standard RDF/XML mapping is unsuitable for this since multiple XML serializations are possible for the same RDF graph, making retrieval complex [43]. Non-standard RDF-to-XML mappings are possible, and have been used in some implementations. However the simpler mappings are unable to support advanced features of RDF, such as the ability of RDF to treat both properties and statements as resources, which allows metadata describing these elements to be incorporated seamlessly into the data model and queried in an integrated fashion.

2.4. Data Extraction

Given a standard set of data representation languages and data stores for their storage, it is obvious to have a standard mechanism for extracting subsets of information from these data stores. A Semantic Web query language is a language to retrieve and manipulate data stored in Semantic Web language format [46]. RDF data model is a semantic network model that has slightly different forms for data and knowledge representation. Semantic Web query needs to be more complex than SQL since the RDF data model is more complex than the relational data model. Specifically, while a relational query executes over one or more tables each containing tuples with the same structure, an RDF query executes over a RDF container that may contain resources of different types each with different properties. Values of properties, rather than being merely data, can be resources themselves [47].

Until now several designs and implementations of Semantic Web query languages have been proposed [44][47]. Currently SPARQL is a W3C Candidate Recommendation [35][45]. In this research work we will briefly discuss SPARQL, the most eminent query language.

SPARQL [35][36][48] is a graph-matching query language. Given a data source D, a query consists of a pattern which is matched against D, and the values obtained from this matching are processed to give the answer. The data source D to be queried can be composed of multiple sources.



```
SELECT ?name
WHERE
   {?name  <http://seecs.edu.pk/memberOf >  <http://seecs.edu.pk/research_group/DELSA>}
ORDER BY ?name
```

Figure 10 Specimen Query Structure

The query shown in Figure 10 will select all unique values for ?name, where there is a triple that matches any subject ?name, and the specified predicate and object (in this case, anyone who is the member of DELSA group). The data is returned in a standard XML-based format that will be ordered on variable ?name.

A SPARQL query consists of three parts

1. Pattern matching part
2. Solution modifier
3. Output of query

The *pattern matching part* includes several interesting features of pattern matching of graphs, e.g. optional parts, union of patterns, nesting, filtering (or restricting) values of possible matching and the possibility of choosing the data source to be matched by a pattern. The *solution modifiers* are the one that comes into place when output of the pattern has been computed (in the form of a table of values of variables), allows to modify these values applying classical operators like projection, distinct, order, limit, and offset. Finally, the *output* of a SPARQL query can be of different types: yes/no queries, selections of values of the variables which match the patterns, construction of new triples from these values, and descriptions of resources.

2.5. Semantic Web databases Performance Evaluation

As mentioned in section 2.3.1 different open source and commercial Semantic Web databases are available. Many new stores are being introduced with the popularity and increased used of Semantic Web applications. Performance evaluation of Semantic Web databases is necessitated as:

1. Each one claimed to perform best. It is evidently important to measure the performance of different Semantic Web databases to verify that they perform up to developer performance claim
2. To assess the performance of a newly created or updated Semantic Web databases against existing once
3. To facilitate the user who needs to select from a library of Semantic Web databases according to their application requirement.

In general, database community has two schools of thoughts while evaluating the performance of database management system. The most popular one is introduced by Transaction Processing Performance Council (TPC)[3]. They generate some high level queries for a dataset, run these queries on data stores and produce some facts and figures that show the overall performance of the stores. While TPC-style benchmarks provide a sight of overall performance of the system, it does not mean to judge the subcomponents of that system [11]. They do not provide the

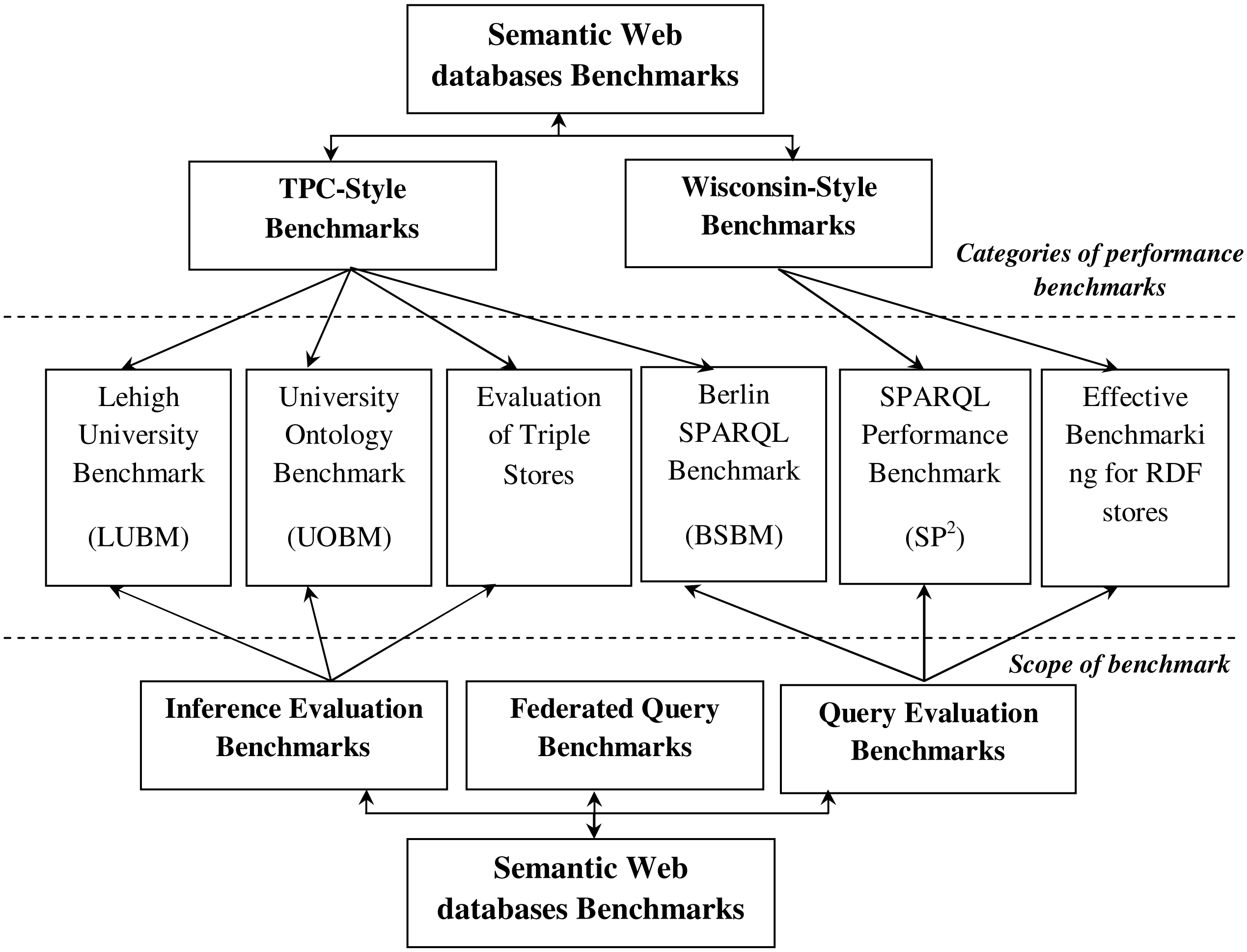
[3] http://www.tpc.org/

guidelines for the researchers or future developers of data stores. The second philosophy of generating benchmarks presents the Wisconsin-style benchmarks. Such benchmarks test the performance of subcomponents by running a large number of transactions on data stores, such as join mechanism, query optimizers, and storage layout etc. This type of benchmark provides a diagnostic measure of the weak subcomponents of any data store, so it also opens the future directions for the developers and researchers to put their efforts on right track [12]. Wisconsin-style benchmarks do not give a winner while evaluating two or more stores. Since most of the Semantic Web databases' users are interested in making decisions about purchasing best suited Semantic Web database so Wisconsin family of benchmarks are less popular among the users of Semantic Web databases. Whilst Wisconsin-style benchmarks are not important from users' perspective but they are highly considerable from research viewpoint [22] because these provide trivial facts of Semantic Web databases.

Semantic Web databases have been a key aspect of the field of Semantic Web since its inception. Various studies have been conducted to analyze the semantic storage systems. Major contribution to this area is [13]-[28]. However, according to [17] "*a benchmark is only a good tool for evaluating a system if evaluated dataset and the tested capabilities of system are similar to the once expected in the target use case*". Therefore, different benchmarks are suitable for testing different capabilities of the stores. Depending upon their scope, we can broadly divide these Semantic Web databases' performance evaluation benchmarks into three categories that are:

1. Query performance evaluation benchmarks
2. Inferencing performance evaluation benchmarks
3. Federated Query Performance evaluation benchmarks

These three types of benchmarks check the Semantic Web database performance depending upon their querying, inferencing and federated querying performance evaluation, as shown by the name of category. Depending upon the type and their scope, existing eminent benchmarks are categorized in Figure 11.



Semantic Web databases Benchmarks
TPC-Style Benchmarks
Wisconsin-Style Benchmarks
Categories of performance benchmarks
Lehigh University Benchmark (LUBM)
University Ontology Benchmark (UOBM)
Evaluation of Triple Stores
Berlin SPARQL Benchmark (BSBM)
SPARQL Performance Benchmark (SP^2)
Effective Benchmarking for RDF stores
Scope of benchmark
Inference Evaluation Benchmarks
Federated Query Benchmarks
Query Evaluation Benchmarks
Semantic Web databases Benchmarks

Figure 11: A classification of eminent Semantic Web databases benchmarks

2.5.1. Lehigh university benchmark

Lehigh university benchmark (LUBM) [17] is the one of the most popular performance evaluation so far for testing the OWL inference performance. LUBM is supposed to offer a convenient standard for comparing the performance of Semantic Web databases at a high level, and has been used in many tests of both RDF and OWL stores [17]-[19], [23]. LUBM query set is designed on the basis of some factors that include input size, selectivity, complexity, hierarchy information and logical inference. Performance metrics include load time, repository size, query response time, query completeness and soundness and a combine metric that computes the tradeoff between query completeness and query soundness.

Limitations:

1. Whilst the benchmark has been designed for the purpose of testing OWL inference performance, so it is valuable as for as OWL inference performance is measured. But currently LUBM is used for testing the Semantic Web databases' query performance. Results produced by LUBM for performance measurement of Semantic Web databases are not complete for query performance evaluation [22].

2. LUBM evaluation methodology includes query response time as a sole performance measure, missing the computation of resource consumption. Resource consumption includes CPU time and main memory utilization provides the *total cost* that is incurred in processing a query.

3. LUBM's data set is created synthetically by iterating for a variable number over a simple OWL ontology by UBA (Univ-Bench Artificial data generator) tool developed by creators of LUBM. The data produced by UBA has a miniature, clumsily repeated ontology with few properties [20]. UBA's generated datasets are quite different from Semantic Web data where data is usually comprises of semantically rich ontology.

4. LUBM's method used for performance evaluation is not very authoritative for data stores, for example time of execution of each query is taken ten times, and the average of these ten values is considered the actual query response time. Query cache influences a lot over the final results in this practice [21].

2.5.2. University Ontology Benchmark (UOBM)

University Ontology Benchmark extends the LUBM with the addition of some axioms to make full use of OWL Lite and OWL DL constructs [23]. Rohloff et al. presented their work on evaluation of triple stores in [19]. They evaluated the triple store technologies for large data stores. LUMB was the underline evaluation framework for these benchmarks.

Limitations:

Both UOBM and Rohloff's work inherit all limitations from their ancestor

2.5.3. Berlin SPARQL Benchmark (BSBM)

Berlin SPARQL Benchmark (BSBM) [23]-[25] is a language specific benchmark that uses Wisconsin benchmark techniques. BSBM measures the performance of the semantic storage systems that expose SPARQL endpoints and some SPARQL to SQL writer. This benchmark presents a more comprehensive query set than that of LUBM. The query set is design to test the SPARQL query language feature and to test the performance of stores under workload. Two different Query mixes, comprises of number of different queries, are introduced to test the anticipated user access patterns. Justification of each query is given along with the SPARQL construct that is being tested. BSBM provides following performance metrics. *Query Mix per Hour* (QMpH), the main performance metric of BSBM, measures the query mixes that are answered by any tested system in an hour. *Query per Second* (QpS) measures the number of a typical type of query that is executed in a second by any tested system. *Load time,* that gives the cumulative time required to load a dataset from its source file into the tested system. BSBM has lot of advantages such as multiuser scenarios testing and two variations of query mix to check the performance of store under anticipated user access patterns but limitations are still there.

Limitations:

1. This benchmark presents comparison of stores in an e-commerce scenario so it lacks generality.

2. Query execution time is the only considered dimensions for evaluation metric, while resource utilization is important to be considered due to limited resources.

3. Queries are not defined to measure insertion (after the initial bulk insertion) and deletion performance of Semantic Web databases.

4. BSBM analyzed the stores' performance for different artificially generated dataset, largest of which comprises of 100M triples. Large semantic datasets are freely available such as Barton Library Dataset [29] (about 50M triples) and U.S. Census [30] (about 1B triples), as artificially generated dataset lack originality to some extent so it's better to use genuine available datasets.

2.5.4. SPARQL Performance Benchmark (SP^2 Bench)

SPARQL Performance Benchmark (SP^2 Bench) [27][28] is a language specific benchmark framework particularly designed to test SPARQL engines for SPRAQL constructs and broader range of RDF data access patterns. SP^2 Bench measures the performance of engine with regard to query optimization techniques. SP^2 bench is the most comprehensive Wisconsin-style Semantic Web databases benchmark at the moment. It offers extensive query set that vary in general characteristics such as selectivity, query complexity and output size, and different types of joins. The benchmark proposes newer and comprehensive performance metrics that covers diverse aspects of evaluation process such as success rate, load time, per query performance, global performance and memory consumption. SP^2 is the only benchmark; that provide limited information about the main memory utilization of various Semantic Web databases for query execution.

Limitations:

1. Whilst very appealing yet benchmark is well-suited only to identify the deficiencies in SPARQL engines.
2. Secondly analysis are conducted on six different sizes of synthetically generated datasets, maximum of these data sizes is 25M triples. Real datasets with greater number of triples are online available for testing purpose.
3. Last but not the least, insertion and deletion test cases are not handled.

2.5.5. Effective benchmarking for RDF store

Effective benchmarking for RDF store [22] by Alisdair at al. is another Wisconsin-style benchmark. Although authors claim to have a detail analysis of Semantic Web databases, yet the results lack in detail evaluation and analysis. Time is the only considered performance metric. Store’s performance on assertion and deletion are provided with a little emphasis on its query performance.

Limitations:

1. The benchmark presents the test cases that check some data management techniques and SPARQL query constructs but queries related to these test cases are neither presented nor tested.

2. Performance metrics considered is response time only. For a database operation response time is not the sole performance metric for any system.

3. The benchmark proposed a data generator to generate a test dataset. This synthetic data generator generates tree structure RDF data, but in reality Semantic Web data is always in graph format so results produced on this synthetic data will be somewhat different from results obtained from real graph based data.

Apart from these benchmarks, numerous technical reports and surveys have been published to study the Semantic Web databases [13]-[16]. The most initial studies to these evaluations include the **MIT Scalability Report** [13]. Although they worked with standard datasets but due to specific study conducted for browser like applications, metrics are defined only according to simile application requirements and best store suited for such applications was suggested. Time was the major consideration to test Semantic Web databases. The **Semantic Web Advanced Development for Europe** (SWAD-E) project produced surveys to understand the Semantic Web databases features and storage structure [14][15][16]. These limited surveys provide an overview of a number of open source Semantic Web databases' implementations that were available at that time but do not include a detail quantitative analysis of stores' performance. Beside lack of performance measurement of Semantic Web databases, these surveys are out-dated as many newer Semantic Web databases with better performance and structure has been purposed.

2.6. Critical Analysis

In a nutshell, we can say that existing benchmarks test the performance of Semantic Web databases, but their main limitations that we overcome in this research are:

- Most of the existing benchmarks, reports and surveys follow the TPC-style techniques for performance evaluation, so they do not provide a detail subsystems level analysis to identify the key strengths and weakness.

- Almost all benchmarks used synthetically generated data sets. All synthetic data generators have some limitations that prevent them to produce data similar to Semantic Web data. So performance results produced over synthetic data generators cannot reliable.

- Query response time cannot be a sole performance measurement for any database system. A database operation utilizes system resources that are important to consider. A good Semantic Web database utilizes all resources efficiently. Resource utilization of the Semantic Web databases is not computed by any existing benchmark.

- Database operations can be broadly divided into four major operations i.e. CRUD operations (Create, Read, Delete and Update). No benchmark has presented the detail study of all database operations.

- All these benchmarks provide a comparative study, but none of them provide any information regarding the scalability behavior shown by tested Semantic Web databases.

- All the studies made about Semantic Web databases are done in performance evaluation perspective; none of the surveys provide an insight to a particular storage schema with which a level of performance is attained.

Summary of Chapter:

In this chapter we discussed the pre-requisite knowledge required to understand the research, presented in this thesis. We described Semantic Web and its languages followed by Semantic Web databases. A brief comparative analysis between other database systems and Semantic Web databases is presented to establish understanding why Semantic Web stores are necessary. By the end of this chapter, some eminent benchmarks are discussed and analyzed critically.

OVERVIEW OF EXISTING SEMANTIC WEB DATABASES

This chapter provides the detail of evaluated *Semantic Web databases' storage architecture and access methods*. It first describes the database architecture and then storage schemas and type of indexes used for storage and retrieval of the triples in a particular Semantic Web database. We start with the native stores by stating the storage architecture of TDB, Sesame_{N} and AllegroGraph. Then we present the storage architecture of SDB and $\text{Sesame}_{\text{RDB}}$ to show how RDF triples are stored in RDBMS. Last but not the least section describes the in-memory Semantic Web databases storage architecture.

3.1. Database Architecture

This section provides a brief description of existing Semantic Web databases' system architecture. A feature comparison of considered Semantic Web databases is presented in Table **2**.

3.1.1. Jena

The software producers of *Jena* [57] are the HP Labs[4], which are a part of the Hewlett-Packard Development Company. Jena was developed in the terms of the HP Labs Semantic Web Research. The associated license of the Jena project is completely *open source*. This implies that redistribution and use in source and binary forms with or without modification are permitted[5] . The Jena download package includes the source files of the entire Jena project implemented in Java. This provides a basis for implementations extending the framework, for instance with new indices.

[4] http://www.hpl.hp.com/

[5] http://jena.sourceforge.net/license.html

An architectural overview of Jena is presented in Figure 12. This framework offers methods to load RDF data into a memory based Semantic Web database, a native storage or into a persistent triple store.

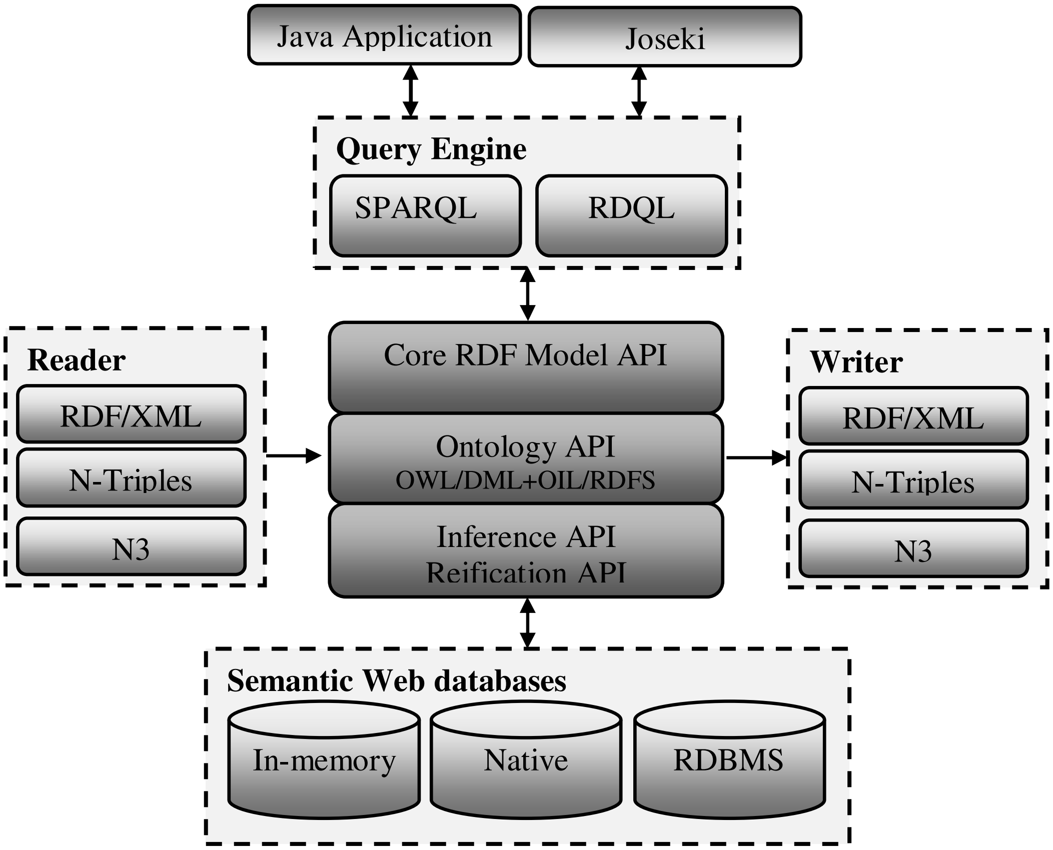


Figure 12: Architectural Overview of Jena [63]

In order to build a persistent triple store a variety of relational databases, can be used. Jena supports Microsoft SQL Server 2005 including SQL Server Express, Oracle 10gR2 including Oracle Express, IBM DB2 including DB2 Express, PostgreSQL v8, MySQL 5.0 (>=5.0.22), HSQLDB 1.8, H2 1.0.73 and Apache Derby 10.2. The stored data may be retrieved through SPARQL queries. A standard implementation of the SPARQL query language is encapsulated in the ARQ package of Jena. SPARQL queries can be executed using Java applications or by the use of the graphical frontend *Joseki*. The Ontology API provides methods to work on ontologies of different formats, like OWL and RDFS. Jena's Core RDF Model API offers methods to create, manipulate, navigate, read, write or query RDF data. The remaining major components are on the one hand the Inference API, which allows the integration of inference engines or reasoners into the system. On the other hand the Reification API is a proposal to optimize the representation of reification.

OWL support is given in form of the Ontology API. The inference subsystem[6] enables the use of inference engines or reasoners in Jena. Besides SPARQL, RDQL is a supported query language. In a tutorial about RDQL, it is recommended that new users of Jena should use SPARQL instead. Jena uses readers and writers for RDF/XML, N-Triples and N3, which are commonly known RDF data formats.

3.1.2. Sesame

The *Sesame* system is a web-based architecture that allows the persistent storage of RDF data and schema as well as online querying of the information [64]. The software producer of Sesame [60] is Aduna[7]. This company sets the focus of their work in revealing the meaning of information. Like Jena, Sesames associated license is *open source* underlying the BSD-style license. In order to use Sesame, *Apache Tomcat* is recommended. The Sesame package also contains two web applications, the Sesame server which stores the RDF data and the OpenRDF Workbench as a graphical frontend for the server. This workbench can manage repositories, load RDF data and execute queries. Sesame is able to handle RDF/XML, N-Triples, N3 and Turtle format of RDF data. It supports in memory, native and relational database storage. Alternatively

[6] http://jena.sourceforge.net/inference/

[7] http://www.aduna-software.com/

to SPARQL Sesame is able to interpret the Sesame RDF Query Language (SeRQL) [1] integrated for enhancing the functionality of RQL and RDQL. Sesame offers parsers for various well known RDF formats N3, N-Triples, RDF/XML, Turtle and two new formats TriG[8] and TriX.

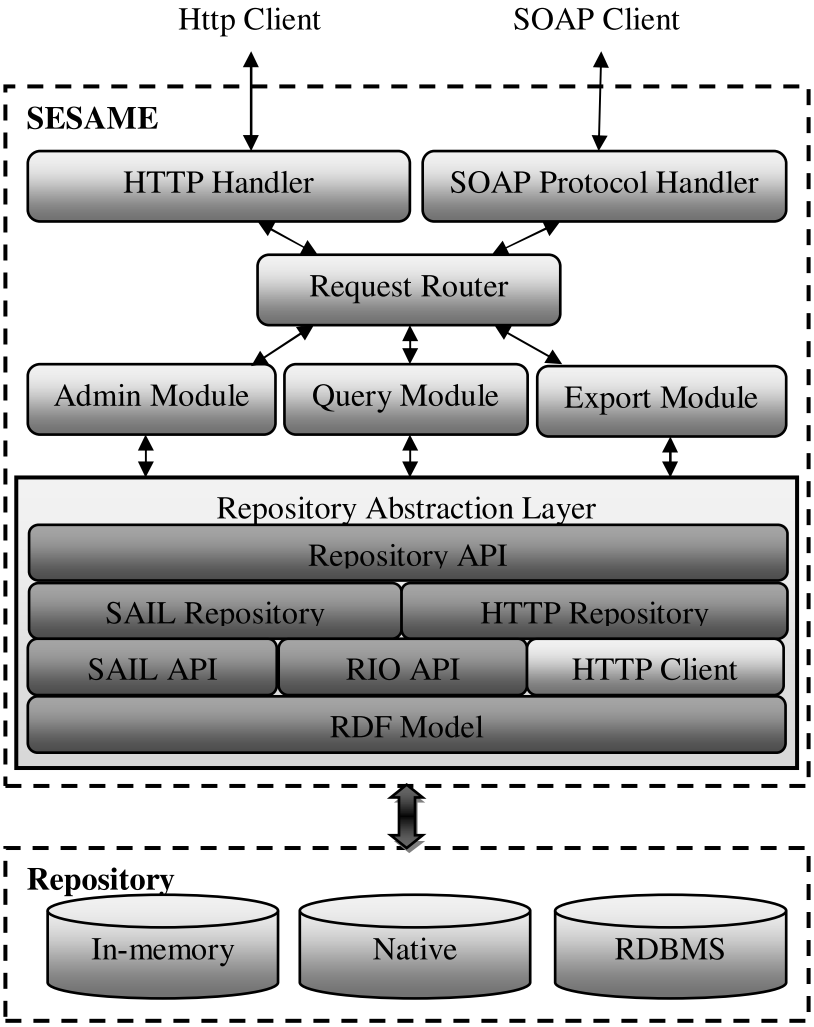


Figure 13: Architectural Overview of Sesame

Figure 13 illustrates an *architectural overview* of Sesame. The *Repository Abstraction Layer (RAL)* is an interface that offers RDF-specific methods to its clients and translates these methods to calls to its specific repository [64]. It contains all repository specific code, in order to keep

[8] http://www4.wiwiss.fu-berlin.de/bizer/TriG/

Sesame repository independent, making it possible to implement Sesame on top of a wide variety of repositories without changing any other component.

The *RDF Model* implements basic concepts about RDF data. The component *RDF I/O (Rio)* consists of a set of parser and writer for the handling of RDF data. This is for instance used by the *Storage And Inference Layer (Sail)* API for initializing, querying, modifying and the shutdown of RDF stores. On the topmost layer constitutes the *Repository API* the main entrance to address repositories. Compared to Sail, which is rather a low level API, the Repository API is the associated high level API with a larger amount of methods for managing RDF data. The *HTTP Repository* is an implementation that acts like a proxy in order to connect to a remote Sesame server via the HTTP protocol.

Admin module, *query module* and *export module* are three functional modules of Sesame. These are the clients of the RAL. Query module evaluates queries posed by the users, administration module allow RDF data and schema information to be inserted into as well as deleted from a repository and export module allows for the extraction of the complete schema and /or data from a model in RDF format.

3.1.3. AllegroGraph

The software producer of AllegroGraph RDF Store is Franz Inc.[9]. The company has been founded in 1984 and is well known for its Lisp programming language expertise. Recently, they also started developing semantic tools, like AllegroGraph. The associated licenses of AllegroGraph come in two different flavors. The version evaluated in this research is the free edition, which is limited to 50 million triples maximum. In contrast to that, the enterprise version

[9] http://franz.com/

has no limits regarding to the number of stored triples but underlies a commercial license. AllegroGraph is not extensible. It is closed source and stores data as well as the database indices inside its particular storage stack. An architectural overview is not possible because of its closed source. Figure 14 shows client server architecture of AllegroGraph.

