RDF Overlay for HDFS and HBase

By
Muhammad Mudassar
2007-NUST-MS-PhD IT-37

Supervisor
Dr. Khalid Latif
NUST-SEECS

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(March 2011)

Approval

It is certified that the contents and form of the thesis entitled "**RDF Overlay for HDFS and HBase**" submitted by **Muhammad Mudassar** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Khalid Latif**

Signature: _____
Date: _____

Committee Member 1: **Dr. Sharifullah Khan**

Signature: _____
Date: _____

Committee Member 2: **Dr.Zahid Anwar**

Signature: _____
Date: _____

Committee Member 3: **Muhammad Bilal**

Signature: _____
Date: _____

Abstract

The concept of Semantic Web was initially proposed by Tim Berners-Lee in 1999. In Semantic Web, information is represented using specific languages, like Resource Description Framework (RDF), Web Ontology Language (OWL) etc. RDF is simple and has been standardized by World Wide Web Consortium (W3C), due to which, its usage in knowledge management applications has widely increased. So, a storage infrastructure, which should be capable to store and process large RDF dataset, is an essential need. Existing RDF processing frameworks handle small dataset efficiently, but to process large dataset, costly and high power server setup is required. There is an essential need to cope this challenge in order to provide cost effective and scalable system that can handle efficiently the massive amount of RDF data.

Distributed and parallel processing models are commonly used to process massive dataset efficiently and effectively. Hadoop is such a distributed and parallel processing open-source framework. Hadoop Distributed File System (HDFS), HBase (a distributed database of Hadoop) and Hive (a data warehousing framework) are already being used to process massive data. We developed a framework based on HDFS, HBase and Hive to store and retrieve massive RDF dataset by using cheap commodity hardware. We stored massive RDF data in HDFS and HBase to test scalability and then executed various queries to analyze performance and efficiency of our framework.

Result analysis indicated that we are able to cope with scalability issue by storing massive RDF data through configuration of few simple machines over distributed environment, and moreover, execution of various queries also proved that, our proposed framework is very effective and efficient as compared to the existing frameworks like Jena, Sesame, AllegroGraph etc.

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 NUST 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 NUST 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: **Muhammad Mudassar**

Signature: _____

Acknowledgment

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.

I would like to thank my parents for their support throughout my educational career, specially in the last year of my Master degree. They have always supported and encouraged me to do my best in all matters of life. I dedicate this work to them. I would also like to thank my brothers and sister for their love and prayers for the successful completion of this work.

My heart felt thanks to my committee members, Dr. Sharifullah Khan, Dr.Zahid Anwar, Muhammad Bilal, and to everyone at DELSA Lab, and all others who contributed in any way towards the successful completion of this thesis.

Finally, this thesis would not have been possible without the expert guidance of my advisor