The software is developed especially for 64 Bit systems and runs out of the box, as it does not need any other databases or software. Storage, indexing and query processing is performed inside AllegroGraph. The software can be accessed using Java, C#, Python or Lisp. There are bindings for Sesame or Jena integration available and also an option to access AllegroGraph via HTTP.

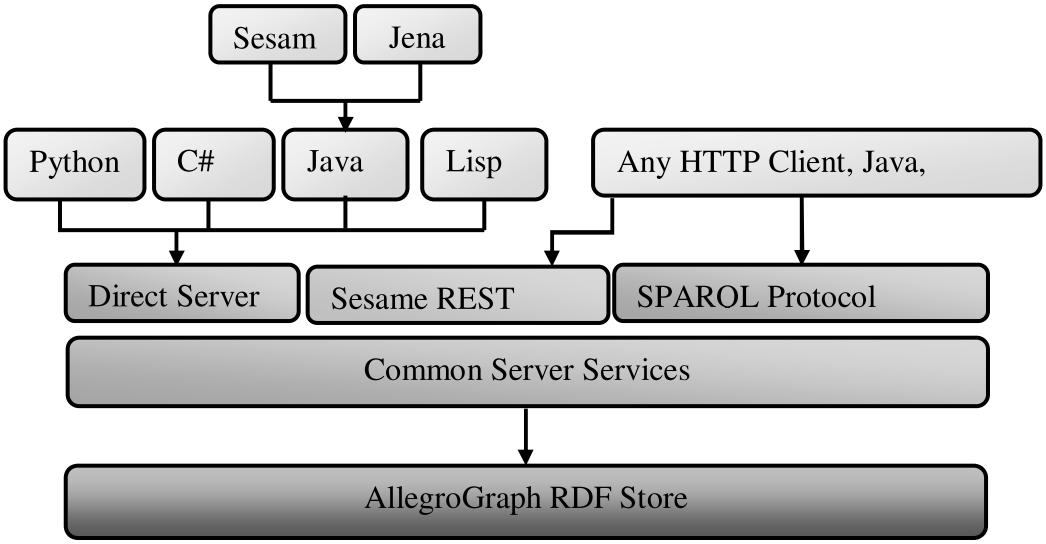


Figure 14: Client Server Architecture of AllegroGraph [63]

Franz Inc. suggests using TopBraid Composer[10] by TopQuadrant Inc. for OWL support. The available query language of the software is SPARQL, but it also supports low level API calls for direct access to triples by subject, predicate and object. With those API calls, it is possible to retrieve all datasets matching a certain triple. The API calls provide functionality, which can be

[10] http://www.topquadrant.com/topbraid/composer/index.html

compared to SQL SELECT statements. The interpretable RDF data formats of AllegroGraph are RDF/XML and N-Triples. Other formats are planned to be supported in future versions.

Table 2: Overview of Evaluated Semantic Web databases

Name	Supported Storage	Supported RDF format	Supported Query Language	Programming Language	License
Jena	In-memory, native disk storage, relational backend	RDF/XML, N3 and N-Triples	SPARQL, RDQL	Java	Open source
Sesame	In-memory, native disk storage, relational backend	RDF/XML, N3, Turtle and N-Triples	SPARQL, SeRQL	Java	Open source
AllegroGraph	Native disk storage	RDF/XML and N-Triples	SPARQL	Java	Commercial and free edition

3.2. Storage Layouts and Access Mechanisms

3.2.1. Native Stores

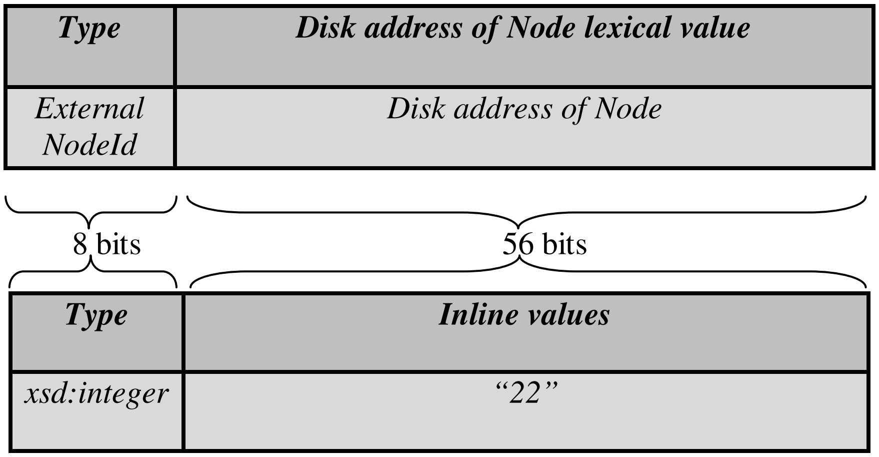
Native stores provide persistent storage for Semantic Web data. These databases create disk based files to store Semantic Web data. Native stores implement different data structures, a detail study of few triple stores is provided here.

3.2.1.1.TDB

Jena TDB stores RDF triples in a directory on the disk in filing system. Whenever a TDB store is created, it creates some files for the storage and retrieval of triples, that can be broadly divided into three categories that are *Nodes*, *Prefixes*, *Triples & Quads*

Nodes:

TDB files '*nodes'* and '*node2id'* provide two types of mappings from node to nodeId and from nodeId to node. The '*Node to NodeId mapping'* is used during data loading and when converting constant terms in queries from their Jena Node representation to the TDB-specific internal ids. The '*NodeId to Node mapping'* is used to turn query results expressed as TDB nodeIds into the Jena node representation and also during query processing when filters are applied if the whole node representation is needed for testing e.g. regex.



Type	***Disk address of Node lexical value***
External NodeId	*Disk address of Node*

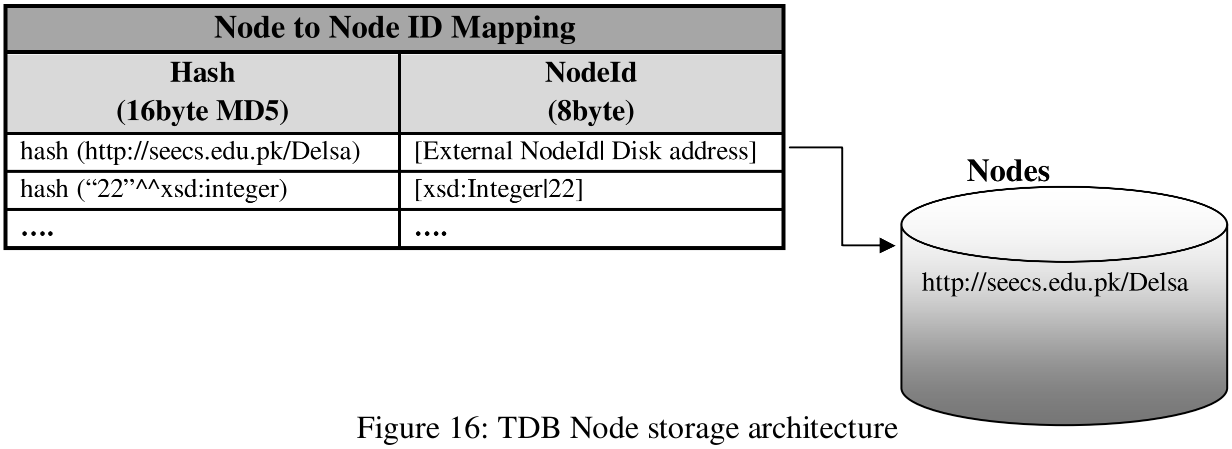
Type	***Inline values***
xsd:integer	*"22"*

Figure 15: Structure and Example of NodeId Types

A nodeId can be of two types in Jena TDB i.e. *External nodeId* or the *Value space* as shown in Figure 15. First byte of the nodeId stores the type of the nodeId. If type is external nodeId then next 7 bytes contain the physical address of the node as describe above. If the type of nodeId is value space then the values of data types, which are considered in value space, are stored as a part of nodeId in lower 7 bytes. Data types that are considered in value space are xsd:decimal, xsd:integer, xsd:dateTime, xsd:date and xsd:Boolean.

The *node* file stores the actual Jena node representation. The node to nodeId mapping is based on the hash of the lexical value of the node and is stored in *node2id* file that is implemented as a B+ tree. The size of an entry in this file is of 24 bytes. The first 16 bytes are the hash value of node and next 8 bytes are the disk address of the node lexical value (except for the inline values) in the node file.

Whenever a node is asserted into a TDB store MD5 hash of the node is computed and enters into the first 16 bytes. Then a unique id is assigned against this hash value, this id represents the physical address in node file. Actual lexical value of that node is stored at the address which is represented by the nodeId. The storage process of node is represented in the Figure 16.



Node to Node ID Mapping	
Hash (16byte MD5)	**NodeId (8byte)**
hash (http://seecs.edu.pk/Delsa)	[External NodeId| Disk address]
hash (“22”^^xsd:integer)	[xsd:Integer|22]
….	….

Figure 16: TDB Node storage architecture

Prefixes

This category contains three files that are *prefixes*, *prefix2id* and *prefixidx*. These provide supports for TDB *Prefix Mappings*. Just like nodes and node2id, *prefixes* and *prefix2id* provide two types of mapping for prefixes and prefixIds. Prefixidx is another implementation of B+ tree that is ordered on GPU (Graph, Prefix, URI).

Triple and Quads

Remaining files are categorized under this category. These are *SPO*, *POS*, *OSP*, *GSPO*, *GPOS*, *GOSP*, *SPOG*, *POSG*, and *OSPG*.

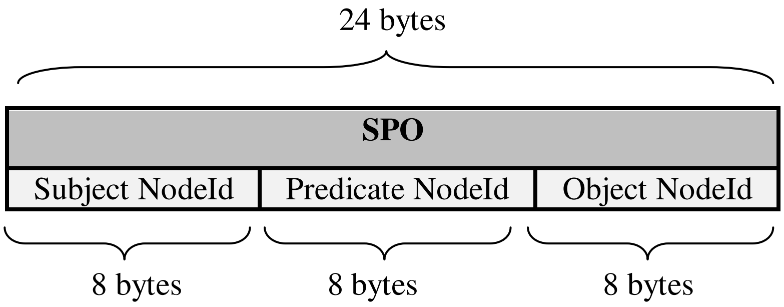


Figure 17: Triple entry in SPO file

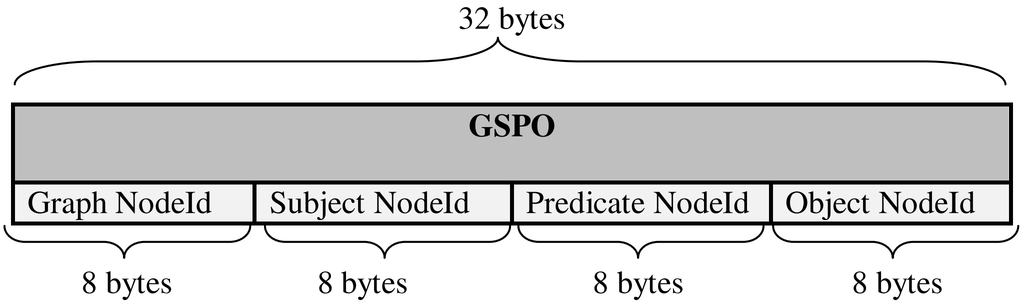


Figure 18: Quad entry in GSPO file

There is no distinguishing triple file and then indexes on this file. SPO, POS and OSP are triple index files that have B+ tree implementation. These are populated when no provenance information is stored about the triples. Each entry of these files is of 24 bytes and has all the information about a triple. Triples in these files are represented as a combination of three nodeIds in different orders, one for subject second for predicate and third for object. Name of each file represents the order of triple in terms of subject, predicate and object as shown in Figure 17.

Whenever a triple is asserted into Jena TDB store three entries are made in three different files, one entry in each of SPO, POS and OSP files.

Quads index files are used to represent the named graphs. Default storage of these files in Jena TDB is B+ tree. These are populated when provenance information is stored about the triples. Each entry in quad index files is of 32 byte representing subject, predicate, object and graph for a triple as shown in

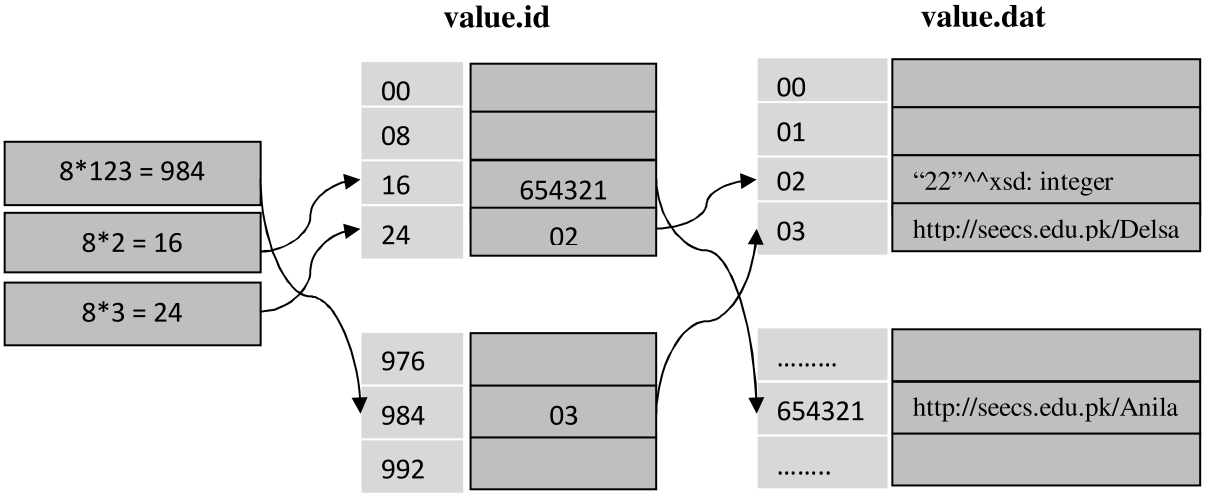
Figure **18**. Whenever a quad is asserted into Jena TDB store six entries are made in six quad index files, one entry in each file.

3.2.1.2.Sesame$_N$

Sesame$_N$ stores triples permanently on disk for inferencing and querying of Semantic Web data. Sesame$_N$ schema in the disk directory can broadly be divided into three logical blocks. (1) Namespaces (2) Values (3) Triples.

Values

This logical part is composed of three disk based files that are named as *value.dat*, *value.id* and *value.hash*. These three files provide two types of mapping i.e. *"value to value-Id"* and *"value-Id to value"*. The '*value to value-Id mapping'* is used during data loading and when converting constant terms in queries from their node representation to the Sesame specific internal value-Ids. The '*value-Id to value'* is used to turn query results expressed as Sesame value-Id into the RDF node representation and also during query processing when filters are applied if the whole node representation is needed for testing e.g. regex.



valueId(http://seecs.edu.pk/Delsa) = 123
valueId(http://seecs.edu.pk/Anila) = 2
valueId("22"^^xsd: integer) = 3

Figure 19: value-Id to value mapping in Sesame$_N$

The actual values of URIs, blank nodes and literals are stored sequentially in the *values.dat* file. *Value-Id to value mapping* is maintained in value.id file. It is a B-tree disk based implemented file. The values.dat offset for value X is stored in values.id at offset 8 * X, where X is a positive 32 bit integer and offsets are 64 bit longs. So, to look up the lexical value for id 123, the native store fetches the long stored at offset 8 * 123 = 984 from values.id. The value of this long is, for example, 654321. The lexical value can then be read from values.dat at offset 654321 as shown in Figure 19.

The *value to value-Id mapping* is based on the *value.hash* file that is a disk based hash table. It stores value identifiers using a hash code derived from the actual value. Hash of any RDF node's lexical value (i.e. resource, literal or blank node) returns the physical address of the value-Id for that node, within the address space of *value.hash* as shown in

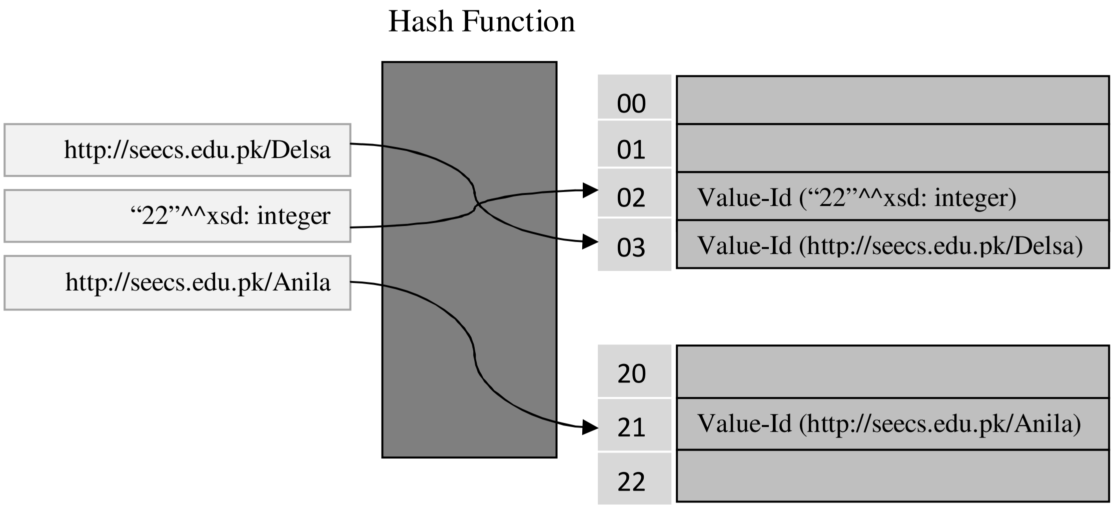

Hash Function
http://seecs.edu.pk/Delsa
"22"^^xsd: integer
http://seecs.edu.pk/Anila
00
01
02
03
Value-Id ("22"^^xsd: integer)
Value-Id (http://seecs.edu.pk/Delsa)
20
21
22
Value-Id (http://seecs.edu.pk/Anila)

Figure 20: value to valueId mapping

Triples

Triple-sopc, triple-posc and triple-cosp fall under this category. These are on-disk *indexes* to speed up querying. These uses B-trees for indexing statements, where the index key consists of four fields: subject (s), predicate (p), object (o) and context (c). These file only store identifiers (integer ids) instead of actual URIs, blank nodes and literals as shown in Figure 21. The order in which each of these fields is used in the key determines the usability of an index on a specify statement query pattern. Searching statements with a specific subject in an index that has the subject as the first field is significantly faster than searching these same statements in an index where the subject field is second or third. In the worst case, the 'wrong' statement pattern will result in a sequential scan over the entire set of statements.

SOPC			
Subject valueId	Object valueId	Predicate valueId	Context valueId

Figure 21: Triple in Sesame$_N$

By default, the native repository only uses two indexes, one with a subject-predicate-object-context (spoc) key pattern and one with a predicate-object-subject-context (posc) key pattern. However, it is possible to define more or other indexes for the native repository, using the *Triple indexes* parameter. This can be used to optimize performance for query patterns that occur frequently. Creating more indexes potentially speeds up querying, but also adds overhead for maintaining the indexes. Also, every added index takes up additional disk space.

Namespaces

A single file contains the namespaces of the dataset.

3.2.1.3.AllegroGraph

Allegrograph store's structure can logically be divided into five major blocks, which are assertions, indices, strings, Freetext indices and deletion records as shown in Figure **22**.

.

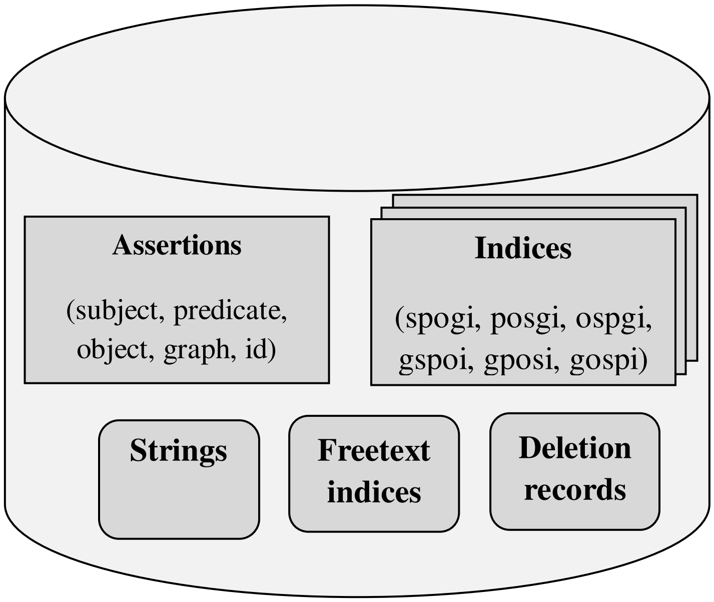

Assertions
(subject, predicate, object, graph, id)
Indices
(spogi, posgi, ospgi, gspoi, gposi, gospi)
Strings
Freetext indices
Deletion records

Figure 22: Logical Schema of an AllegroGraph Store

In Semantic Web world, triples are RDF statements comprise subject, predicate and object value in the context of graph. AllegroGraph stores triples that are composed of five fields; subject, predicate, object, graph and triple-id. Where graph encodes the context of the triple to save the provenance information and triple-id is a unique identifier assigned to each unique triple. Main purpose of triple id is to make reification supper efficient in AllegroGraph.

All of subject, predicate, object, and graph are strings of arbitrary size. It is very inefficient to store all of the duplicated strings directly as a part of a triple. On the basis of this argument in AllegroGraph a special number, called a Unique Part Identifier (UPI) is assigned to each unique RDF node value.

1st byte	***11 bytes***
Type (0-3)	*Hash of String value of Node*

1st byte	***11 bytes***
Type (4-44)	*Encoding of UPI contents*

Figure 23: Type of UPIs in AllegroGraph

There are two types of UPIs in AllegroGraph as shown in Figure 23. A UPI size is twelve byte. First byte of UPI denotes the type code of its corresponding RDF node. If value of first byte is ranging from 0-3 i.e. it node type is either a resource, a literal, a literal datatype and a literal language, then next eleven bytes are used to store the hash of the lexical value of the node and actual lexical value is store in "***String Dictionary***" that keeps the mappings between UPI and nodes string values. This allows the prevention of duplicate values. However if first byte has a type code from any other supported types then next eleven bytes store the nodes actual contents. Table 3 shows the currently supported node types in AllegroGraph. Its ability to encode values directly into its UPIs provide the facility to bypass the *String Dictionary* completely thus allowing both more efficient data retrieval and extremely efficient range queries.

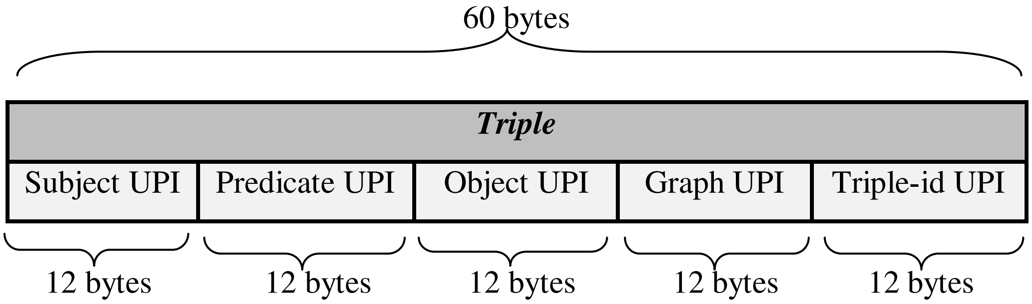


Figure 24: Triple Structure in AllegroGraph

Whenever a triple is store in AllegroGraph store. All its parts i.e. subject, predicate, object and graph are either assigned the corresponding URIs (hashed /encoded) and complete triple is assigned a unique identifier called *triple-id* and an entry of 60 bytes is made in “***Triple Directory”*** as shown in Figure 24.

AllegroGraph provides the facility to create up to six B-tree indices for efficient data retrieval. These indices are named as ***spogi, posgi, ospgi, gspoi, gposi,*** and ***gospi***. Each index is named by the order in which triple is sorted. Apart from these six indices AllegroGraph provides the facility of “***Freetext Indices”*** to perform free text searching in the *Triple Directory*. Final block of AllegroGraph keep track of deleted records in ***“Deletion Record Directory”***.

Table 3: Node types in AllegroGraph

Type code	Type name	Type code	Type name	Type code	Type name	Type code	Type name
0	Node/ resource	1	Literal	2	literal-typed	3	literal-language
7	literal-short	8	blank-node	11	single-float	12	double-float
14	unsigned-byte	15	unsigned-short	16	unsigned-int	17	unsigned-long
18	Byte	19	Short	20	Int	21	Long
23	Date	24	Time	25	date-time	26	gyear
27	telephone-umber	28	latitude	29	Longitude	30	triple-id
31	default-graph	36	geospatial	41	Subscript	42	long-88
43	unsigned-long-88	44	Plain				

3.2.2. Non-memory non-native Stores

Non-memory non-native stores also provide persistent storage for Semantic Web data. They use the storage and querying techniques provided by existing RDBMS. Detailed study of different storage layouts deployed by Semantic Web databases is presented in this study.

3.2.2.1.SDB

SDB is a subsystem of Jena that is design to support the scalable storage and query of RDF and OWL data using conventional SQL databases [58]. SDB is designed specifically to support SPARQL.

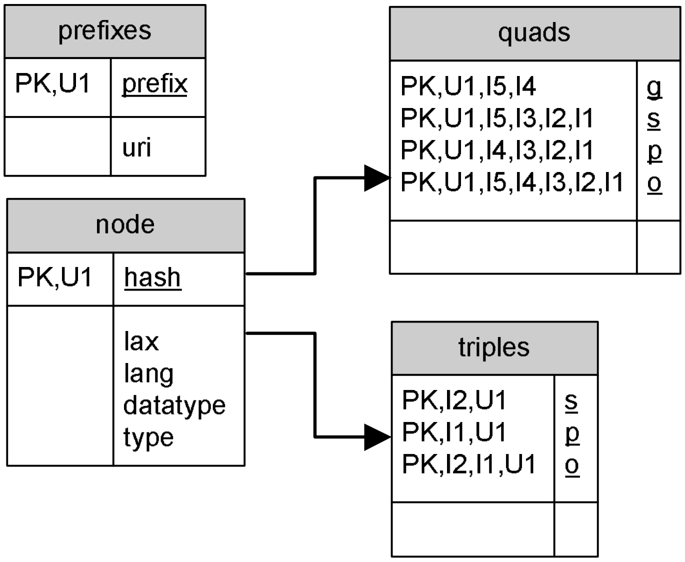


Figure 25: SDB Layout in MySQL

SDB follows the '*schema oblivious'* approach as storage schema does not change even if the schema of the data to be stored changes [64]. SDB always creates four tables in database with table name; *Prefixes*, *Nodes*, *Triples*, and *Quads.* The storage schema factors out the common prefixes of URI to reduce the storage space. It creates a separate table named as *"Prefixes"* to store the association of unique identifiers with each distinct prefix as shown in Table 4. SDB uses an *id-based approach* for triple storage that requires an additional table *"Nodes"* for storing one to one mapping between lexical values and corresponding identifiers as shown in Table **5**. SDB supports two types of node tables depending upon the layout choice. In case of *Index layout*, a separate id is used beside hash values of the node. In *hash layout*, hash of the node is used as a node identifier. Figure 25 is presenting the hash layout. In hash layout, "*Triple"* table stores the hash value as an id for all nodes of the triples as shown in

Table 6 and lexical values for these triples are stored in nodes table. *"Quads"* table store the ids of dataset quads for maintaining the provenance information as shown in Table 7.

Table 4: Prefixes table Design

Prefixes		
Column	***Type***	***Description***
Prefix	varchar (50)	Prefix for the corresponding asserted uri
Uri	varchar (500)	Asserted URI
Primary key: prefix		

Table 5: Nodes Table Design

Node		
Field	***Type***	***Description***
hash	bigint(20)	CRC32 hash value for the asserted node
Lax	Longtext	Actual value of the asserted node
lang	varchar (10)	Language Identifier for the nodes if literals
datatype	varchar (200)	Data type for the nodes if literals
type	int(10) unsigned	Type of asserted node to differentiate blank nodes, literals and URIs. Type will be 1 for blank nodes, 2 for URIs and 3 for Literals.
Primary key: hash		

Table 6: Triples Table Design

Triples		
Column	***Type***	***Description***
S	bigint(20)	Hash value for the subject of asserted statement
P	bigint(20)	Hash value for the predicate of asserted statement
O	bigint(20)	Hash value for the object of asserted statement
Primary key: s,p,o		

Table 7: Quads Table Design

Quads		
Column	***Type***	***Description***
G	bigint(20)	Hash value for the graph of asserted statement
S	bigint(20)	Hash value for the subject of asserted statement
P	bigint(20)	Hash value for the predicate of asserted statement
O	bigint(20)	Hash value for the object of asserted statement
Primary key: g,s,p,o		

SDB creates one index on both of Prefixes and Nodes table, three indices for Triples table and six indices for Quads table. Detailed Description of each index is shown in Table 8.

Table 8: Detail of Indices on SDB tables

Index on Prefixes Table			
Key_name	**Column_name**	**Non/unique**	**Index type**
Primary	prefix	Unique	Btree
Index on Node table			
Key_name	**Column_name**	**Non-unique**	**Index type**
Primary	Hash	unique	Btree
Indexes on Triple Table			
Key_name	**Column_name**	**Non/unique**	**Index type**
Primary	s,p,o	Unique	Btree
ObjSubj	o,s	Non-unique	Btree
PredObj	p,o	Non-unique	Btree
Indexes on Quads table			

Key_name	Column_name	Non-unique	Index type
Primary	g,s,p,o	Unique	Btree
SubjPredObj	s,p,o	Non-unique	Btree
PredObjSubj	p,o,s	Non-unique	Btree
ObjSubjPred	o,s,p	Non-unique	Btree
GraPredObj	g,p,o	Non-unique	Btree
GraObjSubj	g,o,s	Non-unique	Btree

3.2.2.2.Sesame$_{RDB}$

Sesame$_{RDB}$ store stores its data in a relational database. Sesame$_{RDB}$ supports *‘schema-aware’*, *‘schema-oblivious’*, and *‘hybrid’* as storage layouts for RDF data. The database's table layout can be tweaked using the “Max number of triple tables” parameter. Schema-oblivious approach creates a “*monolithic layout*” with a single table that stores all statements, by setting *maximum number of tables’* parameter equal to one. Schema- aware approach creates a “*vertical layout*” that stores statements in a per-predicate table, by setting *maximum number of tables’* parameter equal to zero or a negative value. Hybrid approach creates predicate tables as well as a single triple table that are collectively equal to desired *max number of tables*. The *Schema-aware* layout has better query evaluation performance on most data sets, but potentially leads to huge amounts of tables, depending on the number of unique predicates in dataset. If the number of tables becomes too large, the database's performance can start to decrease or it can even fail completely. Vertical layout in Sesame RDB is shown in Figure 26.

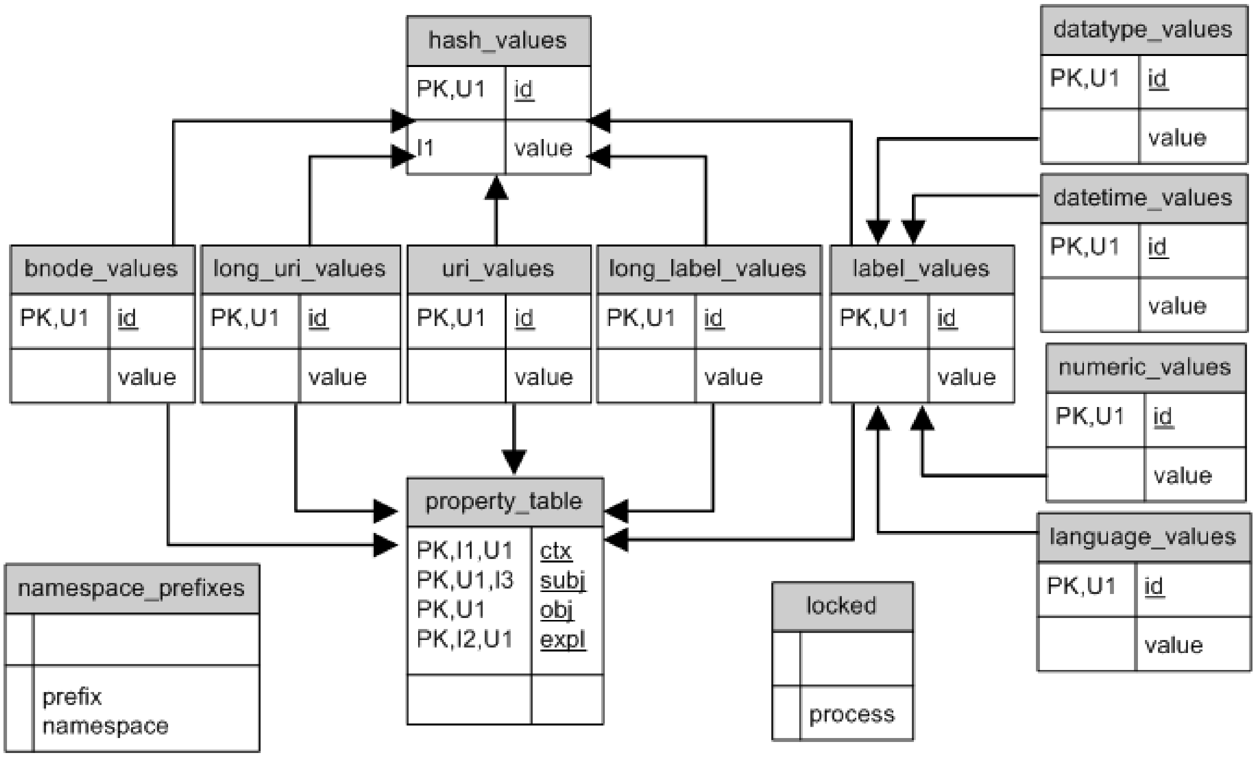


Figure 26: Sesame$_{\text{RDB}}$ Layout in MySQL

As shown in Figure 26 Sesame$_{\text{RDB}}$ always creates twelve fixed tables with table names; *bnode_values*, *uri_values*, *label_values*, *long_label_values*, *long_uri_values*, *datatype_values*, *datetime_values*, *numeric_values*, *language_values*, *hash_values*, *namespace_prefixes*, and *locked*. Along with these twelve tables it creates *per-property* tables in case of schema-aware approach or a single *triples* table in case of schema-oblivious approach. Each table along with fields and their data type is described in Table 9, Table 10, Table 11, Table 12 and Table 13.

Table 9: Property table description

Property_table		
Column	***Type***	***Description***
Ctx	int(11)	Context of the data
Subj	int(11)	Id of the subject node for the triple with predicate of corresponding

		table
Obj	int(11)	Id of the object node for the triple with predicate of corresponding table
Expl	tinyint(1)	Value is zero if the triple is explicit and one if triple is implicit
Primary key: id		

Table 10: {bnode, label, uri, long_uri, long_lable, datetime, datatype, numeric, language}_values tables description

{bnode_values/label_values/uri_values/long_uri_values/long_label_values /datetime_values/datatype_values/numeric_values/language_values}		
Column	**Type**	**Description**
Id	int (11)	Ids of the {bnode/ label/ uri/ long-uri/ long-label/ datetime/ datatype/ numeric/ languages}, asserted into store as a part of statements
Value	varchar (127)^{1} longtext^{2} bigint(20)^{3} double^{4}	Values of the {bnode^{1}/ label^{1}/ uri^{1}/ long-uri^{2}/ long-label^{2}/ datetime^{3}/ datatype^{1}/ numeric^{4}/ language^{1}}, asserted into store as a part of statements
Primary key: id		

Table 11: Hash_values table

hash_values		
Column	***Type***	***Description***
Id	int (11)	Ids of the {bnode/ label/ uri/ long-uri/ long-label/ datetime/ datatype/ numeric/ languages}, asserted into store as a part of statements
Value	bigint(20)	Hash of the {bnode/ label/ uri/ long-uri/ long-label/ datetime/ datatype/ numeric/ languages}, asserted into store as a part of statements
Primary key: id		

Table 12: Namespace_prefixes table description

Namespace_prefixes		
Column	***Type***	***Description***
Prefix	**varchar (127)**	Prefix of the dataset
namespace	**Text**	Corresponding namespace of dataset

Table 13: Locked table description

Locked		
Column	***Type***	***Description***
process	varchar(128)	Process that has locked the database

Sesame$_{RDB}$ an *id-based approach* for triple storage that requires some additional tables i.e. uri_values, label_values, bnode_values, long_uri_values, and long_label_values, for storing one to one mapping between lexical values and corresponding identifiers. Whenever a triple is inserted into Sesame$_{RDB}$ each node of the triple is assigned an id on the basis of its corresponding category i.e. uri, label, bnode, long_uri, and long_label; and the nodes are inserted into that corresponding table and triple is inserted into corresponding property table or in triples table in case of schema-aware or schema-oblivious approach respectively. Hash of each asserted node is computed and inserted into hash table along with its corresponding id. Hash values are used when looking up internal Ids from known terms. The uri_values, label_values, bnode_values, long_uri_values, and long_label_values tables store the URIs or literal text in value column, these cannot be indexed (in many databases), and so another index-able column is needed to lookup the internal ids for the text of URIs or literals. The hash value maps a globally unique 64 bit hash of the term to a local internal 32 bit id.

Literal values of 'label_values' may have some rdf/xml properties such as datatype, datetime, numeric and language. These literals along with their corresponding ids and values are also inserted into the corresponding tables that are datatype_values, datetime_values, numeric_values and language_values, created as a part of schema design of the Sesame$_{RDB}$ database. Namespace_prefixes table contains the namespace of the dataset and locked table contains the entry for each process that has currently locked the sesame database.

Sesame$_{RDB}$ creates four indices on each property table, two indices on hash table and one index on each remaining table (other than locked and namespace_prefixes). Detailed Description of each index is shown in Table 14.

Table 14: Indices on tables in SesameRDB Layout

<table>
<tr><th colspan="4"><b>index on property_table</b></th></tr>
<tr><th><i>Key_name</i></th><th><i>Column_name</i></th><th><i>Non-unique</i></th><th><i>Index type</i></th></tr>
<tr><td>Primary</td><td>ctx,subj,obj,expl</td><td>Unique</td><td>Btree</td></tr>
<tr><td>tableName_subj_idx</td><td>subj</td><td>Non-unique</td><td>Btree</td></tr>
<tr><td>tableName _ctx_idx</td><td>ctx</td><td>Non-unique</td><td>Btree</td></tr>
<tr><td>tableName _expl_idx</td><td>expl</td><td>Non-unique</td><td>Btree</td></tr>
<tr><th colspan="4"><b>Index on corresponding table</b></th></tr>
<tr><th><i>Key_name</i></th><th><i>Column_name</i></th><th><i>Non-unique</i></th><th><i>Index type</i></th></tr>
<tr><td>Primary</td><td>id</td><td>Unique</td><td>Btree</td></tr>
<tr><th colspan="4"><b>Index on hash_values table</b></th></tr>
<tr><th><i>Key_name</i></th><th><i>Column_name</i></th><th><i>Non-unique</i></th><th><i>Index type</i></th></tr>
<tr><td>Primary</td><td>hash</td><td>Unique</td><td>Btree</td></tr>
<tr><td>HASH_VALUES_value_idx</td><td>value</td><td>Non-unique</td><td>Btree</td></tr>
</table>

3.2.3. In-Memory Stores

In-memory Semantic Web databases store and manage data in main memory they deploy different in memory data structures for efficient retrieval and storage of triples. A brief description of two in-memory Semantic Web databases is given in this section.

3.2.3.1.Sesame$_M$

Sesame$_M$ stores and manipulates Semantic Web data in in-memory. It uses a bipartite graph representation for triples in main memory. Vertices of this bipartite graph representation are divided into two parts. First part is a combination of statement nodes, representing the number of triples in the dataset. Second part is a combination of resource nodes, representing all resources, literals and blank nodes. Each statement node references four resource nodes, one reference for each resource role: subject, predicate, object or context. Each resource node also has, for each role it plays, a reference to a list of statement objects in which it plays that particular role. An overview of storage model is shown in Figure 27.

Statements	Subject	Predicate	Object
1	Anila_Sahar	supervisedBy	Sharifullah
2	Sharifullah	supervises	Anila_Sahar
3	DELSA	hasMember	Sharifullah
4	Anila_Sahar	memberOf	DELSA

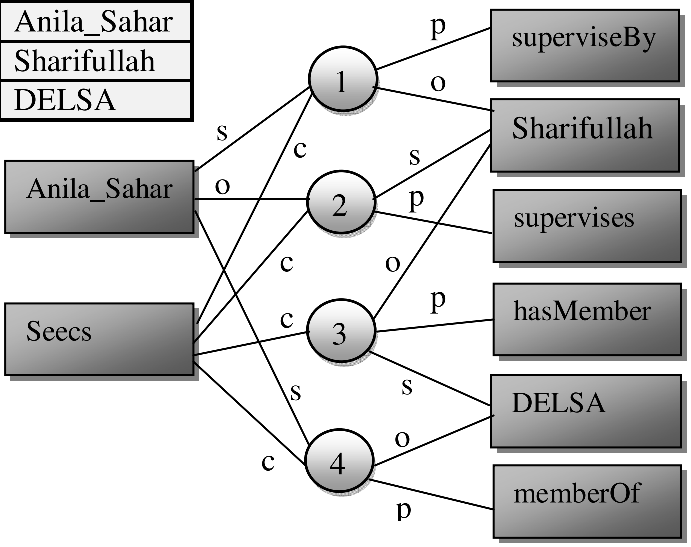


Figure 27: Sesame$_M$ Storage Layout

3.2.3.2.Jena$_{\text{M}}$

Jena$_{\text{M}}$ is another prominent Semantic Web database that stores and manipulates Semantic Web data in main memory. It uses a HashBunchMap for triples storage and retrieval in main memory. At the heart it creates three indexes one for ***subjects to triples***, second for ***predicates to triples*** and third for ***object to triples***. These three indexes map RDF term (subjects, predicates, objects) to triple using a HashBunchMap.

Jena uses its own hash maps which are more compact and faster than the standard Java ones although they provide fewer facilities, just what is needed for RDF indexing in Jena. In fact, the indexes map terms to "bunches" (triples with common indexing term) e.g. a subject index takes all the resources, that appears as a subject in triples, as hash-keys and stores all the triples as value for that key that contains that resource as its subject as shown in Figure 28. This complete set of triples against any key is called a bunch and the bunch is stored as an array if small and a set (using the same code as Jena's hash maps) if larger. Similarly predicate and object indexes store triples with predicates and objects values as keys and store triples with having these keys as shown in Figure 29 & Figure 30.

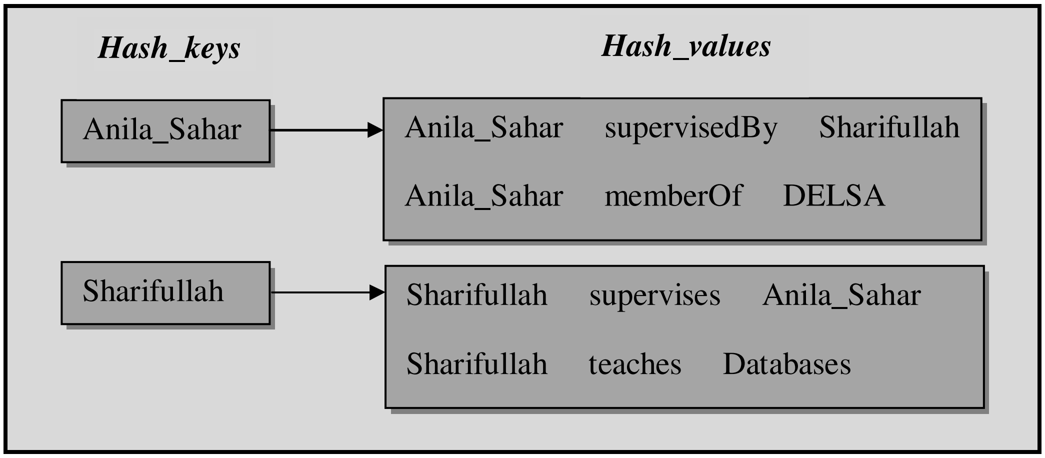

Hash_keys
Hash_values
Anila_Sahar
Anila_Sahar supervisedBy Sharifullah
Anila_Sahar memberOf DELSA
Sharifullah
Sharifullah supervises Anila_Sahar
Sharifullah teaches Databases

Figure 28: Jena Subject Index Structure

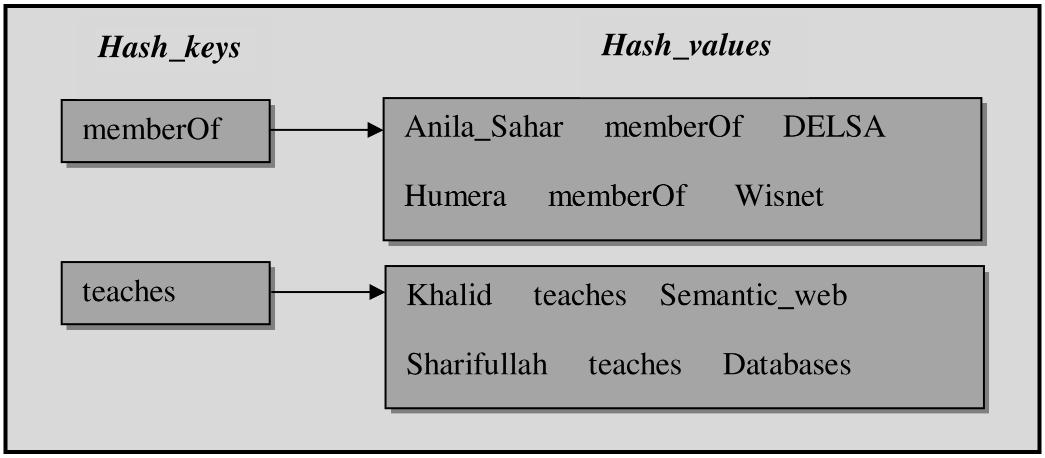

Hash_keys
Hash_values
memberOf
Anila_Sahar memberOf DELSA
Humera memberOf Wisnet
teaches
Khalid teaches Semantic_web
Sharifullah teaches Databases

Figure 29: Jena Predicate Index Structure

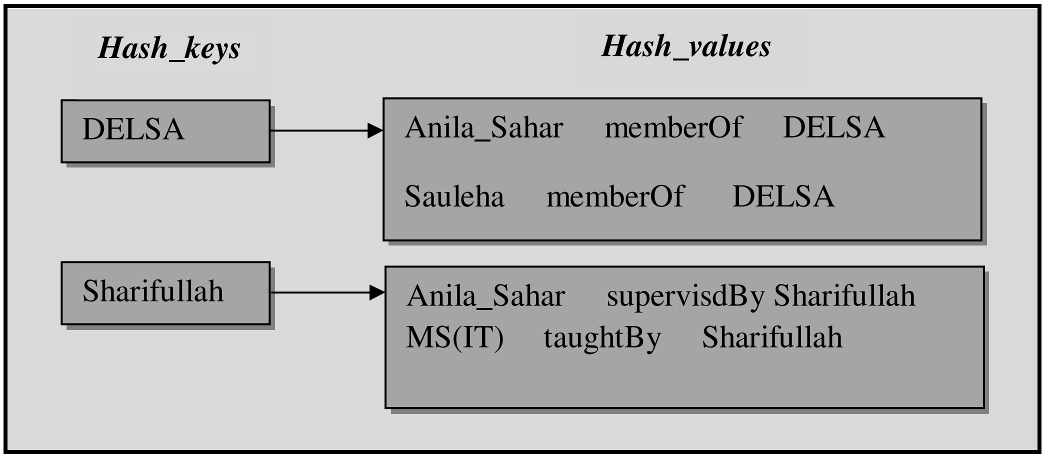

Hash_keys
Hash_values
DELSA
Anila_Sahar memberOf DELSA
Sauleha memberOf DELSA
Sharifullah
Anila_Sahar supervisdBy Sharifullah
MS(IT) taughtBy Sharifullah

Figure 30: Jena Object Index Structure

Summary of Chapter

This chapter documents the architectural study and features comparisons of these stores. Following this, we presented storage architecture and access mechanism of different open source Semantic Web databases. For describing the Semantic Web databases' storage layouts we surveyed the past approaches used by selected Semantic Web databases and presented their common approaches to store the data e.g. id-based vs. value-bases approach, schema oblivious vs. schema aware approach, and prefix compression. We also described the access mechanism of these databases i.e. we detailed type and number of indexes, value-to-Id, and Id-to-value mapping techniques

CHAPTER 4

SEMANTIC WEB DATABASES EVALUATION FRAMEWORK

This chapter provides the detail of *evaluation framework* for our research. It starts with the introduction of Semantic Web databases that we evaluated followed by logical schema of dataset used for evaluation. Last but not the least section describes the detailed methodology for the comparative and scalability analysis of Semantic Web databases.

4.1. Semantic Web databases

We have analyzed and evaluated all three types of Semantic Web databases as given in Figure 31. Semantic Web databases evaluated in this research includes $Jena_M$ [57], $Sesame_M$ [60], *TDB* [59], *SDB* [58], $Sesame_N$ [60], $Sesame_{RDB}$ [60], and *AllegroGraph* [37].

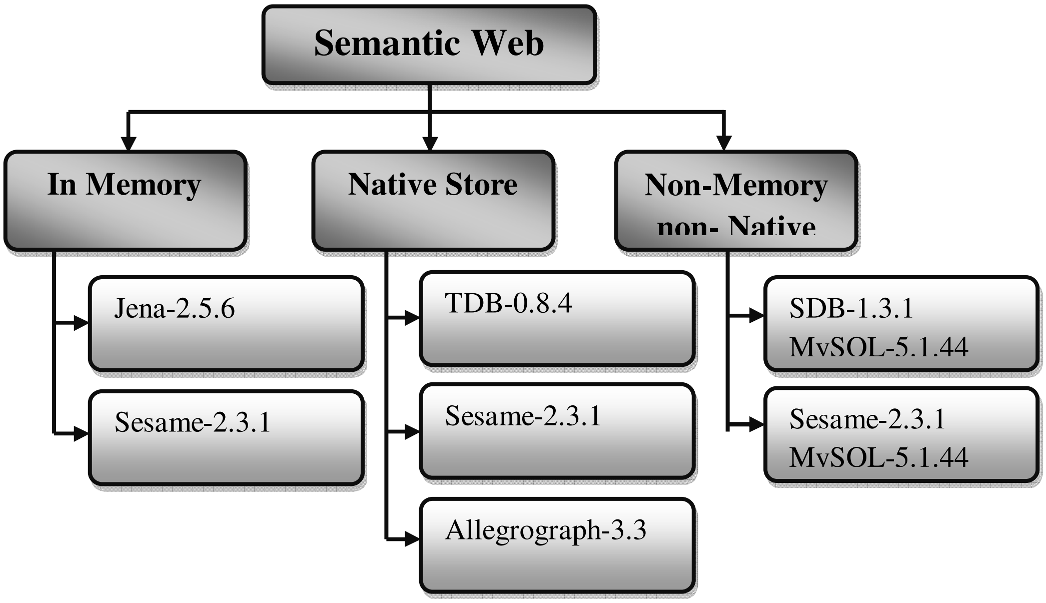


Figure 31: Evaluated Semantic Web databases

All these Semantic Web databases are quite popular and frequently used for performance evaluation and benchmarking of RDF storage architecture in Semantic Web research community. Figure 31 presents the category and version of each store.

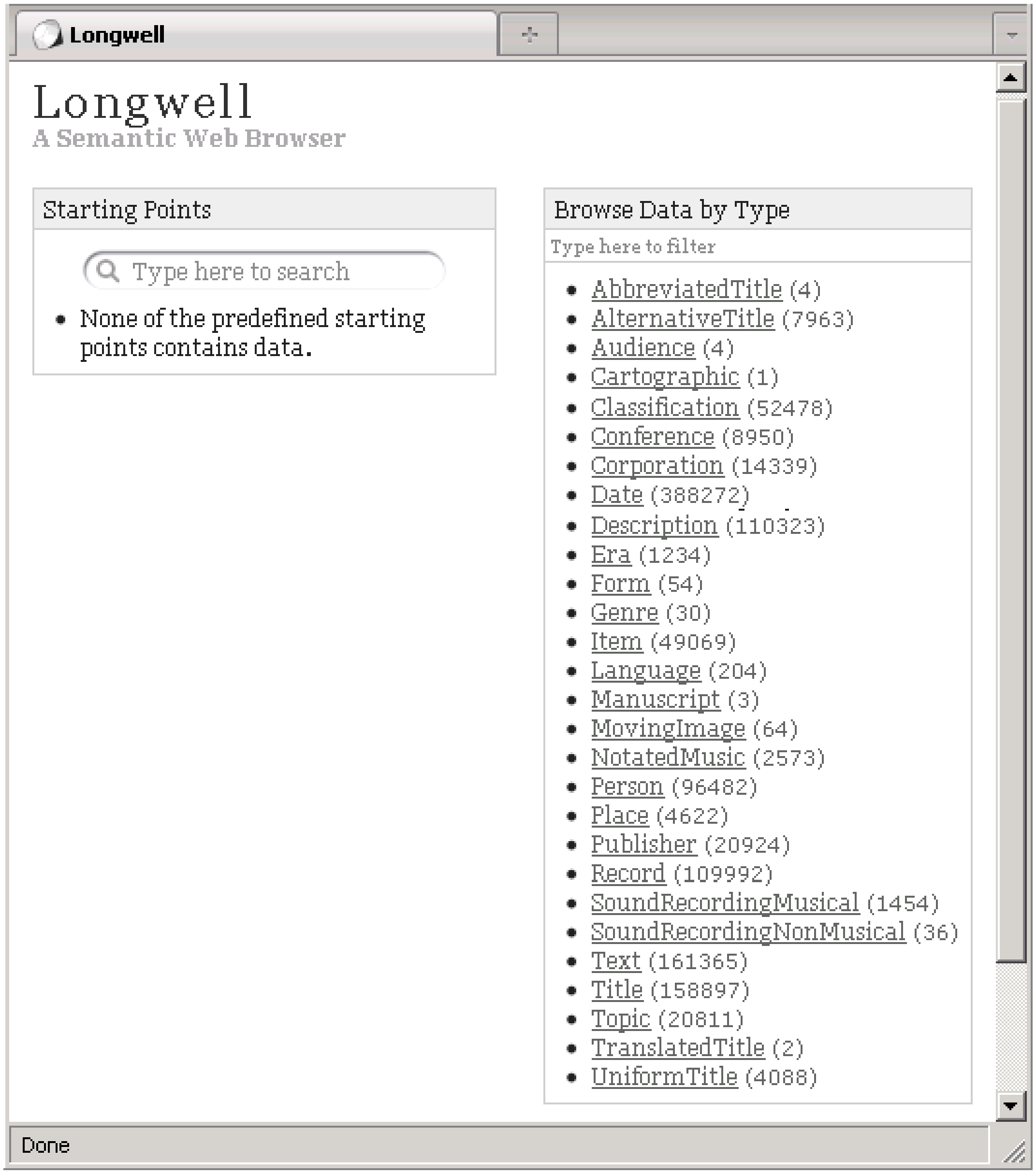
4.2. Evaluation Dataset

The objective of this research is performance evaluation of scalable Semantic Web databases. A real dataset allows realistic and accurate quantification of Semantic Web databases. Therefore, we are interested in using a real and public data set to evaluate the performance of Semantic Web databases. Different RDF datasets are available online for testing purpose, such as, Barton libraries [29], DBpedia [52] and DBLP [51]. They are amongst the most commonly used RDF datasets for Semantic Web databases evaluation.

The Barton Libraries dataset [29] is used for the performance evaluation. This data is provided by the Simile Project [53], which develops tools for library data management and interoperability. The data contains records that compose an RDF-formatted dump of the MIT Libraries Barton catalog. This dataset has been used as a standard data set for Semantic Web analysis. Semantic Web data has two basic characteristics (a) large size of data and (b) irregular structure of data. The presence of both the characteristics at the same time makes a dataset true representation of Semantic Web data that affects the performance of Semantic Web databases. The analysis performed only on a large dataset, does not present true picture of the performance of Semantic Web databases. While selecting the dataset, we made tradeoff on dataset size and dataset structure and selected it for our evaluation. The Barton Library dataset was derived from multiple sources and it follows a semantically rich ontology. It preserves the irregular structure of data and it is one of the large size data currently available online.

4.2.1. Barton Libraries Dataset

The dataset is analyzed using longwell [54], a faceted browser that combines the flexibility of RDF data model with user interface paradigm. Data is categorized on the basis of the types of resources present in the dataset. The index page of longwell is shown in Figure 32. The right panel shows Barton dataset types arranged in alphabetical order. The integer number with each type gives instance count of that particular type.



Longwell

A Semantic Web Browser

Starting Points

Type here to search

- None of the predefined starting points contains data.

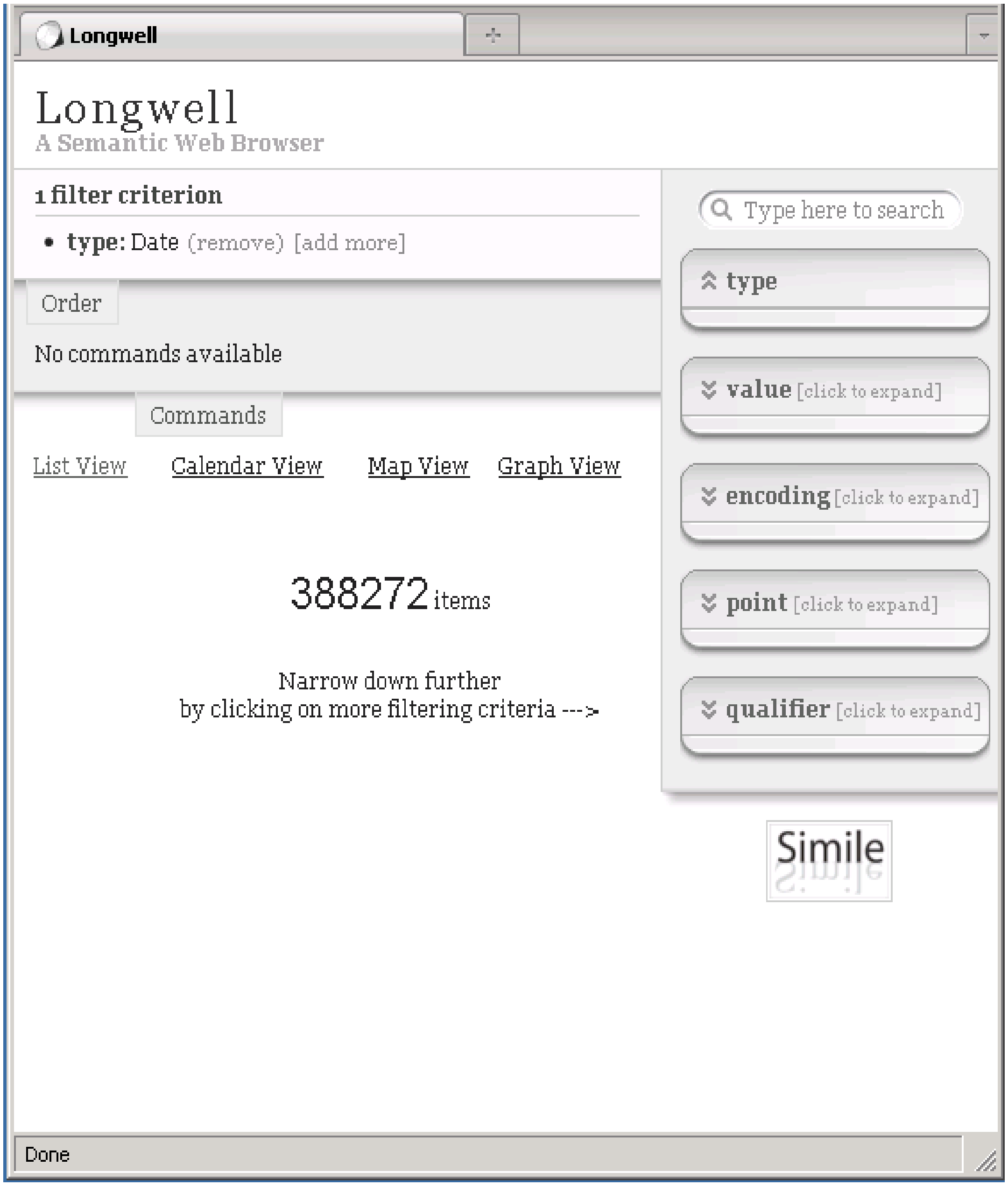
Browse Data by Type

Type here to filter

- AbbreviatedTitle (4)
- AlternativeTitle (7963)
- Audience (4)
- Cartographic (1)
- Classification (52478)
- Conference (8950)
- Corporation (14339)
- Date (388272)
- Description (110323)
- Era (1234)
- Form (54)
- Genre (30)
- Item (49069)
- Language (204)
- Manuscript (3)
- MovingImage (64)
- NotatedMusic (2573)
- Person (96482)
- Place (4622)
- Publisher (20924)
- Record (109992)
- SoundRecordingMusical (1454)
- SoundRecordingNonMusical (36)
- Text (161365)
- Title (158897)
- Topic (20811)
- TranslatedTitle (2)
- UniformTitle (4088)

Figure 32: Longwell Screenshot

By clicking on any type, facets are shown to further drill down the data. The facets of each class represent the predicates that are associated with resources of class type. Figure 33 shows the facets for *Date* type resources.



Longwell

A Semantic Web Browser

1 filter criterion

- type: Date (remove) [add more]

No commands available

388272 items

Narrow down further
by clicking on more filtering criteria --->

Figure 33: Longwell screen shot of facets for *Date* type resources

An overview of the Barton data model, analyzed through longwell, is presented for the better understanding of dataset in Figure 34.

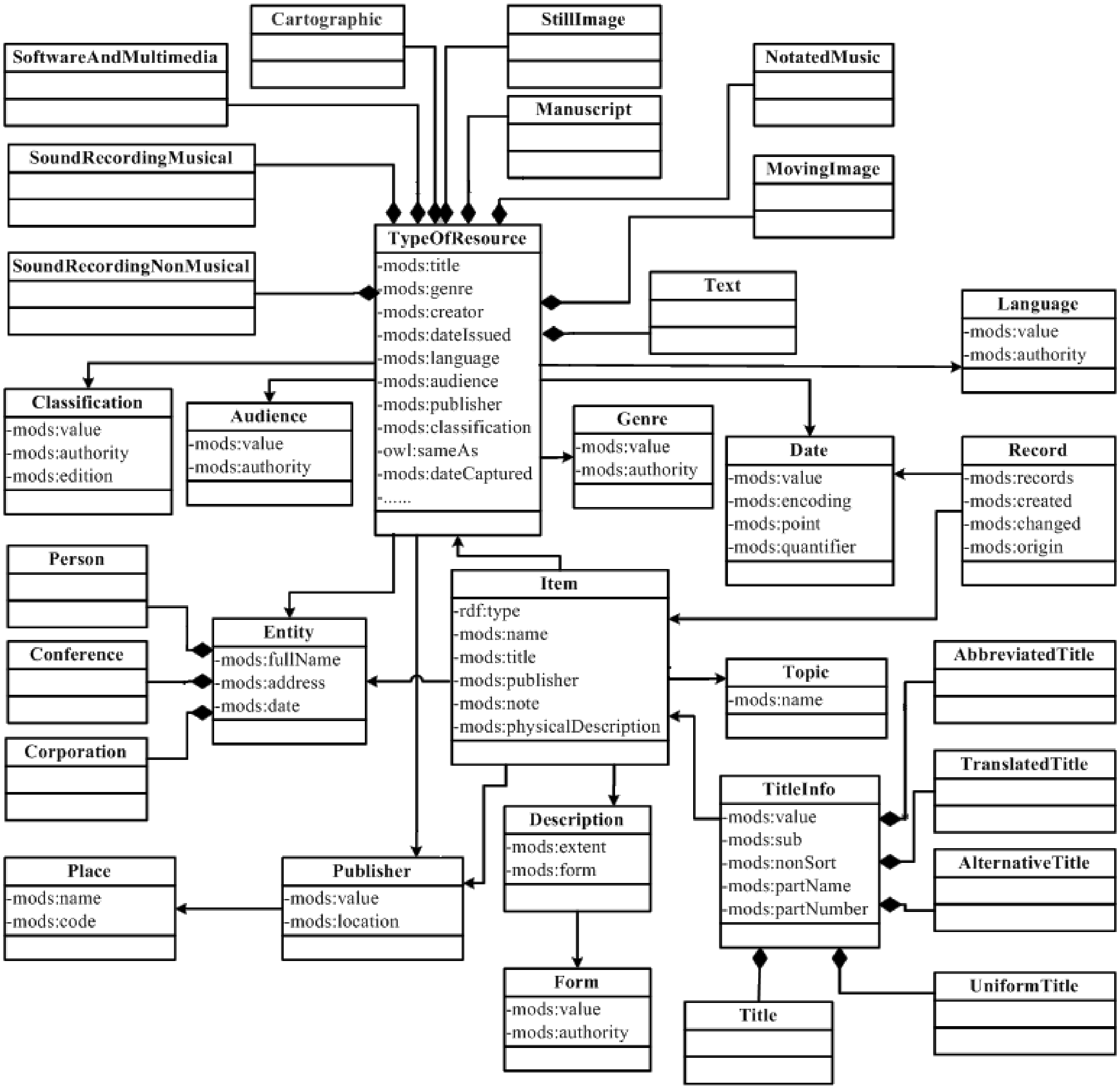


Figure 34: Overview of Barton Data Model

In Figure 34 classes represent the types of resources in dataset, attributes of each class presents all those predicates whose domain is this class. The relationship of a class with other classes shows the range of the predicates of that class. A brief description of classes is given in Table 15. A detailed description of attributes and their associated object type are given in [58].

Table 15: Barton Dataset Classes Description

CLASSES DESCRIPTION	
Class Name	**Definition**
Record	Record present information about the metadata record
Item:	Item is a resource that is being described
TitleInfo	TitleInfo is an abstract class represents all those words, phrases, characters, or group of characters, that constitutes the chief title, abbreviated title, translated title, alternative title and uniform title of a resource.
Topic	Topic class models all those subjects of resources that are not appropriate under title class
Date	Date class represents the information about date on which a record is created, changed, issued, and copyrighted or any other date that needs to be specified
Description	Description may be used to give a textual description for a resource when necessary
Form	Forms provides the information about the designation of physical presentation of the resource
Publisher	Publisher is the entity that published, printed, distributed, released, issued, or produced the resource
Place	Place describes the all those places that are associated with the issuing, publication, release, distribution, manufacture, production, or origin of a resource
Entity	Entity class represents all those persons, corporations and events (e.g. conference) who can be related to a resource in some way
Language	language class provides all those languages in which contents of the resources of dataset is expressed

Audience	Audience class provides a description of the intellectual level of the audience for which the resource is intended
TypeOfResource	Type of resource defines the term that specifies the characteristics and general type of content of the resource. Type of resource may be from one of text, cartographic, notated music, sound recording musical, sound recording nonmusical, still images, moving images, software and multimedia, and manuscripts
Classification	Classification class indicates all those categories in which resources can be organized according to subject area

4.2.1.1.Statistics of Barton Dataset:

A detail analysis of Barton dataset provides us the characteristics of evaluated data that are presented in Table 16. There are slightly more than 25 million triples in dataset, previously it was claimed that this dataset contains 50 million triples [62].

Table 16: Summary Statistics of Barton Dataset

Dataset Characteristics	
Total Number of Triples	25176626
Total Nodes	9716253
Total types of Instances	30
Total Unique Properties	199
Multi-valued Properties[11]	72
Single-valued Properties[12]	127

[11] Multi-valued properties means these properties appear more than once for a given subject

[12] Single-valued properties means these appear only once for a given subject

4.2.1.2.Namespaces

Similar string at the start of properties of dataset belongs to some predefined schemas, such as RDF, OWL or some other schemas. These are declared as namespaces in the document. URIs or namespaces for the Barton dataset are given in Table 17.

Table 17: Prefix to URI mapping

Prefix	URI
modsrdf:	http://simile.mit.edu/2006/01/ontologies/mods
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
role:	http://simile.mit.edu/2006/01/role/
owl:	http://www.w3.org/2002/07/owl#

4.2.1.3.Dataset Scaling and population

We performed our evaluation on four different sizes of Barton dataset. Dataset 1, dataset 2, dataset 3 and dataset 4 contains 0.2M, 1 Million, 5 Million and 25 Million triples respectively. As we sampled our datasets from a large dataset, we tried to make our sampling fair and realistic. For fair sampling, we selected ten different samples of each dataset 1, dataset 2 and dataset 3. Average of ten samples for each dataset represents its population. Detail of each dataset is given in Table 18.

Table 18: Dataset Scaling and Population

Class Name	Scaling Factor			
	Dataset 1 (200K)	Dataset 2 (1Million)	Dataset 3 (5Million)	Dataset 4 (25Million)
Number of Date	15697	78382	391951	1959758
Number of Title	6453	32706	163502	817508
Number of Text	5499	30400	152230	760564
Number of Description	4421	21760	108802	544011
Number of Record	4430	20569	102813	514067
Number of Classification	3219	15538	77588	387942
Number of Person	2856	14198	90561	353635
Number of Item	2516	12511	62671	312270
Number of Alternative Title	780	3810	19098	95489
Number of Publisher	746	3852	23698	89589
Number of Corporation	485	2345	15838	58639
Number of Topic	459	2100	22804	52681
Number of Uniform Title	286	1162	5736	28679
Number of Conference	237	1125	5702	28139
Number of Place	189	853	5103	15542
Number of Notated Music	126	716	3008	15016
Number of Sound Recording Musical	91	449	2208	11022
Number of Abbreviated Title	87	498	2167	10835
Number of Cartographic	28	103	477	2353
Number of Manuscript	15	79	351	1753
Number of Moving Image	26	44	222	1109
Number of Language	35	58	178	399
Number of Genre	9	28	111	352
Number of Sound Recording Nonmusical	5	17	57	286

Number of Software and Multimedia	10	15	52	260
Number of Form	6	17	65	242
Number of Translated Title	5	7	16	82
Number of Audience	0	5	17	66
Number of Still Image	0	2	4	18
Others				3653947
Total	48716	243349	1257030	6062306

4.2.1.4.Dataset Cleaner

Publically available dump of Barton libraries dataset contains illegal URIs. Some Semantic Web databases such as AllegroGraph and Mulgara do not allow loading datasets that contains illegal URIs. To load Barton dataset, illegal URIs are identified and transform into legal URIs before testing different stores using this dataset.

4.3. Evaluation Methodology

As we concern with the performance and scalability evaluation of Semantic Web databases. According to the Bondi [49] scalability can be divided into four major categories: load scalability, space scalability, space-time scalability and structural scalability as shown in Figure 35. Our research scope covers the performance evaluation of Semantic Web databases with respect to their space scalability and space time scalability.

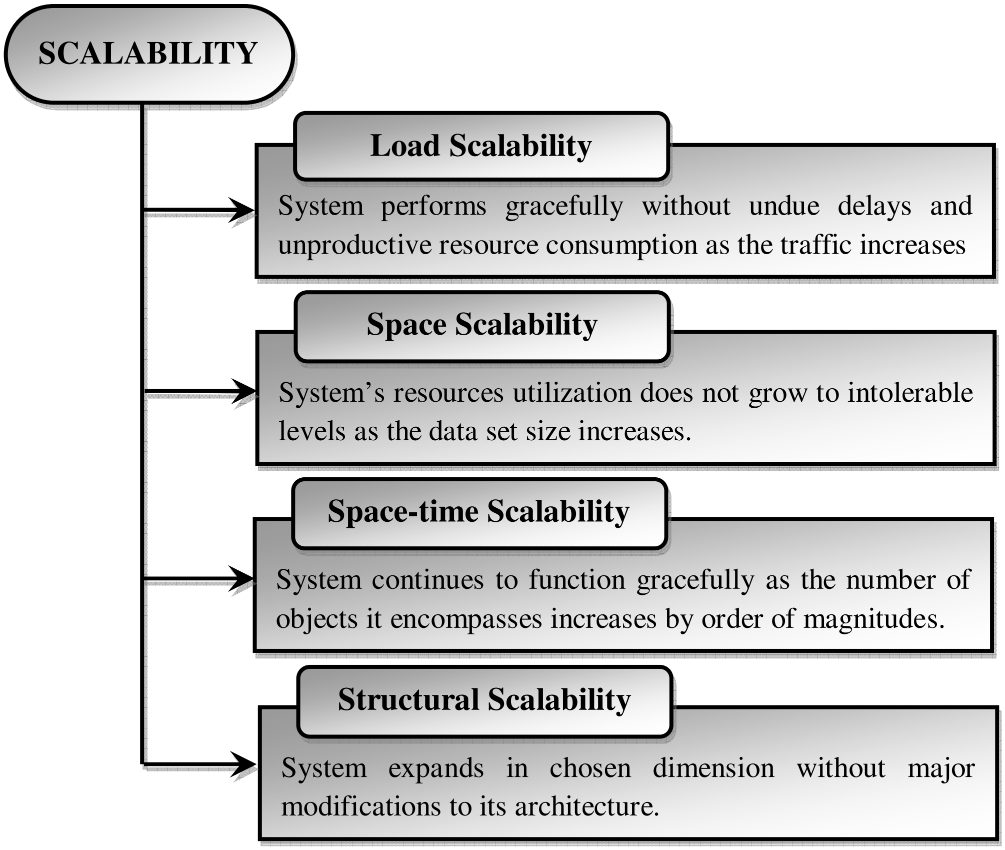

SCALABILITY
Load Scalability
System performs gracefully without undue delays and unproductive resource consumption as the traffic increases
Space Scalability
System's resources utilization does not grow to intolerable levels as the data set size increases.
Space-time Scalability
System continues to function gracefully as the number of objects it encompasses increases by order of magnitudes.
Structural Scalability
System expands in chosen dimension without major modifications to its architecture.

Figure 35: Types of Scalability

We divide our proposed evaluation methodology for the analysis of Semantic Web databases into two parts. The first part describes the test cases designed to check the performance of stores while the second part highlights the proposed performance and scalability metrics for evaluation.

4.3.1. Test Cases

Database community broadly divides all database operations into four major Database communities broadly divides all database operations into four major categories [69], as given below;

1. Load data to a data store (Create)
2. Get existing data from a data store (Read)
3. Modify existing data in a data store (Update)
4. Delete data from a data store (Delete)

These operations are collectively known as CRUD operations. Optimizing a data store for some operations of a category can result in performance degradation of another category's operations. For example multiple indexes can reduce the data access time considerably; but, can degrade the performance of create, delete and update operations. Each addition or deletion requires updating all the indexes so does each update operation also requires changing an index key of indexes [55]. Therefore performance testing results of a data store, for single category, are not reliable. All these operations need to be tested for any data storage system in order to get a clear vision of data store performance. On the basis of this argument, we have divided our test cases into four different categories representing the CRUD operations.

4.3.1.1.Create

Most of the Semantic Web databases support two types of RDF data loading, *bulk load* and *incremental load*; therefore, test cases are designed to test stores' performance for both data loading types.

- *Bulk Load:* This test case measures a Semantic Web database's performance while operating in bulk load. Cost i.e. execution time and system resources utilization, of bulk load is an additive cost of creating new store, their appropriate indexes and data set load cost.

$$C_{BulkLoad} = C_{RepositoryCreation} + C_{IndexCreation} + C_{DataLoading}$$

- *Incremental Load:* This test case evaluates a Semantic Web database's performance when loading some statements into an existing store at the cost of updating the indexes.

$$C_{IncrementalLoad} = C_{DataLoading} + C_{IndexUpdation}$$

4.3.1.2.Read

As mentioned before, RDF data query patterns are unpredictable, so we cannot define the test cases for the most anticipated query patterns. Some general query patterns are defined in [27] also giving rise to inspiration to design some test cases. Our test cases for read operations measure the Semantic Web databases performance for their *data access patterns* and *query optimization techniques.*

Moreover, we have designed queries for all test cases and strive best to evaluate each test case independently, out of the effect of all other test cases.

- *Simple query performance:* This test case returns a result set against some specified subject, predicate or objects value. The test case evaluates the storage and indexing mechanism of the store. A store that is indexed or stored on the basis of anyone within subject, object and predicate, will show comparatively decreased performance on other two. See Appendix (SimpleQuery)

- *Complex query performance:* This test case has been designed to evaluate the effect of query complexity on the performance of Semantic Web database. It is hard to define discrete difference between a simple and a complex query, but a query that involves more number of triple patterns is considered to be a complex query [27]. Two patterns for complex queries we considered are *Long pattern* and *bushy patter*. In *Long Pattern*, RDF resources are related to each other through a long path and whereas in *bushy pattern,* a single resource is linked to many other resources. See Appendix (ComplexQuery)

- *Queries with different result sizes:* It has been observed that time and resource utilization increases with the increase in the result size; a query returns. Therefore a separate test case is designed especially to analyze the effect on store's performance with the increase in result size. Queries for this test case will return different result sizes to check the performance. A store where hash or node ids are used to store triples and actual node values are stored somewhere else. It would require more time while retrieving the final results while mapping node ids to node values. See Appendix (ResultSize)

- *Query with different join selectivity:* A good query optimizer executes join first that has highest selectivity as compared to other joins in the query. This test case determines the query optimizer's intelligence to create an optimal query plan. Therefore, a query to test this test case has different selectivity for each join. Selectivity of each join is considered as high or low with respect to other joins selectivity of query. See Appendix (SelectivityEstimation)

- *Queries having irregular access patterns:* This test case is designed to assess the irregular access patterns for the data access. This is the characteristics of Semantic Web query, where data is mostly accessed irregularly. See Appendix (IrregularAccessPatterns)

4.3.1.3.Update

We employ a simple test case to determine the update performance of a Semantic Web database. Updating a triple would also result in updating the index entry for that particular triple.

- Update some statements in the existing store: This test case examines the performance of triple stores while updating the existing triples in the store. This test case also determines the triple store's criteria of maintaining its versioning information and updating the triple at the cost of updating its indexes.

$$C_{Update} = C_{TripleUpdate} + C_{IndexUpdate}$$

4.3.1.4.Delete

Semantic Web databases usually provide two types of deletion i.e. *deletion of entire store* and *deletion of some statements from store*. Two test cases are designed to check the deletion performance of the Semantic Web databases.

- *Deletion of entire store:* The test case evaluates the deletion performance for single model Semantic Web databases. Deletion of an entire store is simple if Semantic Web database supports only a single model or it does not support the cross model inferences.

- *Deletion of some statements from a store:* Deletion of individual statements is relatively simple if a Semantic Web database supports RDF-only stores. In case of RDFS and OWL, if a store supports forward chained entailments, it becomes very complex because deletion of single statement needs to alter the inferred statement.

4.3.2. Performance & Scalability Metrics

In this thesis, a set of performance and scalability metrics are proposed that captures various aspects of the space scalability and space time scalability evaluation of Semantic Web databases. While evaluating the performance of a Semantic Web database, we consider two important parameters as a measure of its execution cost for each proposed test case, i.e. *execution time* and *resource usage*. Depending upon the scenarios we will consider *secondary disk space*, *main memory*, and *CPU time* as cost primitives of resource usage to have an in-depth study of performance. The execution cost of operation on a Semantic Web database is influenced by a number of factors, including number of nodes in dataset, number of triples in dataset and type of operation [68]. Therefore we record the execution cost for different operations and dataset sizes. Table 19 presents corresponding cost primitives for each type of operation and for all datasets.

Table 19: Test case categories with their corresponding performance metrics

Metric Test Cases	Load Time	Query Execution Time	Repository Size	Main Memory	CPU	Success Ratio	Cumulative Query Response
Create	√	×	√	√	√	√	×
Read	×	√	×	√	√	√	√
Update	×	√	×	√	√	×	×
Delete	×	√	×	√	√	×	×

4.3.2.1.Load Time:

The load time metric reports the loading time for datasets of different sizes. Load time is measured as a commutative time to build a repository structure, build initial index structures and generate statistics about the dataset for query optimization.

$$T_{Load} = T_{RepositoryCreation} + T_{loadDataset} + T_{IndexCreation} + T_{StatisticsGenration}$$

4.3.2.2.Query Execution Time

This metric provides the query execution time for each test on datasets of varying sizes. In order to eliminate the query cache effect, query execution time is computed at three different instances followed by their mean and uses this mean value as a measure of query execution time. Query execution time includes the time to connect the repository, execute query, print result set and then close connection.

$$T_{QueryExecution} = T_{OpenConnection} + T_{ExecuteQuery} + T_{PrintResultSet} + T_{CloseConnection}$$

4.3.2.3.Repository Size

This metric presents the detail of the storage size occupied by a dataset after loading it into a persistent storage. Repository size is a composite figure of total size of all files present in the repository including data files and index files. For main memory systems we also compute the largest dataset size handled by each system on fixed memory platforms. The main memory Semantic Web database that handles the largest dataset size, consumes memory more efficiently.

$$S_{Repository} = S_{DataFiles} + S_{IndexFiles}$$

4.3.2.4. Main Memory

This metric computes the memory requirement for the execution of a particular test case. This memory is the amount of memory needed to hold the working buffers. Size of main memory, hardly few GBs, is a limit on the size of the semantic data that can reside or process in main memory [56]. A Semantic Web database consuming memory more efficiently is supposed to be more scalable and efficient.

4.3.2.5. CPU time

This metric shows the CPU time consumed for processing a particular test case. Whenever a test case is executed by a user, it engages some of the system resources among which CPU time utilization is the most critical. CPU time utilization is important because the higher the percentage of the CPU used by a Semantic Web database the less power the CPU can devote to other tasks. *User CPU Time* and S*ystem CPU Time* is computed for in-depth analysis.

4.3.2.6. Success Ratio

Success ratio describes the fraction of successfully executed test cases either for a semantic store or for a dataset. For success ratio computation, we considered only bulk load test case and six read test cases (i.e. total seven test cases on a dataset) to develop a clear effect of failed tests. For each store and dataset size, success ratio is calculated using formulae given in Equation 1.

$$\mathrm{SRstore} = \frac{\sum_{\mathrm{i}=1}^{4} \mathrm{ST}_{\mathrm{dataset(i)}}^{\mathrm{store}}}{\sum_{\mathrm{i}=1}^{4} \mathrm{TT}_{\mathrm{dataset(i)}}^{\mathrm{store}}}$$

(a)

$$\mathrm{SRdataset} = \frac{\sum_{\mathrm{i}=1}^{4} \mathrm{ST}_{\mathrm{store(i)}}^{\mathrm{dataset}}}{\sum_{\mathrm{i}=1}^{4} \mathrm{TT}_{\mathrm{store(i)}}^{\mathrm{dataset}}}$$

(b)

Equation 1: Success Ration Computation Formula for (a) Semantic Store (b) Dataset

Where

$\mathrm{SRstore}$ = Success Ratio of store

$\mathrm{ST}_{\mathrm{dataset(i)}}^{\mathrm{store}}$ = Number of successful test cases on dataset(i) for this store

$\mathrm{TT}_{\mathrm{dataset(i)}}^{\mathrm{store}}$ = Number of total test cases on dataset(i) for this store

$\mathrm{SRdataset}$ = Success Ratio of dataset

$\mathrm{ST}_{\mathrm{store(i)}}^{\mathrm{dataset}}$ = Number of successful test cases on store(i) for this dataset

$\mathrm{TT}_{\mathrm{store(i)}}^{\mathrm{dataset}}$ = Number of total test cases on store(i) for this dataset

4.3.2.7. Cumulative Query Response Time

Cumulative performance metric describes three averages i.e. mean query response time, mean main memory usage and mean CPU time for read test cases. Computations results have huge variations that necessitate the selection of geometric means over arithmetic means. As our cumulative performance metrics preserve the performance behavior of each store, so they show that our choice for computing geometric mean is quite appropriate. For all failed test cases we considered large values for response time as 86400sec, main memory as 6144 MB and CPU time as 60 sec to penalize the respective store.

Summary of Chapter

In this chapter we discussed the Semantic Web databases evaluation framework in detail along with introduction to some prevalent Semantic Web databases that are being evaluated. We describe the logical schema of Barton dataset in detail to evaluate previously mentioned Semantic Web databases. Dataset description provides an insight to the Barton data model and dataset statistics along with description of its important classes. The last section provides details of our proposed evaluation methodology that includes evaluation framework, test cases description and proposed performance and scalability metrics.

In next chapter, we will present an overview of the Semantic Web databases architecture.

PERFORMANCE EVALUATION RESULTS

This chapter documents the outcomes of our research**.** The chapter is divided into three parts. The first part describes the configuration setting of evaluation both for machine and Semantic Web databases. The second part presents the comparative performance evaluation results of tested Semantic Web databases while the third part presents their scalability analysis.

5.1. Evaluation Configuration

This section describes experimentation set up including hardware and software where all the test cases were executed followed by the configuration set up for each Semantic Web database. All the results for comparative evaluation and scalability analysis are dependent upon these configuration settings.

5.1.1. Test Environment

All experiments were performed on a single machine under 64bit Enterprise edition of Microsoft Windows Server 2003 on the top of ACPI Multiprocessor (16 processors) X5550@ 2.67GHz CPU. We used 8 GB RAM and 120 GB PERC 6/I SCSI Disk Device with 8GB virtual memory. All Java engines were executed with Netbeans IDE 6.8 that run on the top of JRE 1.6.0_18 64-bit version. RAM Rush 1.0.5.817 was used as a RAM cleaner.

5.1.2. Semantic Web Databases Configuration

All of tested stores were run using its available Java APIs. We configured these stores to perform up to our known best performance level of each stores. Detailed configuration of each store along with its tested version is given below.

TDB Version 0.8.4

The statistics-based BGP optimizer was used by generating the stats.opt file and copying it to the database location. All three indexes for triple graph i.e. SPO, POS, OSP were created. Java options of Tomcat were set to -Xmx6144m.

Sesame$_N$ Version 2.3.1

Store type was set to “Native”. Indexes created include SPOC, POSC, and OSPC. Java options of Tomcat were set to -Xmx6144m.

AllegroGraph Version 3.3

Open source copy of AllegroGraph was used. All six index flavors (spogi, gspoi, gposi, gopsi, ospgi, posgi) provided by AllegroGraph were created. It offers two optimization parameters i.e. “Default Expected Resources” that was set according to dataset size and “Chuck size” was configured to 4 GB. Java options of Tomcat were set to -Xmx6144m.

SDB Version 1.3.1

SDB was configured with MySQL version 5.1.44 with hash layout. In MySQL, “innodb_buffer_pool_size” was tuned to 4096M for performance improvements. Java options of Tomcat were set to –Xmx3072m.

Sesame$_{RDB}$ Version 2.3.1

Store type was set to “RDB”. It was configured with MySQL version 5.1.44. In MySQL, “innodb_buffer_pool_size” was tuned to 4096M for performance improvements. Java options of Tomcat were set to –Xmx3072m.

Sesame$_M$ Version 2.3.1

Store type was set to “Memory”. Java options of Tomcat were set to –Xmx6144m.

Jena$_{\text{M}}$ Version 2.5.6

Jena memory model was created. Java options of Tomcat were set to –Xmx6144m.

5.2. Comparative Evaluation Results

This section provides detailed analysis of test results performed for comparative evaluation. To record the results for a test case against its complement performance metrics, we run that test cases on ten different samples for each dataset and considered the average. After executing each single test, we restarted our machine and cleared the RAM using RAM Rush 1.0.5.817.

5.2.1. Success Ratio Results

Table 20: Semantic Web databases and datasets success ratio

Stores	***Datasets***								***SR*** $_{\textit{Store}}$
	Dataset1 (0.2M)		*Dataset2 (1M)*		*Dataset3 (5M)*		*Dataset4 (25M)*		
	S	*F*	*S*	*F*	*S*	*F*	*S*	*F*	
Sesame$_{\text{M}}$	7	0	7	0	7	0	0	7	0.75
Jena$_{\text{M}}$	7	0	7	0	6	1	0	7	0.71
AllegroGraph	7	0	7	0	6	1	4	3	0.86
Sesame$_{\text{N}}$	7	0	7	0	6	1	5	2	0.89
TDB	7	0	7	0	7	0	5	2	0.93
SDB	7	0	7	0	6	1	5	2	0.89
Sesame$_{\text{RDB}}$	7	0	7	0	6	1	5	2	0.89
SR$_{\text{Dataset}}$	1.0		1.0		0.89		0.49		

Table 20 enlists the success ratio for each store and each dataset. Success ratio of both in-memory stores S_{SesameM} = 0.75 and S_{JenaM} = 0.71 shows that they are less successful than the other stores, because the 25 Million triples dataset fails to load in main memory for both these stores. RDB backed stores success ratio i.e. S_{SDB} = 0.89 and $Ssesame_{RDB}$ = 0.89 showed the same success rate. From native stores TDB with success ratio S_{TDB} = 0.93 is the most successful store. On the other hand success ratio for Dataset1 and Dataset2 was 100%, but success ratio decreases tremendously for the Dataset3 and especially for Dataset4.

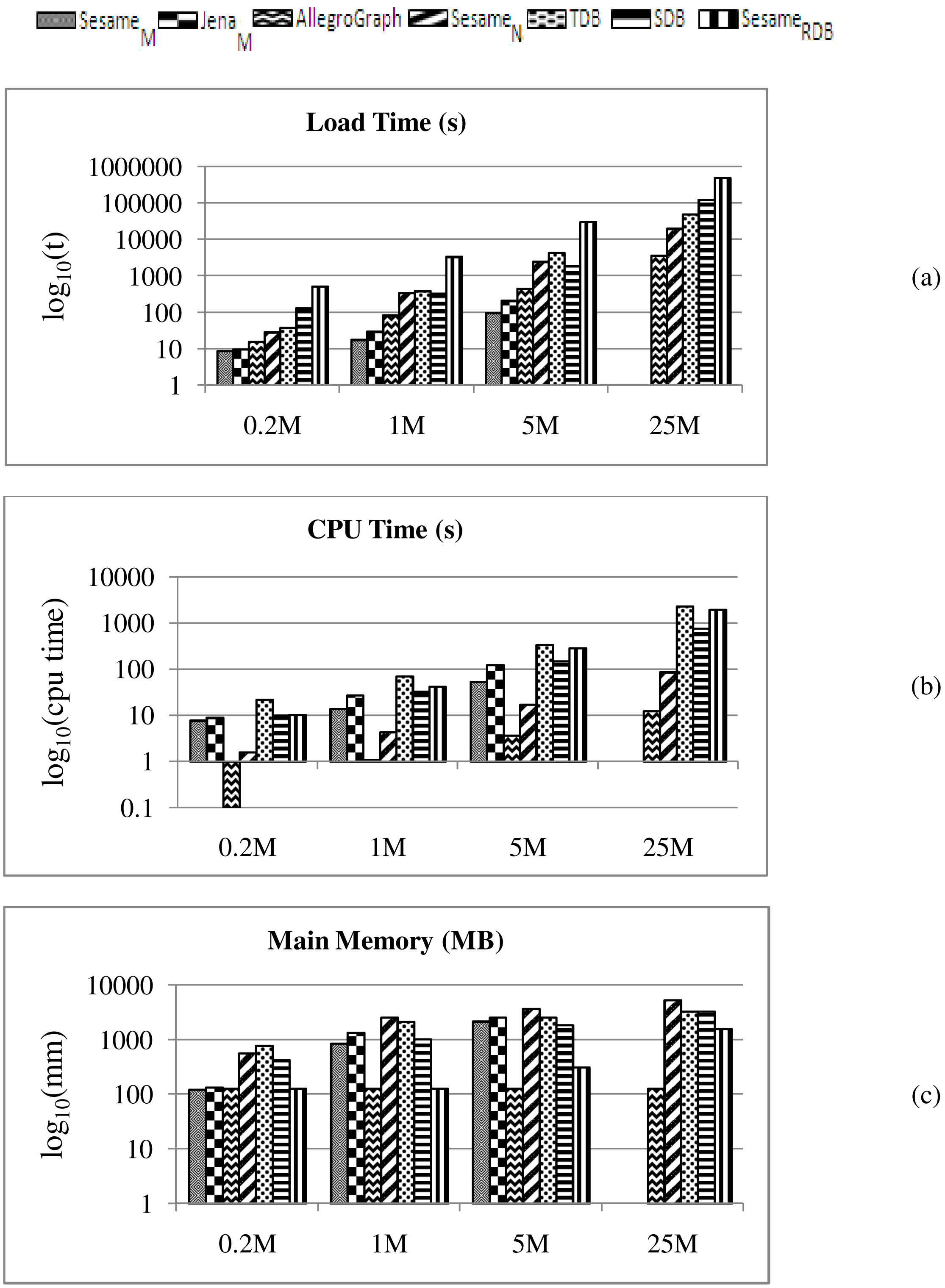
5.2.2. Time and Resource Utilization Results

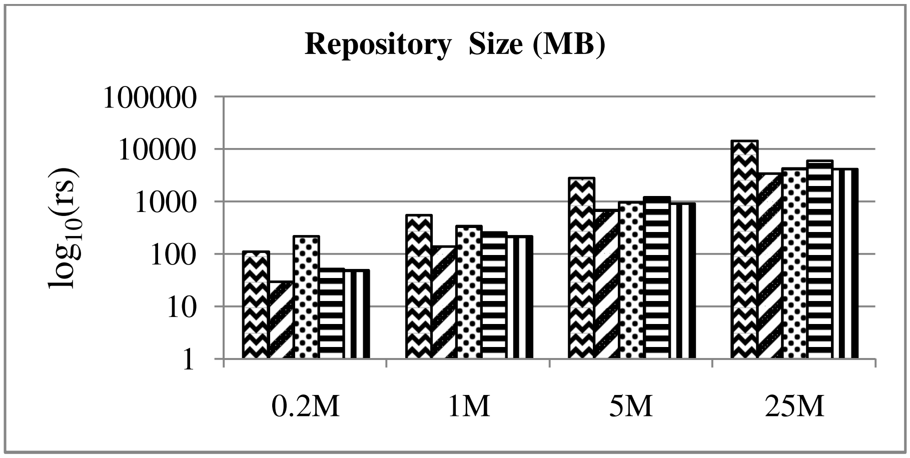
5.2.2.1. Load Results

Load test cases show interesting results regarding load time and resource utilization. Both of load test cases described are ventured. The datasets are asserted into single model by all Semantic Web databases.

Bulk Load: Figure 36 shows the comparative performance of Semantic Web databases on bulk load. Experimental results showed that $Sesame_{M}$ had better load time and resource utilization than $Jena_{M}$ during bulk load. Both these stores failed to load twenty five million dataset.

AllegroGraph showed better performance for bulk load over other native Semantic Web databases. It required less load time, memory usage and CPU time than $Sesame_{N}$ and TDB. $Sesame_{N}$ proved to be faster than TDB in terms of its load time and CPU time, but its main memory requirement for loading was greater than TDB. AllegroGraph, with least consumption of main memory and CPU utilization, occupied lots of disk space. Since physical disk is neither an expensive nor a limited resource, so its better load time, less memory consumption and CPU time make it a better native RDF data loading API. From non-memory non-native category, SDB required less load and CPU time but more memory and disk space than $Sesame_{RDB}$.


Sesame_M
Jena_M
AllegroGraph
Sesame_N
TDB
SDB
Sesame_{RDB}
Load Time (s)
$\log_{10}(t)$
1000000
100000
10000
1000
100
10
1
0.2M
1M
5M
25M
(a)
CPU Time (s)
$\log_{10}(\text{cpu time})$
10000
1000
100
10
1
0.1
0.2M
1M
5M
25M
(b)
Main Memory (MB)
$\log_{10}(\text{mm})$
10000
1000
100
10
1
0.2M
1M
5M
25M
(c)



(d)

Figure 36: Bulk Load Comparative Evaluation Results

Table 21 presents the summary of Bulk Load results. Comparatively good store of each category against its corresponding metrics is shown here.

Table 21: Summary of Bulk Load Results

Metrics	**Semantic Web Databases**		
	In-memory	**Native**	**Non-memory non-native**
Load Time	Sesame_M	AllegroGraph	SDB
CPU Time	Sesame_M	AllegroGraph	SDB
Main Memory	Sesame_M	AllegroGraph	Sesame_{RDB}
Repository Size	×	Sesame_N	Sesame_{RDB}

Incremental Load: We loaded ten triples in already created stores. Tests results showed almost constant time and resources utilization during incremental load for in memory and native stores of all datasets except for AllegroGraph. AllegroGraph is expensive in terms of its load time for Incremental Load. SesameRDB incremental load time was much greater than SDB. Load time results for Incremental Load test case are shown in Table 22.

Table 22: Load Time for Incremental Load

Semantic Web Databases	Load Time (sec)			
	Dataset1 (0.2M)	Dataset2 (1M)	Dataset3 (5M)	Dataset4 (25M)
Sesame$_M$	1	1	1	1
Jena$_M$	1	1	1	1
AllegroGraph	2	3	13	38
Sesame$_N$	1	1	1	1
TDB	2	2	2	2
SDB	1	2	3	5
Sesame$_{RDB}$	12	41	122	347

5.2.2.2. Read Results

Test results for read operations were more complex to analyze. Due to the large number of results retrieved from these tests against different performance parameters, we have only shown results for query response time in detail and described resource usage briefly.

Simple Query: Figure 37 (a)(b)(c) describe the store performance on searching for a single triple pattern for a specific predicate, object and subject value respectively. Sesame$_M$ query response was much better for all three queries than Jena$_M$.

Sesame$_N$ showed better query response time in its category for predicate search on all datasets. But while searching on Object and/or Subject TDB performs even better than sesame on large dataset. From RDB backed stores Sesame$_{RDB}$ showed better results while searching on a predicate but for other two searches SDB showed better response time over Sesame$_{RDB}$.

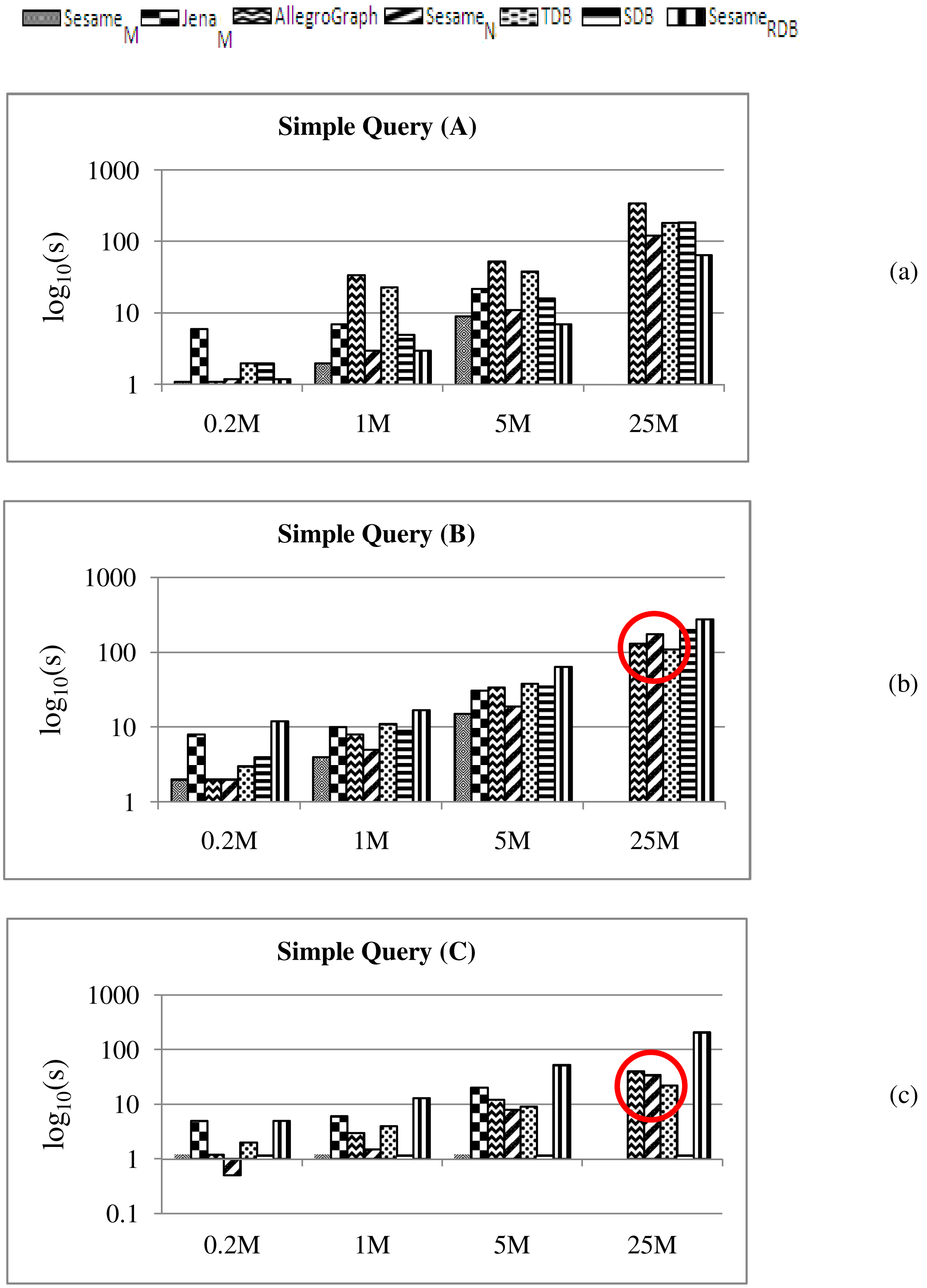


Figure 37: Simple Query Results for Query Response Time

Table 23 presents the summary of Simple Query results. Comparatively good store of each category against its corresponding test case for *Query Response Time (QRT)* are shown here.

Table 23: Summary of Simple Query Results (QRT)

Test Cases	**Semantic Web Databases**		
	In-memory	**Native**	**Non-memory non-native**
SQ(A-Predicate search)	Sesame_{M}	Sesame_{N}	$\text{Sesame}_{\text{RDB}}$
SQ(B-Object search)	Sesame_{M}	TBD (DS-4)	SDB
SQ(C-Subject search)	Sesame_{M}	TDB (DS-4)	SDB

Complex Query: Complex query results are shown in Figure 38. Query results implementing the Bushy Patterns revealed that all stores failed to retrieve the results for a large dataset size. Sesame_{M} showed better response time and scaled well for memory and CPU utilization than Jena_{M}.

Sesame_{N} and TDB showed less query response time than AllegroGraph. $\text{Sesame}_{\text{RDB}}$ executed bushy pattern with less response time than SDB. Figure 38 (b) shows the results for long chain pattern. Jena_{M} displayed better results for long chain pattern queries than Sesame_{M}. TDB has better results for Long Chain Pattern identification than Sesame_{N}. $\text{Sesame}_{\text{RDB}}$ performed better than SDB from non-memory non-native category. For Long Chain Patterns identification $\text{Sesame}_{\text{RDB}}$ performed even better than the native and memory stores.

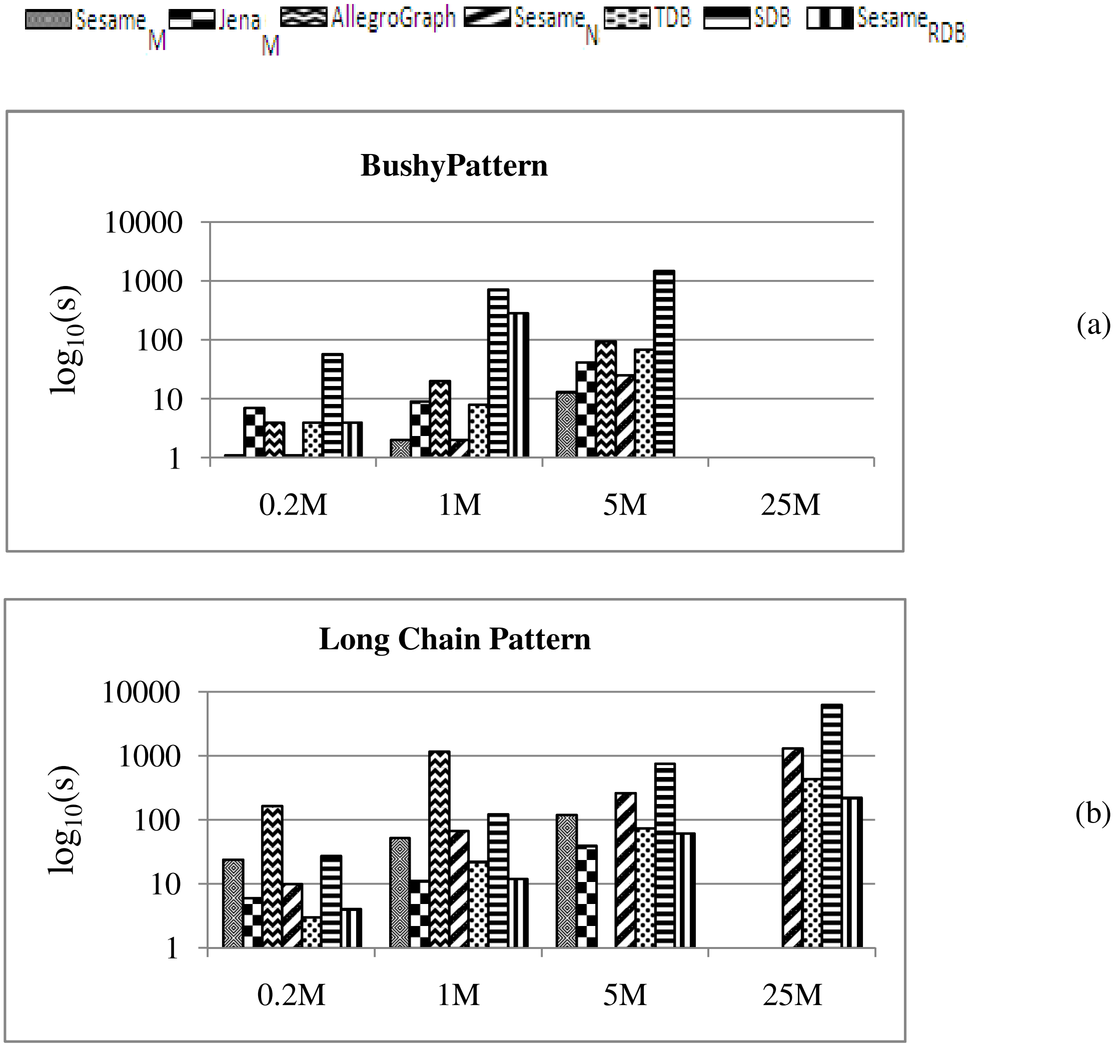


Figure 38: Complex Query Results for Query Response Time

Table 24 presents the summary of Complex Query results. Comparatively good store of each category against its corresponding test case for *Query Response Time (QRT)* are shown here.

Table 24: Summary of Complex Query Results (QRT)

Test Cases	**Semantic Web Databases**		
	In-memory	**Native**	**Non-memory non-native**
Bushy Pattern	Sesame_{M}	Sesame_{N}	$\text{Sesame}_{\text{RDB}}$
Long Chain Patterns	Jena_{M}	TBD	$\text{Sesame}_{\text{RDB}}$

Result Size: Result Size (A), Result Size (B), Result Size(C) queries returns large, medium and small result sizes respectively. For all result set sizes Sesame_{M} exhibited better results than Jena_{M}. Sesame_{N} and $\text{Sesame}_{\text{RDB}}$ performed extremely faster than other native stores for all result set sizes. However while retrieving for large results set on large datasets; AllegroGraph exhibited better results than other native stores. ResultSize results are shown in Figure 39.

Table 25 presents the summary of ResultSize results. Comparatively good store of each category against its corresponding test case for *Query Response Time (QRT)* are shown here.

Table 25: Summary of ResultSize Results (QRT)

Test Cases	**Semantic Web Databases**		
	In-memory	**Native**	**Non-memory non-native**
Result-Size(A)	Sesame_{M}	Sesame_{N}	$\text{Sesame}_{\text{RDB}}$
Result-Size(B)	Sesame_{M}	Sesame_{N}	$\text{Sesame}_{\text{RDB}}$
Result-Size(C)	Sesame_{M}	Sesame_{N}	$\text{Sesame}_{\text{RDB}}$

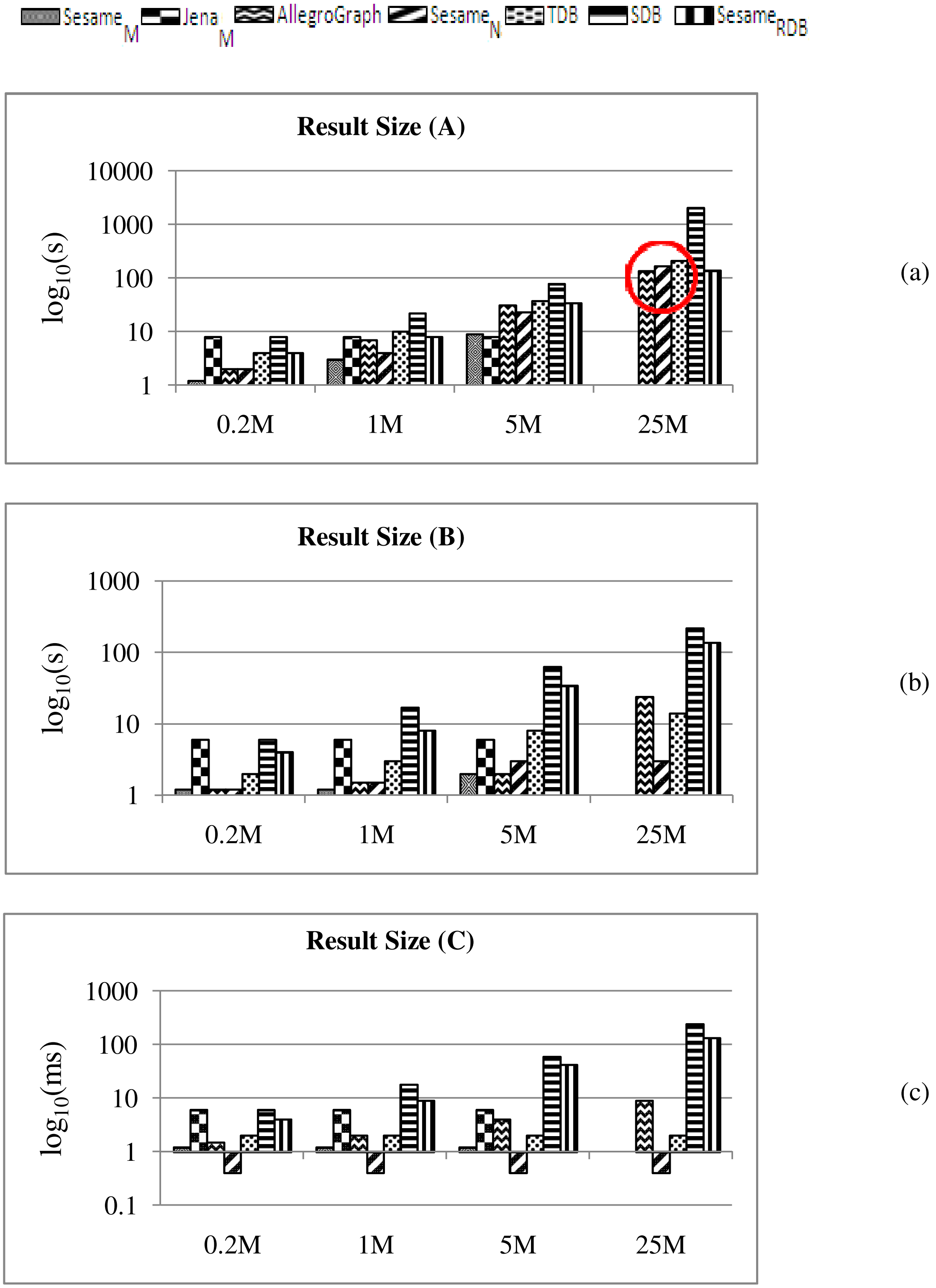


Figure 39: ResultSize results for Query Response Time

Selectivity Estimation: We run the same query of having different triple selectivity and join selectivity in all possible orders for all stores. Each time they produced the same results. This showed that they have optimized the query on the basis of their selectivity. For selectivity estimation results SesameM, Sesame N and SesameRDB performed best from their respective category as shown in Figure 40.

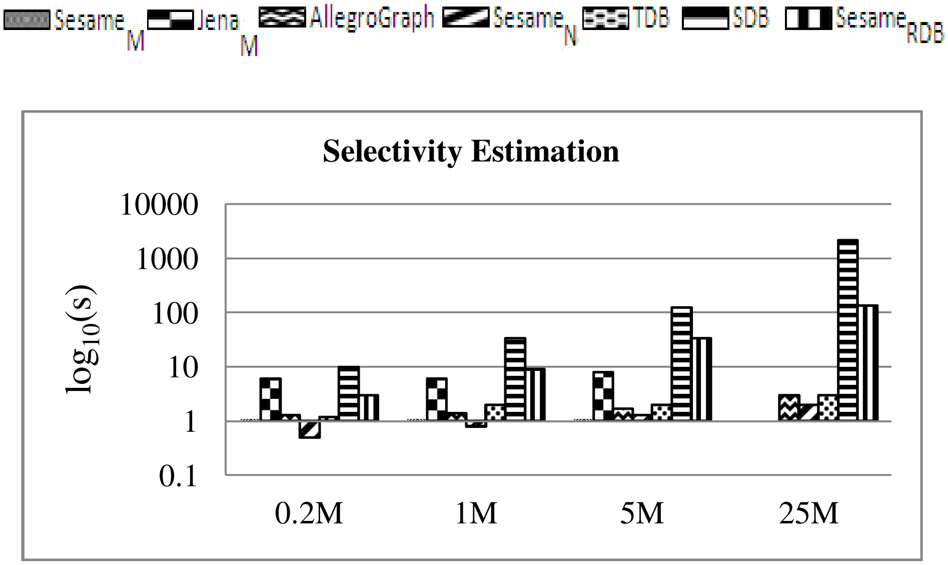


Figure 40: Selectivity Estimation Results for Query Response Time

Irregular Access Pattern: For irregular access pattern all the stores fail to retrieve results on large dataset. Sesame$_M$, TDB and Sesame$_{RDB}$ performed the best from their respective category for this test case. But a very strange behavior of Sesame$_N$ was observed. Despite of its better query execution cost Sesame$_N$ fails drastically while retrieving triples for irregular pattern as shown in Figure 41.

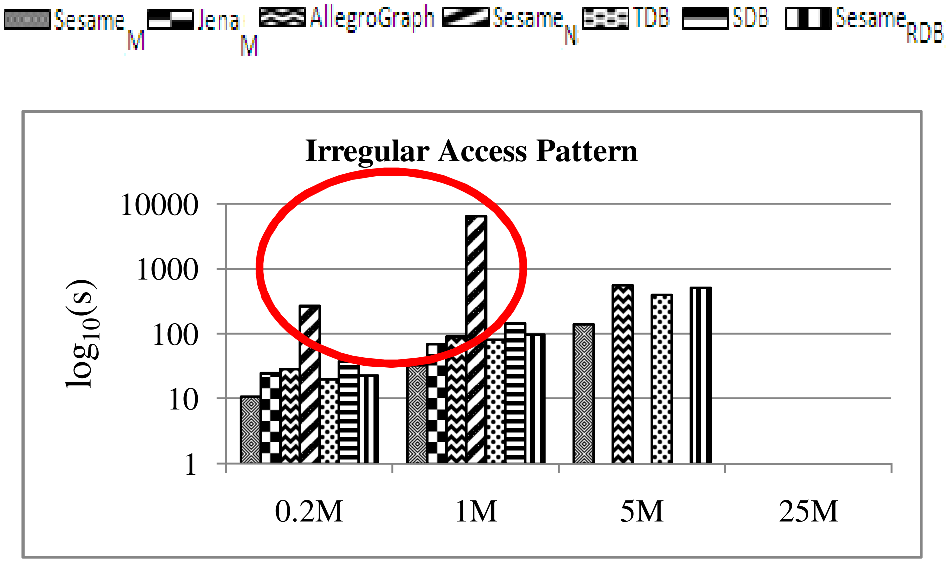
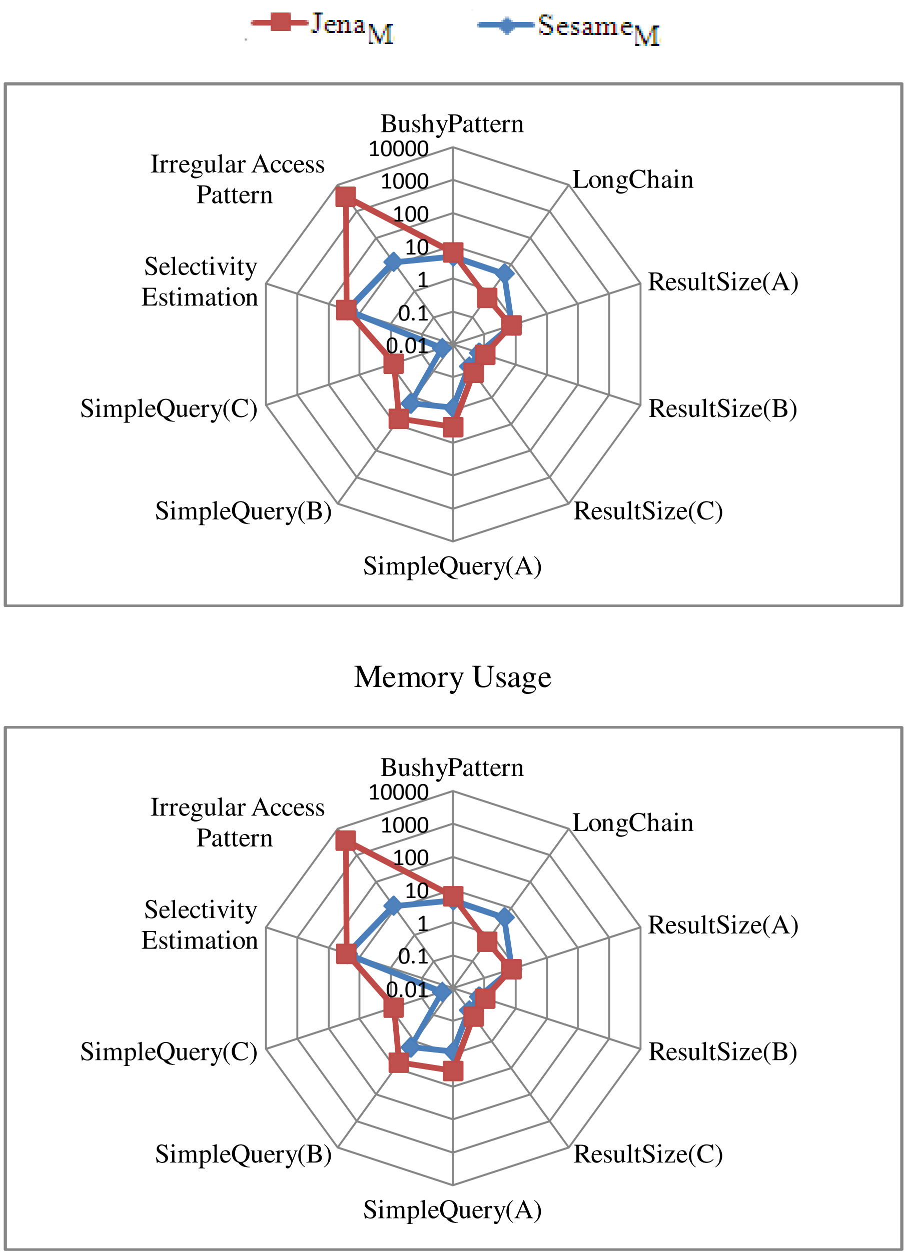


Figure 41: Irregular Access Pattern Results for Query Response Time

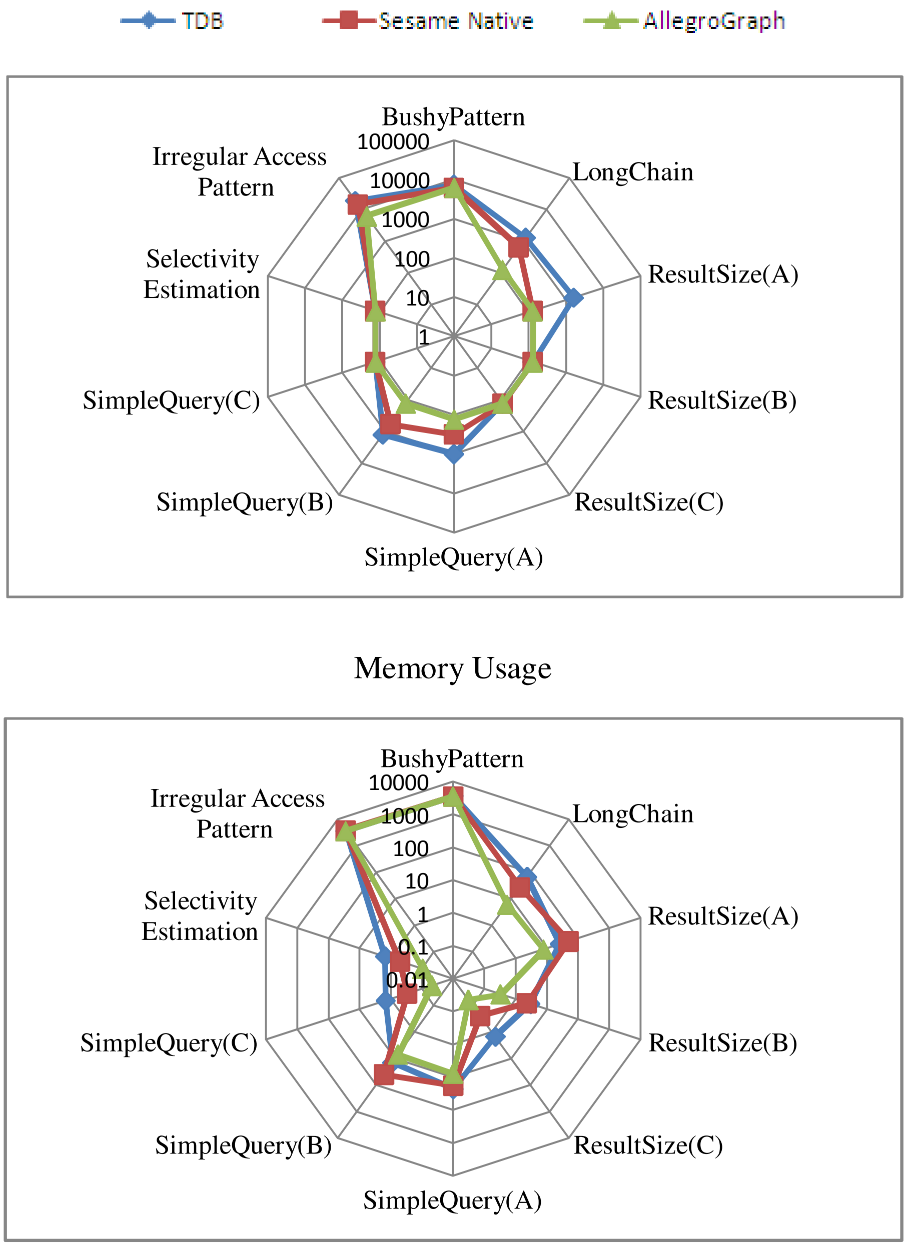
Resource utilization of *In-memory*, *Native* and Non-memory Non-native stores for all Read test cases is shown in Figure 42, Figure 43, and Figure 44. From In-memory category results we concluded that Sesame$_M$ is better resource utilizer than Jena$_M$ as shown in Figure 42 (a) (b). Figure 43 (a) models the main memory usage whereas Figure 43 (b) models the CPU time for each store. It is clearly observed that from native category AllegroGraph utilizes resources more efficiently than Sesame$_N$ and TDB. From remaining two Sesame$_N$ is better than TDB for its resource utilization. From Non-memory non-native category Sesame$_{RDB}$ exhibited better resource utilization than SDB as shown in Figure 44 (a) (b).



Memory Usage

CPU Time

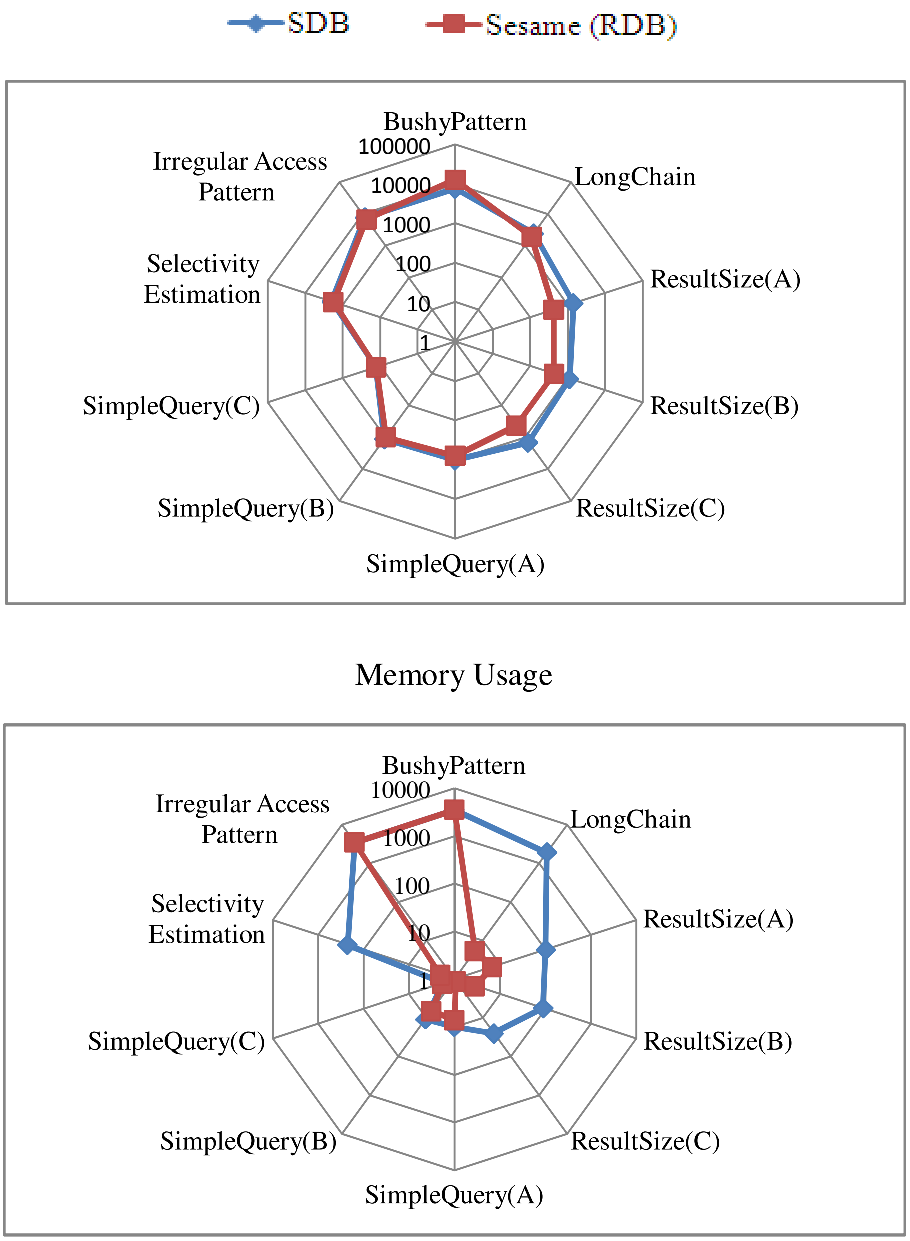
Figure 42: Resource Utilization of In-Memory Stores for Read Test Cases



Memory Usage

CPU Time

Figure 43: Resource Utilization of Native Stores for Read Test Cases



Memory Usage

CPU Time

Figure 44: Resource Utilization of Native Stores for Read Test Cases

5.2.2.3. Delete Results

While all datasets were loaded into single store and reasoning was not considered in our tests, deleting an entire store is relatively a simple matter. For other delete test case i.e. “Deleting some statements from the store”, we deleted ten statements from data stores. All in memory and native stores showed constant time and resource usage for different datasets. TDB took almost double time than other native stores. SDB took much more time than all other stores as shown in Table 26.

Table 26: Delete Results

Semantic Web Databases	Delete Time (sec)			
	Dataset1 (0.2M)	Dataset2 (1M)	Dataset3 (5M)	Dataset4 (25M)
Sesame$_M$	1	1	1	1
Jena$_M$	1	1	1	1
AllegroGraph	1	1	1	1
Sesame$_N$	1	1	1	1
TDB	2	2	2	2
SDB	15	117	239	356
Sesame$_{RDB}$	1	12	40	125

5.2.2.4. Update Results

All these Semantic Web databases do not provide facility for update operations. Updating a triple in a Semantic Web database is done by deleting the original triple and loading the new triple. Therefore, cost of updating a triple is equal to cost of deleting a triple from the store and then loading a new triple into it.

5.2.3. Cumulative Query Performance Results

Cumulative performance results are shown in Table 27, Table 28, and Table 29. These results clearly present an overall query cost estimation for each Semantic Web database. Overall results revealed that Sesame$_M$ exhibited less response time and less resource utilization than Jena$_{M,}$ therefore query cost is less for Sesame$_M$ than Jena$_M$. Since Sesame$_M$ also exhibited better load performance both in terms of time and resource utilization, therefore Sesame$_M$ clearly dominate the Jena$_M$.

Non-memory non-native stores exhibited greater query cost than other two types of stores. Sesame$_{RDB}$ demonstrated better read performance both for query response time and resource utilization. However SDB exhibited better load performance both in terms of bulk load and incremental load. Therefore SDB is *"write optimized"* and Sesame$_{RDB}$ is *"read optimized*".

Sesame$_N$ and TDB both showed better response time than AllegroGraph, but resource utilization of Sesame$_N$ and TDB is much higher than AllegroGraph. AllegroGraph also presented better load performance than the other two in terms of load time, memory usage and CPU time. On the basis of these analyses we believe that Sesame$_N$ and TDB can exhibit better query performance for a system having high computing resources, but with low resources, AllegroGraph would be better option. Results showed that AllegroGraph is *"write optimized"* and Sesame$_N$ and TDB are *"read optimized*".

Table 27: Geometric Mean of Query Response Time for Read Query Set

Datasets	***Read Execution Time (sec)***						
	Sesame$_{M}$	**Jena$_{M}$**	**Allegro Graph**	**Sesame$_{N}$**	**TDB**	**SDB**	**Sesame$_{RDB}$**
Dataset1	2.24	7.31	3.21	2.01	2.93	8.18	5.28
Dataset2	3.31	9.34	11.76	5.73	8.4	26.25	18.34
Dataset3	7.81	36.71	45.98	22.55	22.35	137.09	128.55
Dataset4	A	α	398.48	139.75	173.43	928.283	825.62

Table 28: Geometric Mean of Main Memory Usage for Read Query Set

Datasets	***Read Main Memory Usage (MB)***						
	Sesame$_{M}$	**Jena$_{M}$**	**Allegro Graph**	**Sesame$_{N}$**	**TDB**	**SDB**	**Sesame$_{RDB}$**
Dataset1	646.75	643.23	128.45	171.87	140.87	216.54	190.66
Dataset2	1930.28	2046.98	128.4	172.57	194.36	398.27	284.7
Dataset3	3158.94	3572.66	189.1	252.93	367.48	1042.26	559.15
Dataset4	A	α	409.98	415.05	693.6	1506.09	1005.77

Table 29: Geometric Mean of CPU time for Read Query Set

	Read CPU Time (sec)						
	Sesame$_{M}$	**Jena$_{M}$**	**Allegro Graph**	**Sesame$_{N}$**	**TDB**	**SDB**	**Sesame$_{RDB}$**
Dataset1	2.19	4.86	0.2	0.6	2.26	3.34	1.48
Dataset2	3.09	8.32	0.98	1.43	7.5	5.67	2.36
Dataset3	5.26	28.6	2.6	5.66	19.4	39.4	15.94
Dataset4	A	α	4.55	25.7	89.1	187.8	56.33

Conclusion of Comparative Analysis:

Comparative evaluation results are summarized in Table 30. In this table we concluded the read and write optimized stores of each category (i.e. in-memory, native, non-memory & non-native) with respect to time and resource utilization. From the in-memory category Sesame$_M$ is read & write optimized both in terms of time and resource usage. It also have higher success ratio than Jena$_M$. Sesame$_N$ and TDB are read optimized in terms of time and AllegroGraph is read optimized in terms of resource usage from the native category. However AllegroGraph is also write optimized in terms of both time and resource usage. Success ratio of TDB is better than the other two. From non-memory non-native category Sesame$_{RDB}$ is read optimized and SDB is write optimized, both stores exhibited equal success ratio.

Table 30 : Summary of Comparative Evaluation

	Summary Table for Comparative Evaluation						
	Sesame$_M$	**Jena$_M$**	**Allegro Graph**	**Sesame$_N$**	**TDB**	**SDB**	**Sesame$_{RDB}$**
Read optimized	✓$_{TR}$	×	✓$_R$	✓$_T$	✓$_T$	×	✓$_{TR}$
Write Optimized	✓$_{TR}$	×	✓$_{TR}$	×	×	✓$_{TR}$	×
Successful	✓	×	×	×	✓	=	=

In Table 30

✓$_{TR}$ = Optimized in terms of Time and Resources

✓$_T$ = optimized in terms of Time

✓$_R$ = Optimized in terms of Resources

Overall native stores performed better in terms of their success ratio, time and resource usage.

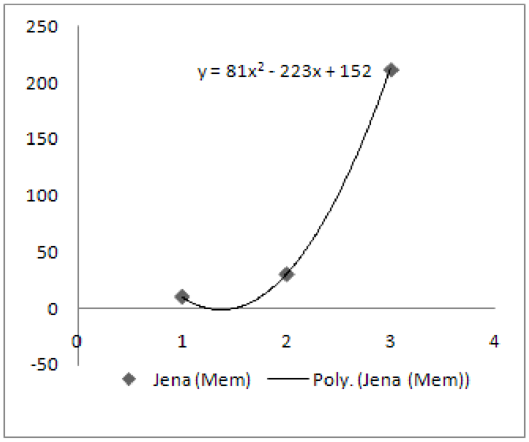
5.3. Scalability Analysis Results

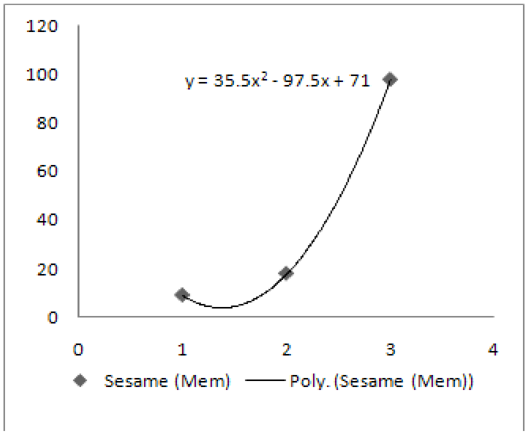
This section provides detailed scalability analysis of tested data stores on the basis of results obtained from our tests for comparative evaluation.

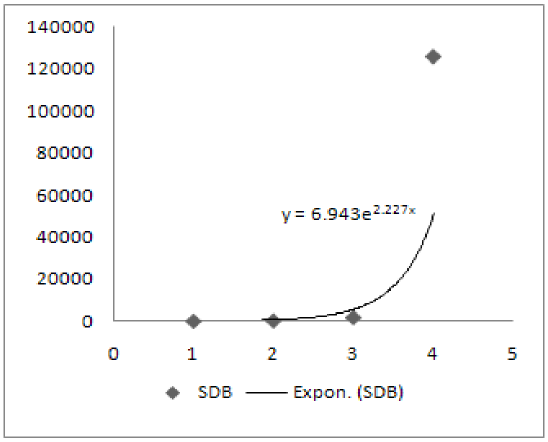
5.3.1. Load Analysis

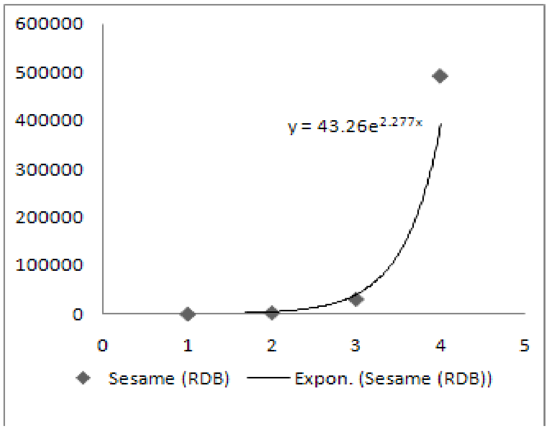
5.3.1.1.Bulk Load

Load Time: Load time analysis revealed that Load time for in-memory stores increases in polynomial time, but for all native and non-memory non-native stores, it increases exponentially as shown in Figure 45. Here x-axis shows the time in seconds and y-axis is representing our linearly increasing datasets from 1 to 4.









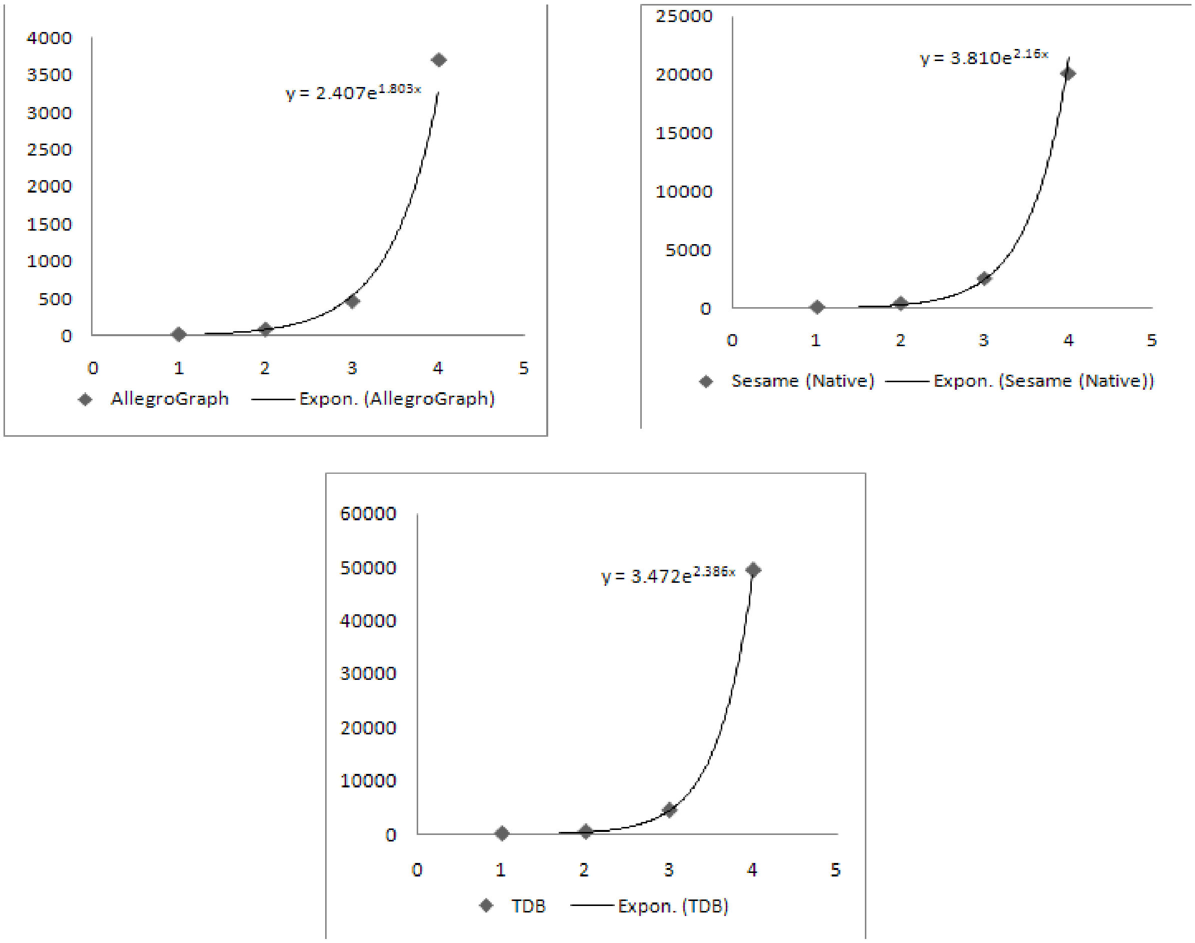
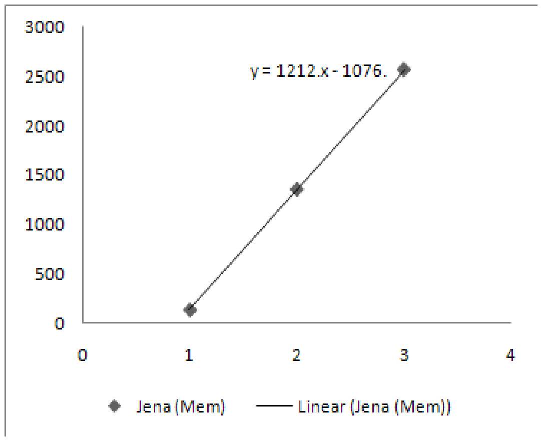
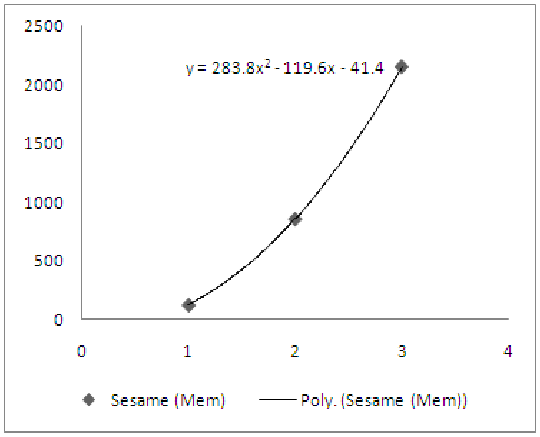
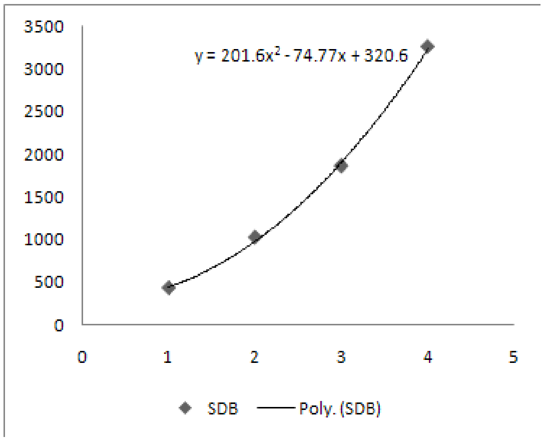


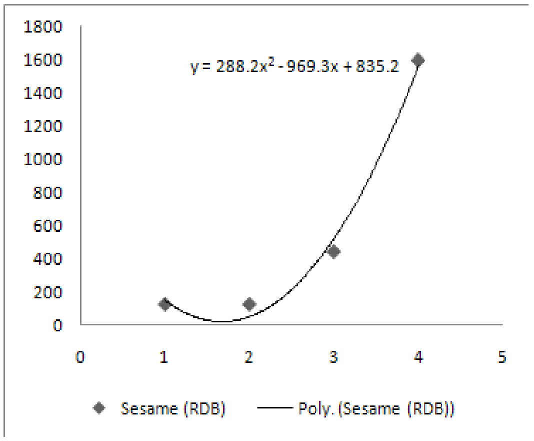
Figure 45: Load Time Scalability Behavior

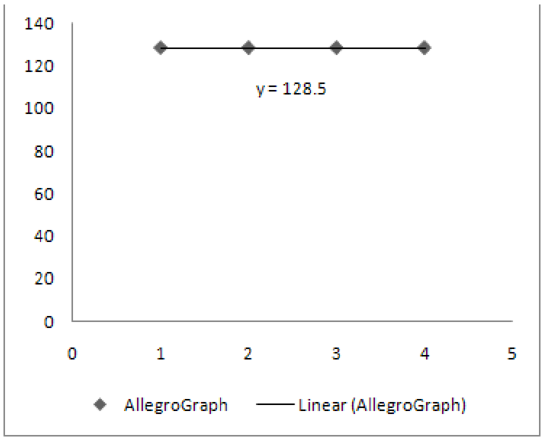
Main Memory: Usage of main memory during bulk load increased almost linearly for in-memory & native stores expect for AllegroGraph for which it remained constant. However for non-memory non-native stores, it increases in polynomial times as shown in Figure 46. Here x-axis and y-axis are representing main memory in MBs and our linearly increasing datasets from 1 to 4 respectively.

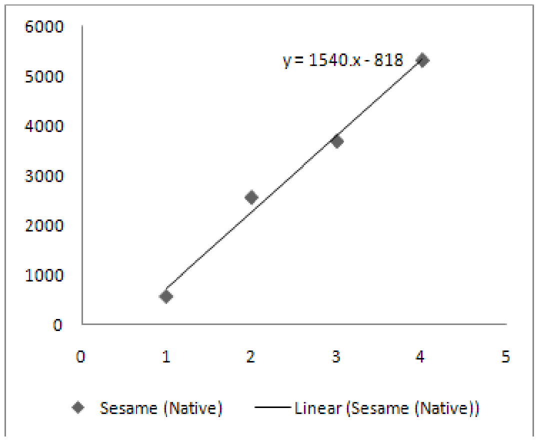

3000
2500
2000
1500
1000
500
0
0
1
2
3
4
$y = 1212.x - 1076.$
◆ Jena (Mem)
Linear (Jena (Mem))


2500
2000
1500
1000
500
0
0
1
2
3
4
$y = 283.8x^2 - 119.6x - 41.4$
◆ Sesame (Mem)
Poly. (Sesame (Mem))


3500
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
$y = 201.6x^2 - 74.77x + 320.6$
◆ SDB
Poly. (SDB)


1800
1600
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
$y = 288.2x^2 - 969.3x + 835.2$
◆ Sesame (RDB)
Poly. (Sesame (RDB))


140
120
100
80
60
40
20
0
0
1
2
3
4
5
$y = 128.5$
◆ AllegroGraph
Linear (AllegroGraph)


6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
$y = 1540.x - 818$
◆ Sesame (Native)
Linear (Sesame (Native))

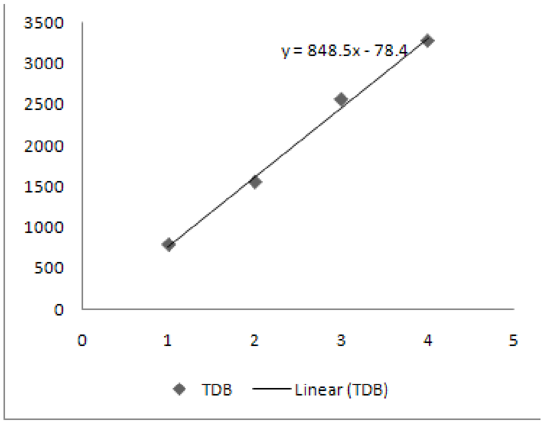
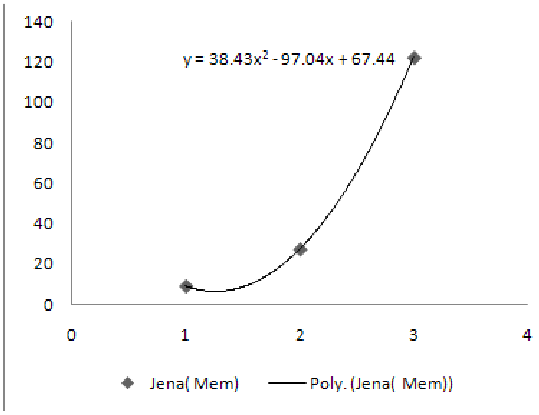
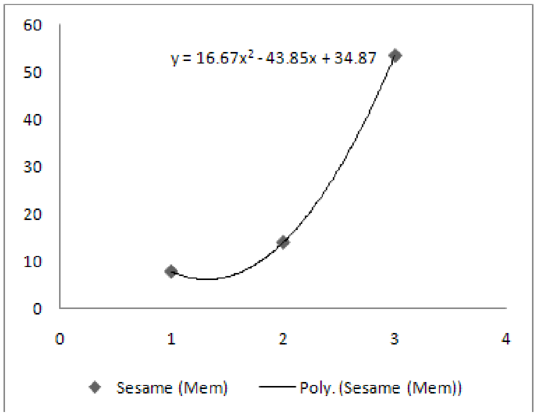


Figure 46: Main Memory Usage Scalability Trends during Bulk Load

CPU Time: CPU time for main memory stores increases in polynomial time, however for native and non-memory non-native stores it increases exponentially during bulk load tests as shown in Figure 47. Here x-axis is representing the CPU time in seconds and y-axis is representing our linearly increasing datasets from 1 to 4.





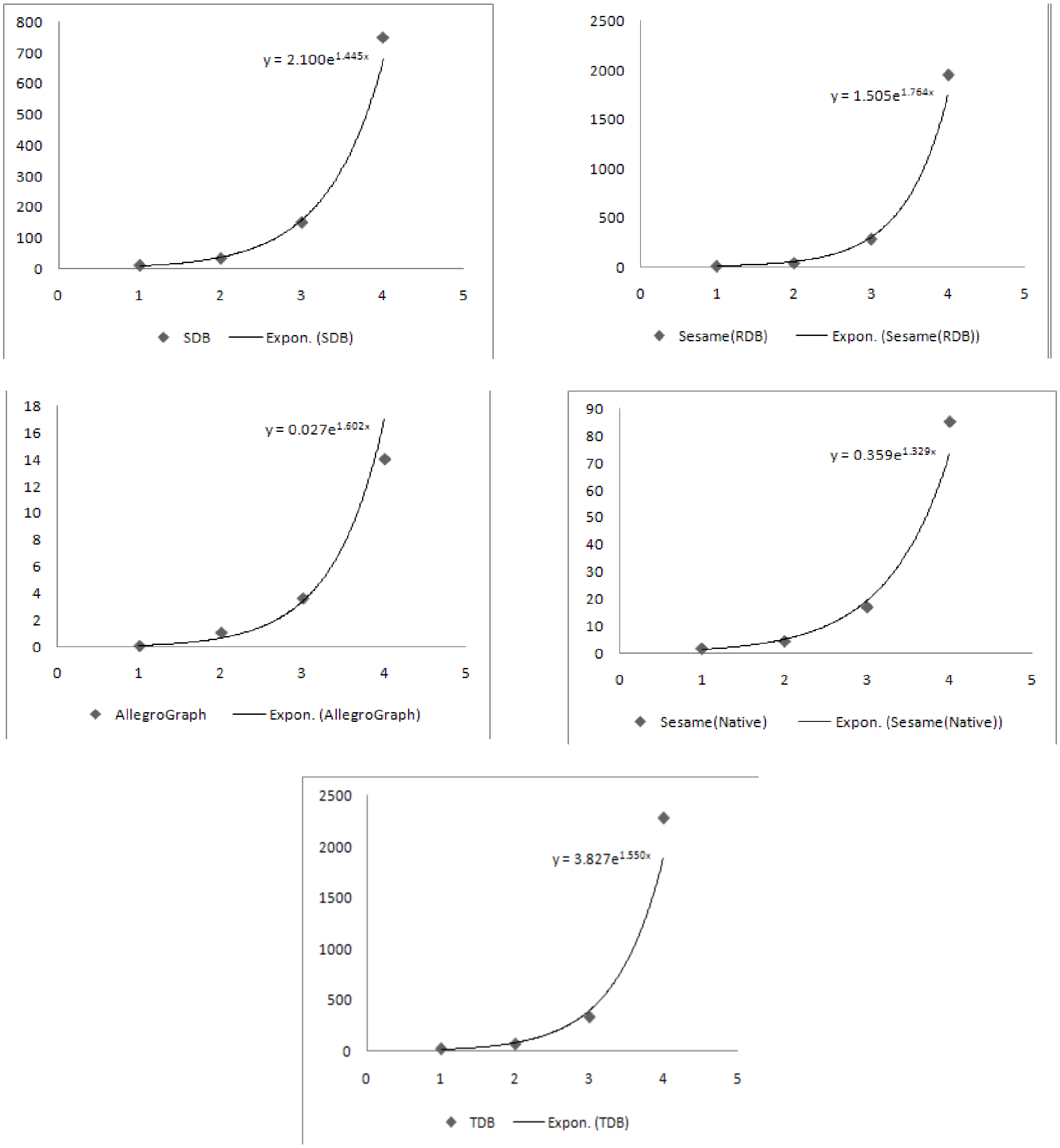


Figure 47: CPU Time Scalability Trends during Bulk Load

Disk Space: During Bulk Load disk space is not used by in-memory stores as they keep and manage data in main memory and do not stores data permanently. However, for all other stores disk space usage increases exponentially for linearly increasing dataset as shown in Figure 48. In this diagram x-axis and y-axis are representing physical disk size in MBs and our linearly increasing datasets from 1 to 4 respectively.

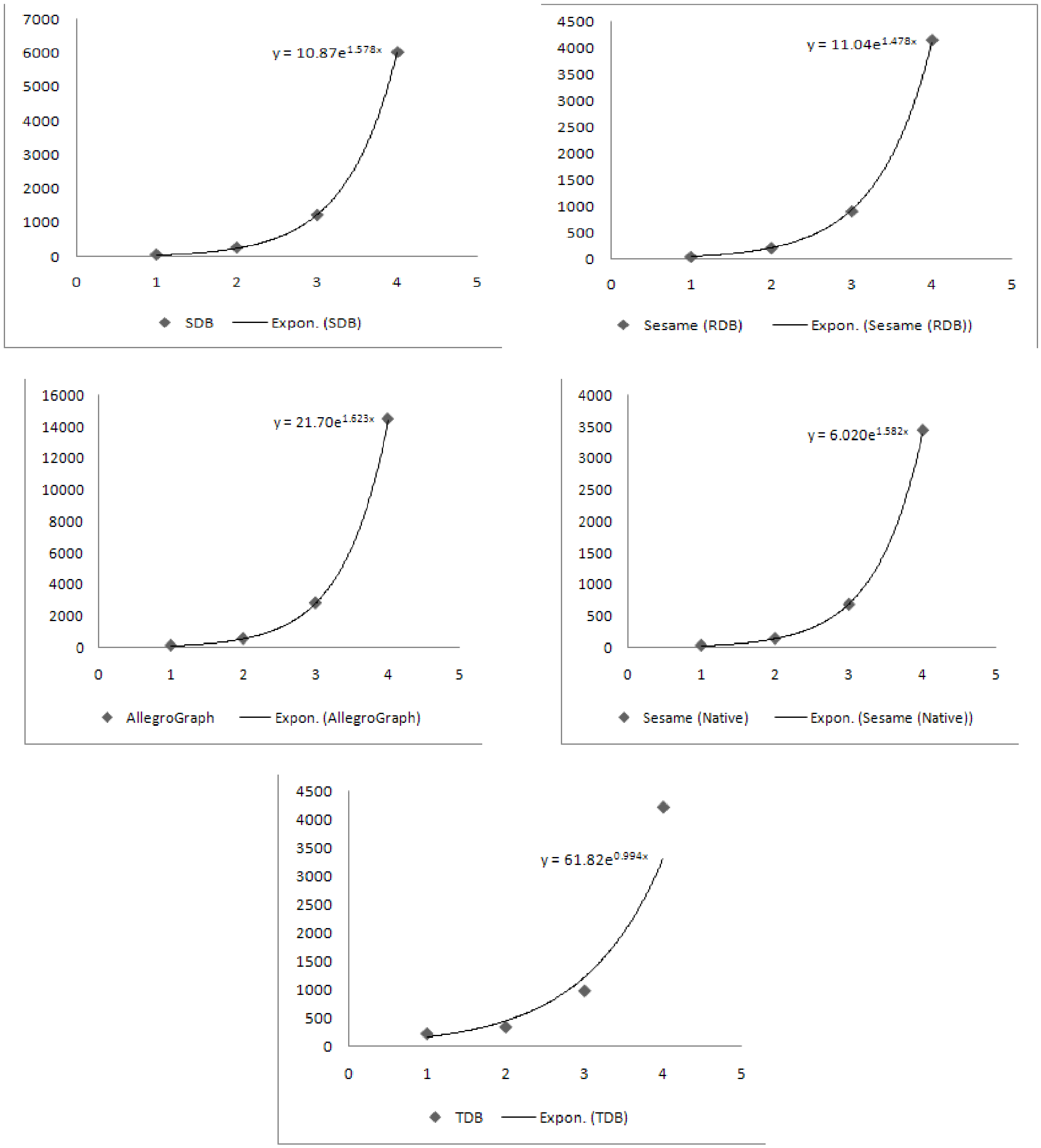


Figure 48: Disk Space Scalability Trends during Bulk Load

5.3.1.2. Incremental Load

Test results for incremental load revealed that both of in-memory stores have a constant time and resource utilization for all datasets. Sesame$_{N}$ and TDB of native stores category also have a constant time and resource utilization for all datasets. However, AllegroGraph exhibited linear increase in load time and resource usage for linearly increasing datasets for this test case. From non-memory non-native category of Semantic Web databases SDB and Sesame$_{RDB}$ requirements, for time and resource usage, increases in polynomial times and exponentially respectively as shown in Figure 49. In this diagram x-axis and y-axis are representing load time in seconds and our linearly increasing datasets from 1 to 4 respectively.

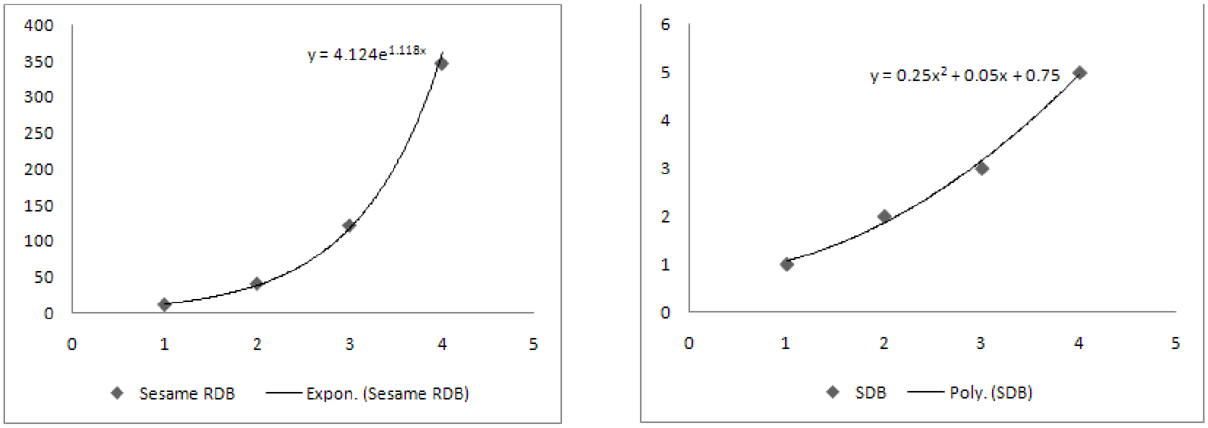
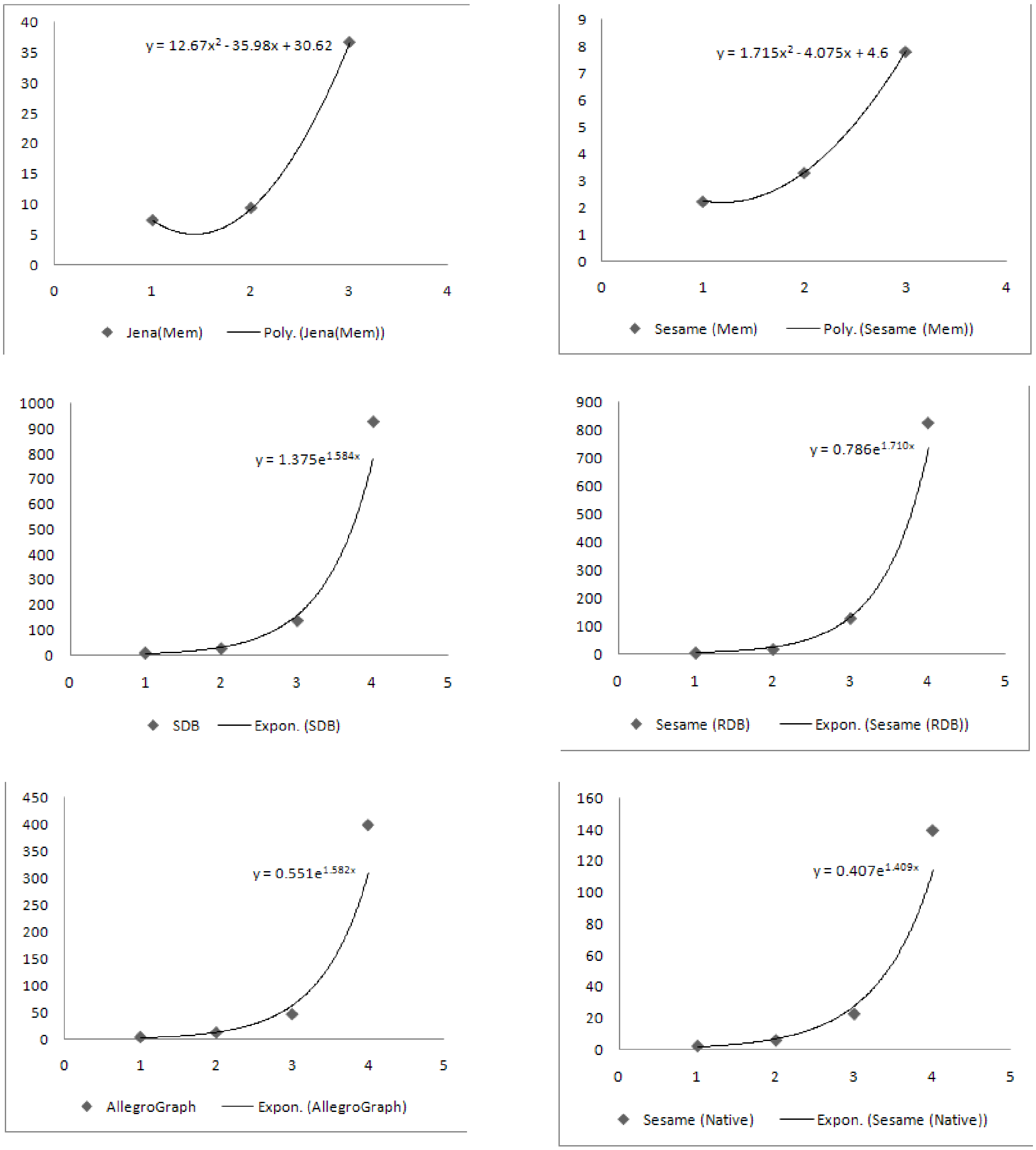


Figure 49: Incremental Load Test results for SDB and Sesame$_{RDB}$

5.3.2. Read Analysis

Scalability analysis for read operations is presented for cumulative query performance results, rather than per query results in this section.

Query Response Time: Read tests results revealed that query response time for in-memory stores increases in polynomial time, but for all native and non-memory non-native stores, it increases exponentially as shown in Figure 50. Here x-axis shows the time in seconds and y-axis is representing our linearly increasing datasets from 1 to 4.



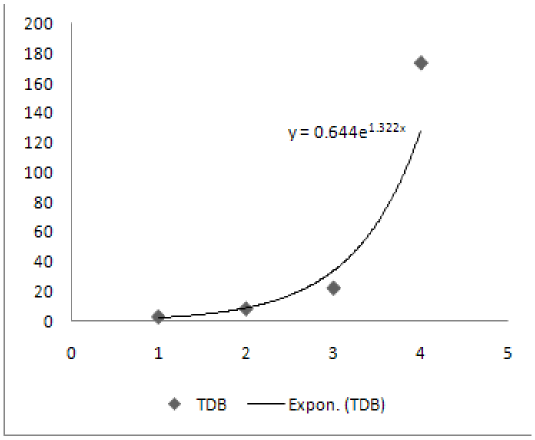
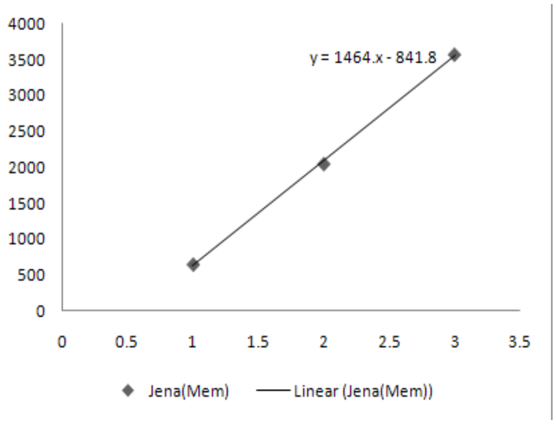
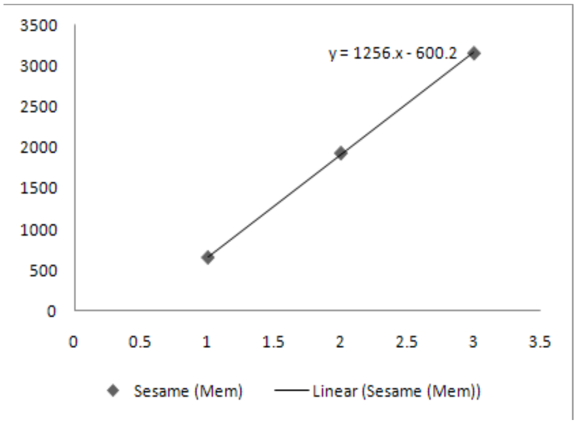


Figure 50: Query Response Time Scalability Trends for Read Test Case

Main Memory: Usage of main memory during bulk load increased almost linearly for in-memory stores. However for all native and non-memory non-native stores, it increases in polynomial times as shown in Figure 51. Here x-axis and y-axis are representing main memory in MBs and our linearly increasing datasets from 1 to 4 respectively.





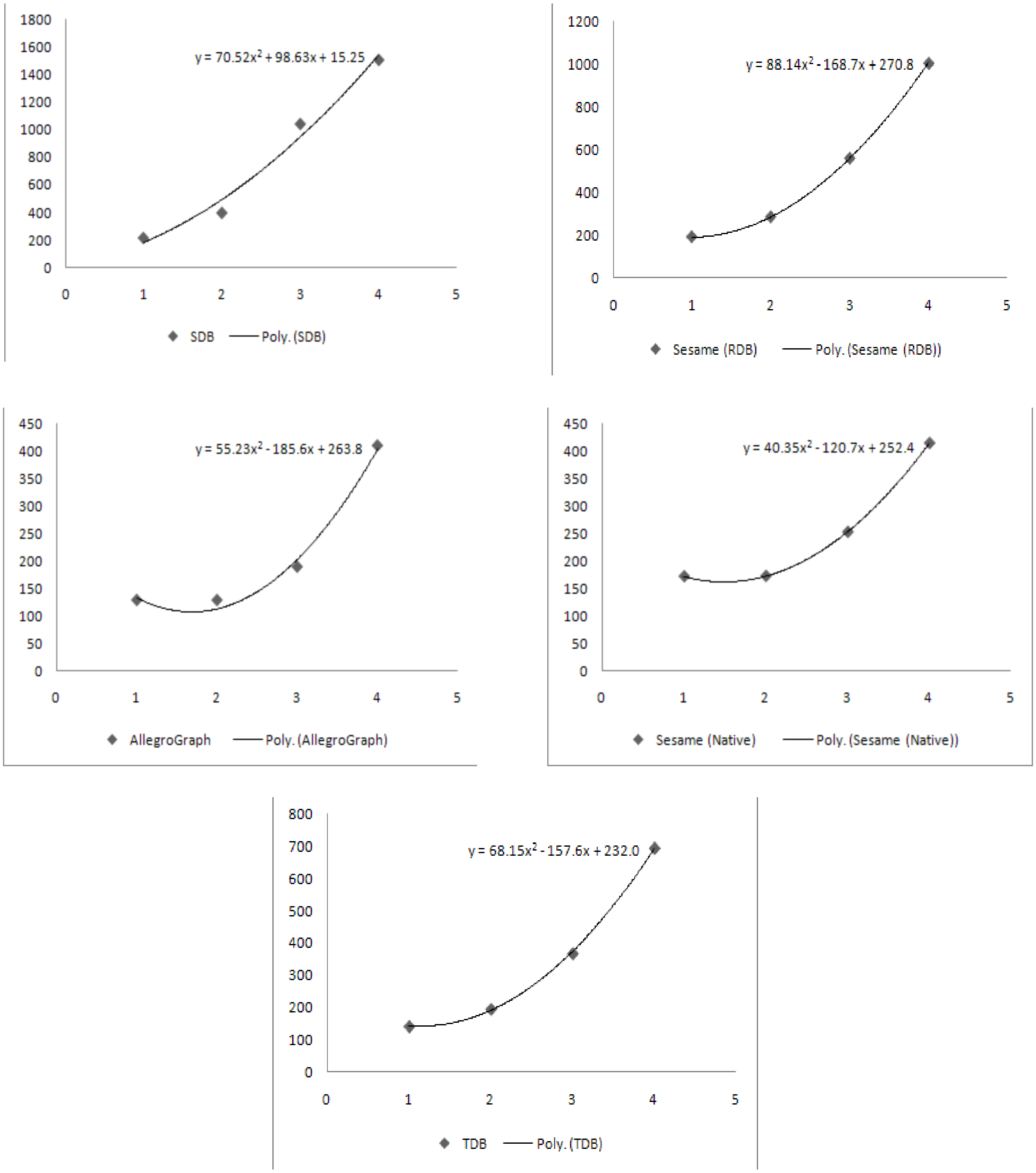
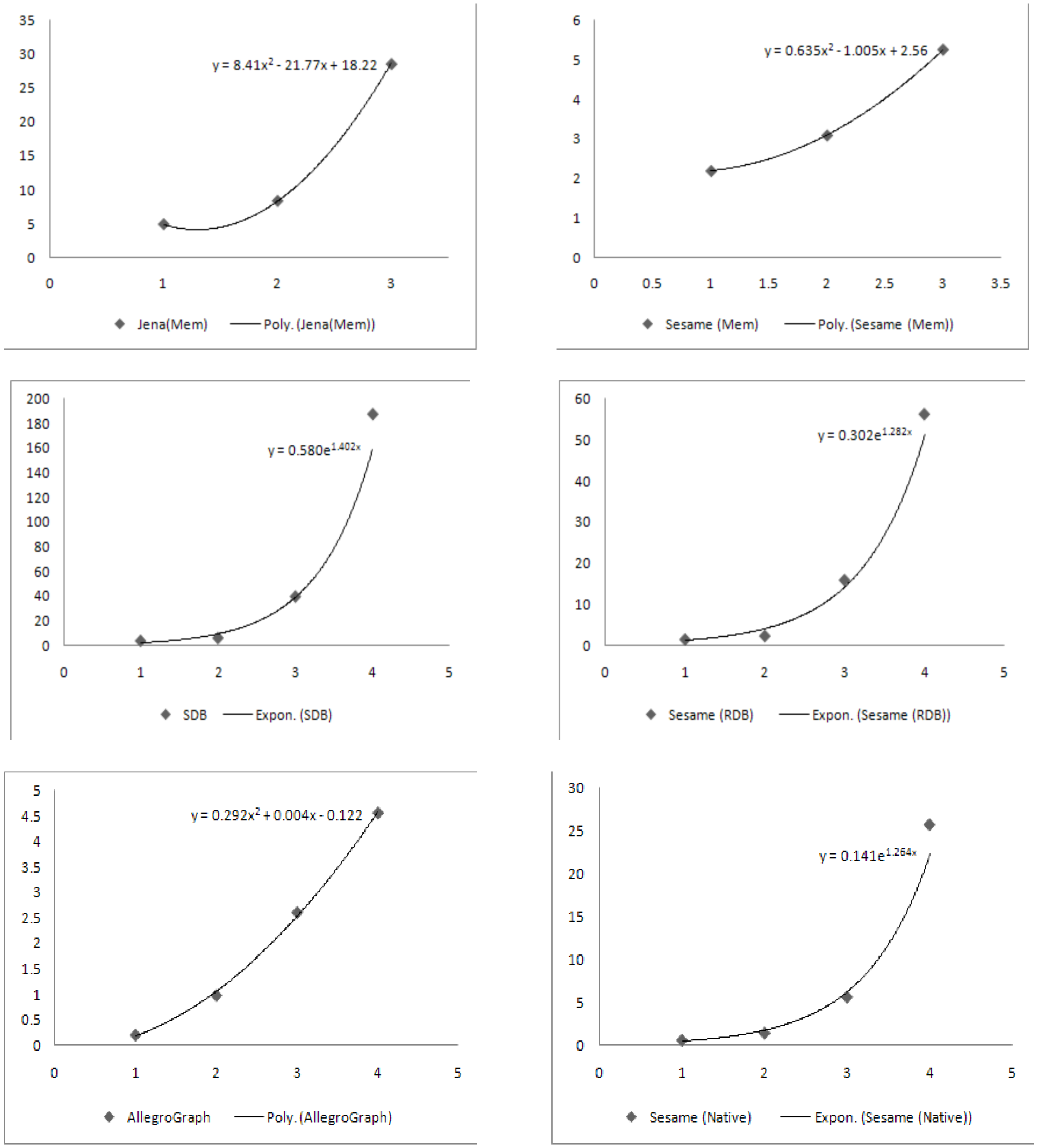


Figure 51: Main Memory Usage Scalability Trends during Read Tests

CPU Time: CPU time for main memory stores increases in polynomial time, however for native and non-memory non-native stores it increases exponentially during read tests as shown in

Figure 52. Here x-axis is representing the CPU time in seconds and y-axis is representing our linearly increasing datasets from 1 to 4.



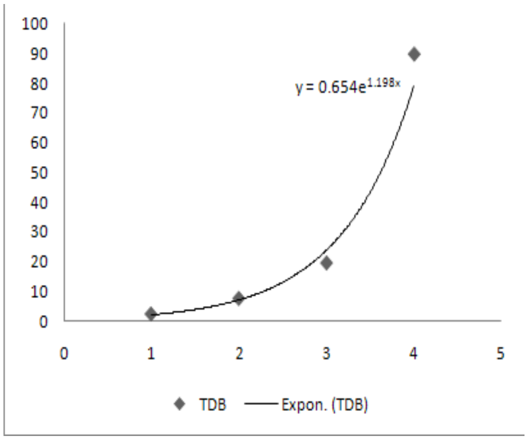


Figure 52: CPU Time Scalability Trends during Bulk Load

5.3.3. Delete Analysis

Test results for delete test case revealed that both of in-memory stores and all native stores needed a constant time and resource utilization for linearly increasing datasets for this test case. From non-memory non-native category of Semantic Web databases SDB and Sesame$_{RDB}$ requirements, for time and resource usage, increases linearly and in polynomial times respectively as shown in Figure 53. In this diagram x-axis and y-axis are representing deletion time in seconds and our linearly increasing datasets from 1 to 4 respectively.

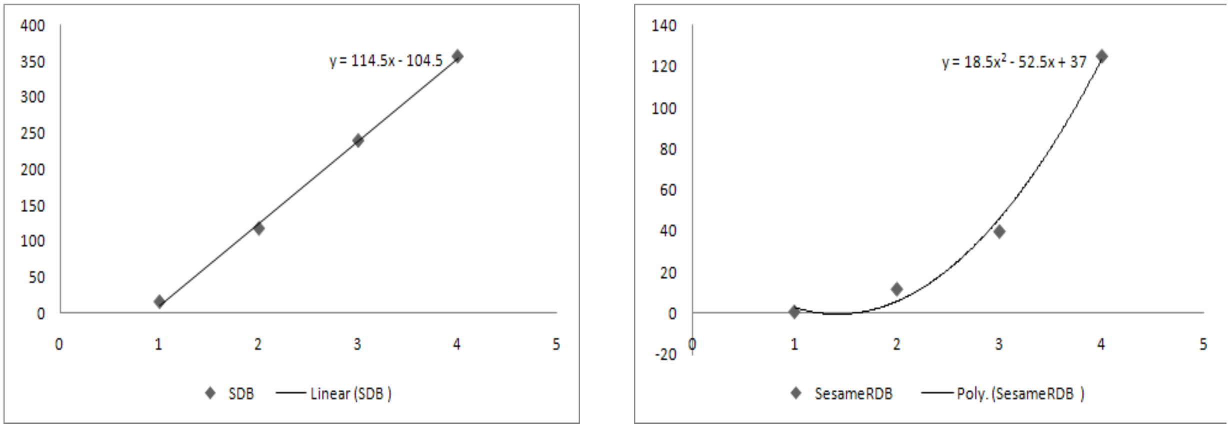


Figure 53: Delete Test results for SDB and Sesame$_{RDB}$

Conclusion of Scalability Analysis:

Table 31 presents the summary of the scalability analysis. Results depicts that although in-memory stores because of their characteristics to reside whole data in main memory are not able to handle large datasets but their time and resource usage does not increase as rapidly as in other two stores.

Table 31: Summary of Scalability Analysis

Test Cases	Metrics	Scalability Behavior of Stores		
		In-Memory Store	Native Stores	Non-memory non-native stores
Bulk Load	Load time	Polynomial	Exponential	Exponential
	Memory usage	Linear	Linear/constant	Polynomial
	CPU time	Polynomial	Exponential	Exponential
	Disk space	×	Exponential	Exponential
Read	Query execution time	Polynomial	Exponential	Exponential
	Memory usage	Linear	Polynomial	Polynomial
	CPU time	Polynomial	Exponential	Exponential

Summary of Chapter:

In this chapter we presented the detailed results and analysis obtained during our research. We presented the test configurations, both for machine and Semantic Web databases. A detailed comparative evaluation of tested Semantic Web databases is presented and strengths or weaknesses of each store are discussed on the basis of these results. By the end of this chapter, scalability analysis for these stores is provided.

CONCLUSION AND FUTURE WORK

This chapter concludes the research work carried in this thesis. It provides an analysis of the work done in this thesis. A bird's eye-view of the future directions where this work can be extended is given at the end of this chapter.

6.1. Conclusion

The Semantic Web offers the potential to vastly improve the manner in which we retrieve and interact with data. Semantic Web databases represent a critical requirement for the emergence of this vision (i.e. Semantic Web): the importance of high performance storage and query over unpredictable Semantic Web data is clear, and has been articulated during the course of this thesis.

The main goal of this research was to develop a better understanding of Semantic Web databases, and to analyze their performance behavior and scalability. We initially propose an evaluation methodology with an aim to obtain insights into key strengths and short comings of existing popular Semantic Web databases. Our evaluation methodology provides a new categorization of Semantic Web databases' operations, and proposes novel performance and scalability metrics especially the 'Resource Utilization', 'Success Ration' and 'Cumulative Query Performance'.

Moreover we evaluate and compare seven prominent Semantic Web databases on Barton Library dataset. These belong to three different categories of the Semantic Web databases and have been used frequently for performance evaluation in Semantic Web databases research literature. Results of comparative evaluation offers several contributions; results exhibit that a store performed efficient in time using some resources, but in the meantime unavailability of these resources could result in system degradation.

Strengths and weaknesses of stores under consideration have also been discovered as part of research by application of proposed framework, i.e. (a) searching for predicate is more efficient in Sesame$_N$, and for object and subject TDB showed better results than other persistent stores (b) for complex queries Sesame$_{RDB}$ exhibited better results than all other tested databases (c) result size effects the query performance of each store, AllegroGraph clearly showed better performance than other databases while searching for a large result size on large datasets and (d) Sesame$_N$ despite of its good performance exhibit degraded performance on irregular queries and long chain patterns identification, and TDB can nicely operate to retrieve long chain triple patterns.

It is also concluded that TDB has highest success ratio than all other Semantic Web database under consideration. The experimental results explained the best suited scenarios for a Semantic Web database, as Sesame$_M$ has a performance edge over Jena$_M$; AllegroGraph is write optimized and less resource utilizer of its respective category. Sesame$_N$ and TDB are read optimized in terms of their query response time. From non-memory non-native category, Sesame$_{RDB}$ is read optimized for its time and resource usage, while SDB is write optimized in terms of its load and CPU time. Over all native category of Semantic Web databases performed better than other two.

Other than these findings, we presented detailed scalability analysis and described the scalability trends on all test cases for each Semantic Web database. We concluded that in-memory stores' requirement for time and resource usage does not increase as rapidly as in other two types of

Semantic Web databases. In addition, different storage and retrieval techniques and data structures used by existing Semantic Web databases are also discussed.

6.2. Future Work

The future work provides an avenue for significant new research that will benefit the Semantic Web community. As Semantic Web databases are getting better, therefore evaluating them becomes more important. We observed that there is recent work on query performance evaluation and lot of literature has been published in recent decade. However, there is no recent work on reasoning performance evaluation. Only inventive work on reasoning performance evaluation was presented by LUBM in 2002. Similarly the third category of Semantic Web database also lack in detailed and recent benchmarks. Therefore, future work on Semantic Web database evaluation that would complement the field includes (1) Federated query and (2) Reasoning performance evaluation. Secondly there is no organized benchmark campaign for Semantic Web databases. It is evidently important to build an organized benchmark campaign for fair and realistic quantifications of Semantic Web databases' performance.

BIBLIOGRAPHY

[1] T. B. Lee, J. Hendler and O. Lassila, ”Scientific American: The Semantic Web,” in Scientific American, May 17, 2001 [online]: http://www.sciam.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21 (1 of 18)

[2] O. Lassila and R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification,” W3C Recommendation, World Wide Web Consortium, Cambridge (MA), February 1st, 1999

[3] G. Klyne and J. J. Carroll. “Resource Description Framework (RDF): Concepts and Abstract Syntax”, [online]: http://www.w3.org/TR/rdf-concepts/, 2003.

[4] K. S. Candan, H. Liu and R. Suvarna, “Resource Description Framework: Metadata and Its Applications,” SIGKDD Explorations, Vol: 3, Issue 1, Page 6, 2001.

[5] D. Reynolds, C. Thompson, J. Mukerji and D. Coleman, “An assessment of RDF/OWL modeling,” in HPL Bristol, HPL-2005-189, October 28, 2005

[6] G. Antoniou and F. V. Harmelen, "Web Ontology Language: OWL," in Book Chapter: Handbook of Ontologies, Sprinkerlink, Pages: 91-110, Sunday, March 14, 2010

[7] McGuinness and F. V. Harmelen, “OWL Web Ontology Language Overview,” W3C Recommendations, 2004, [online]: http://www.w3.org/TR/2003/WD-owl-features-20030331/

[8] Powered by MediaWiki, A List of large Triple Stores.[online]: http://esw.w3.org/topic/LargeTripleStores

[9] M. Janik and K. Kochut, "BRAHMS: A WorkBench RDF Store And High Performance Memory System for Semantic Association Discovery,” Fourth International Semantic Web Conference ISWC 2005, Galway, Ireland.

[10] C. J. Date, "An Introduction to Database Systems", Volume 1, Addison Wesley Publishing Co., Boston, MA, USA, 1990.

[11] D. J. DeWitt, "The Wisconsin benchmark: Past, present, and future," in The Benchmark Handbook for Database and Transaction Processing Systems, 1, 1991.

[12] A. Owens, "An Investigation into Improving RDF Store Performance," [Online]: *eprints.ecs.soton.ac.uk/17917/1/MiniThesis.pdf*

[13] R, Lee, "Scalability report on triple store applications," in Technical report, Massachusetts Institute of Technology, July, 2004, USA, [online]: http://simile.mit.edu/reports/stores/

[14] D. Beckett, "SWAD-Europe deliverable 10.1: Scalability and storage," Survey of free software /open source RDF storage systems. Technical Report IST-2001-34732, July, 2002, EU, [Online]. Available: http://www.w3.org/2001/sw/Europe/reports/rdf scalable storage report.

[15] D. Beckett and J. Grant, "SWAD-Europe Deliverable 10.2, Mapping Semantic Web Data with RDBMS," in Technical Report IST-2001-34732, [Online]. Available: http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report.

[16] A. Barstow, "Survey of RDF/Triple Data Stores," W3C, April 2001

[17] Y. Guo, Z, Pan and J. Heflin, "An Evaluation of Knowledge Base Systems for Large OWL Datasets," Proc. of the Third International Semantic Web Conf.(ISWC 2004), LNCS. Springer Verlag (2004)

[18] B. Liu, and B. Hu, "An Evaluation of RDF Storage Systems for Large Data Applications," skg, pp.59, First International Conference on Semantics, Knowledge and Grid (SKG'05), November 27-November 29, 2005, ISBN: 0-7695-2534-2, Beijing China

[19] K. Rohloff, M. Dean, I. Emmons, D. Ryder and J. Sumner, "An Evaluation of Triple-Store Technologies for Large Data Stores," Lecture Notes in Computer Science, 2007, ISBN: 978-3-540-76889-0, BBN Technologies, 10 Moulton St., Cambridge, MA 02138, USA

[20] T. Weithoner, T. Liebig, M. Luther, S. Bohm, "What's Wrong with OWL Benchmarks?," Proc. of the Second Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006) 101-114, Athens, GA, USA November 2006

[21] Y. Guo, Z. Pan, J.Heflin, "A benchmark for OWL knowledge base systems," Web Semantics, Services and Agents on the World Wide Web 3, pp. 158-182, Publisher: Elsevier 2005, ISSN: 15708268, Bethlehem, PA 18015, USA

[22] A. Owens, N. Gibbins, M. Schraefel, "Effective Benchmarking for RDF Stores Using Synthetic Data," in ISWC 2008: 7th International Semantic Web Conference, Karlsruhe, Germany

[23] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, S. Liu, "Towards a Complete OWL Ontology Benchmark (UOBM)," In: The Semantic Web: Research and Applications, LNCS vol. 4011/2006, pp 125-139, 2006

[24] C. Bizer, A. Schultz, "The Berlin SPARQL Benchmark," International Journal On Semantic Web and Information Systems - Special Issue on Scalability and Performance of Semantic Web Systems, 2009.

[25] C. Bizer, A. Schultz, "Benchmarking the Performance of Storage Systems that exposes SPARQL Endpoints," Proceedings of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS2008)

[26] C. Bizer and A.Schultz, "Berlin SPARQL Benchmark (BSBM)," International Journal On Semantic Web and Information Systems - Special Issue on Scalability and Performance of

Semantic Web Systems, 2009 [Online]. Available: http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/

[27] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "SP2 Bench: A SPARQL Performance Benchmark," Technical Report, arXiv:0806.4627V1 cs.DB, 2008b

[28] M. Schmidt, T. Hornung, N. Küchlin, G. Lausen and C. Pinkel , "An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario," Lecture Notes In Computer Science; Vol. 5318, Proceedings of the 7th International Conference on The Semantic Web, pp, 82 – 97, Springer-Verlag Berlin, Heidelberg

[29] Barton Library Dataset [online]: http://simile.mit.edu/wiki/Dataset:_Barton

[30] U.S. Census Dataset [online]: http://www.rdfabout.com/demo/censu/

[31] E. Codd, "A relational model of data for large shared data banks," Communications of the ACM, Vol. 13, Issue – 6, pp- 377-387, 1970, New York, Ny, USA

[32] M. Atkinson, D. Dewitt, D. Maier, F. Bancilhon, K. Dittrich, S. Zdonik, "The object-oriented database system manifesto," In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Vol. 59, pages 223-40, Kyoto, Japan, December 1989

[33] J. Broekstra, A. Kampman, F. van Harmelen, "Sesame: An architecture for storing and querying RDF data and schema information," Spinning the Semantic Web, pages 197-222, 2003.

[34] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, "Jena: implementing the semantic web recommendations," In International World Wide Web Conference, pages 74-83. ACM Press New York, NY, USA, 2004.

[35] S. Harris, N. Shadbolt, "SPARQL query processing with conventional relational database systems," Lecture Notes in Computer Science, pages 235-244, 2005, Springer Berlin / Heidelberg

[36] Allegro Graph [online]: http://www.franz.com/agraph/support/documentation/current/agraph-introduction.html

[37] O. Erling and I. Mikhailov, "RDF support in the virtuoso DBMS," In CSSW 2007

[38] Semantic web [online]: http://en.wikipedia.org/wiki/Semantic_Web

[39] J. J. Carroll, "An Introduction to the Semantic Web, Considerations for building multilingual Semantic Web sites and applications," Digital Media Systems Laboratory, HP Laboratories Bristol , HPL-2005-67, April 22, 2005

[40] Semantic web and other technologies [online]: http://www.w3.org/2008/Talks/1009-bratt-W3C-SemTech/Overview.html

[41] S. Margherita, S. Gauri, C. Pardy, J. Albert, J. Keizer, S. Katz, "Ontology-based Navigation of Bibliographic Metadta," Food, nutrition and Agriculture Journal, DRTC, February 2007

[42] K. Wilkinson, C. Sayers, H.A. Kuno, D. Reynolds, "Efficient RDF storage and retrieval in Jena2," In Proceedings of 1st International Workshop on Semantic Web and Databases (SWDB-03), pages 131-150, Septermber 7-8, 2003

[43] P. Haase J. Broekstra, A. Eberhart, R. Volz, "A Comparison of RDF Query Languages," In Proceedings of the 3rd International Semantic Web Conference, pages 502–517, ISWC 2004

[44] E. Prud'hommeaux, A. Seaborne, "SPARQL Query Language for RDF," W3C Candidate Rec. 6 April 2006. http://www.w3.org/TR/rdf-sparql-query/

[45] RDF query language [online]: http://en.wikipedia.org/wiki/RDF_query_language

[46] RDF Query Specification: http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html

[47] RDF Query Survey [online]: http://www.w3.org/2001/11/13-RDF-Query-Rules/

[48] J. P´erez, M. Arenas, C. Gutierrez, "Semantics and Complexity of SPARQL," ACM Transactions of Database Systems (TODS), Volum 34, issue 3, ISSN:0362-5915, Article No. 16, ACM New York, NY, USA

[49] A. B. Bondi, "Characteristics of Scalability and their impact on performance," in WOSP '00: proceeding of 2nd international workshop on Software and performance, page 195-203, ISBN:1-58113-195-X, New York, NY, 2000. ACM press

[50] DBLP dataset [online]: http://kdl.cs.umass.edu/data/dblp/dblp-info.html

[51] DBpedia dataset [online]: http://wiki.dbpedia.org/Datasets

[52] SIMILE Project [online]: http://simile.mit.edu/

[53] Longwell [online]: http://simile.mit.edu/wiki/Longwell

[54] MODS User guide [online]: www.modsperu.org/MODS_user_guide.pdf

[55] Database performance study [online]: http://mis.umsl.edu/bov/TuningPaperV5.pdf

[56] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, D. Reynolds, "SPARQL Basic Graph Pattern Optimization using Selectivity Estimation," Proceeding of the 17th international conference on World Wide Web, Pages: 595-604, ISBN:978-1-60558-085-2, Beijing China, ACM New York, NY, USA

[57] Jena [online] : http://jena.sourceforge.net/

[58] Jena SDB [online]: http://openjena.org/SDB/

[59] Jena TDB [online]: http://openjena.org/TDB/

[60] Sesame [online]: http://www.openrdf.org/

[61] BigOWLIM [online]: http://www.ontotext.com/owlim/big/index.html

[62] D. A. abdi, A. Marcus, S. R. Madden, K. Hollenbach "Using Barton Libraries Dataset as an RDF Benchmark," Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, MIT-CSAIL-TR-2007-036

[63] J. Broekstra, A. Kampman, "SeRQL: A Second Generation RDF Query Language [online], http://www.w3.org/2001/sw/Europe/events/20031113-storagepositions/aduna.pdf , November 2003

[64] F. Stegmaier, U. Gröbner, M. Döller, H. Kosch, G. Baese, "Evaluation of Current RDF Database Solutions" In: Proceedings of the 10th International Workshop on Semantic Multimedia Database Technologies (SeMuDaTe 2009)

[65] Aduna B. V. "User Guide of Sesame" [online], http://www.openrdf.org/doc/sesame/users/index.html

[66] M. Hausenblas, W. Slany, D. Ayers, "A Performance and Scalability Metric for Virtual RDF Graphs", Proceedings of the ESWC'07 Workshop on Scripting for the Semantic Web, SFSW 2007, Innsbruck, Austria, May 30, 2007

[67] CRUD [online] : http://www.databasejournal.com/features/mssql/article.php/3082201/Implementing-CRUD-Operations-Using-Stored-Procedures-Part-1.htm

APPENDIX

A.1. Barton Queries

Here we present the SPARQL queries over Barton dataset which we have used for testing the Read performance of each Semantic Web database. The SPARQL queries are given below with query description.

SimpleQuery: *(a) Select different types of data in store (b) Select all different subjects for object value "mods:Person" (c) Return all predicate for a subject Id.*

```
a) PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   SELECT DISTINCT ?type
   Where
   {
         ?instances rdf:type ?type
   }

b) PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>
   SELECT DISTINCT ?subject
   Where
  {
        ?subject ?someproperty mods:Person
  }

a) PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>
   PREFIX info:<info:isbn/>
   SELECT DISTINCT ?properties
   Where
   {
         info:0525070893 ?properties ?object
   }
```

ComplexQuery*: (Bushy Pattern) Extract values for recordID, genre, classification, creator, publisher, language, title, subject, contents and optionally audience, format, version, and reference for all text type items.*

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>
PREFIX role:<http://simile.mit.edu/2006/01/role/>
```

```
SELECT ?recordID ?genre ?classification ?creator ?publisher ?language ?title ?subject ?contents
?audience ?format ?version ?reference

Where
{
          ?recordID mods:records ?item.
          ?item rdf:type mods:Text.
          ?item mods:genre ?genre.
          ?item mods:classification ?classification.
          ?item mods:publisher ?publisher.
          ?item mods:subject ?subject.
          ?item role:creator ?creator.
          ?item mods:language ?language.
          ?item mods:title ?title.
          ?item mods:contents ?contents.

OPTIONAL
  {
          ?item mods:audience ?audience.
          ?item mods:otherFormat ?format.
          ?item mods:otherVersion ?version.
          ?item mods:isReferencedBy ?reference
   }
}
```

ComplexQuery*: (Long Chain) Return recordID and type of all that items that are published at more than one location.*

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>

SELECT DISTINCT ?recordID, ?type

Where
{
          ?recordID mods:records ?item.
          ?item rdf:type ?type.
          ?item mods:publisher ?publisher1.
          ?publisher1 mods:location ?location1.
          ?location1 mods:name ?name1.
          ?item mods:publisher ?publisher2.
          ?publisher2 mods:location ?location2.
          ?location2 mods:name ?name2.
          FILTER(?name1 != ?name2)
}
```

ResultSize: *Return all items of type (a) Text (b) Notated Music (c) Still Image.*

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>

    a) SELECT ?recordID
       Where
       {
               ?recordID mods:records ?item.
               ?item rdf:type ?type.
               FILTER(?type = mods:Text)
       }
    b) SELECT ?recordID
       Where
       {
               ?recordID mods:records ?item.
               ?item rdf:type ?type.
               FILTER(?type = mods:NotatedMusic)
       }
    c) SELECT ?recordID
       Where
       {
               ?recordID mods:records ?item.
               ?item rdf:type ?type.
               FILTER(?type = mods:StillImage)
       }
```

SelectivityEstimation: *Return translated title of all text type records in the data store.*

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>

SELECT  ?translatedTitle

Where
{
        ?recordID mods:records ?item.
        ?item mods:title ?title.
        ?title rdf:type mods:TranslatedTitle.
        ?title mods:value ?translatedTitle

}
```

IrregularAccessPatterns: *Select all different properties that has text type item as its domain or range.*

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>

SELECT DISTINCT ?item ?property

Where
{
        {
                ?item rdf:type mods:Text.
                ?item ?property ?object
        }
        UNION
        {
                ?item rdf:type mods:Text.
                ?subject ?property ?item
        }
}
```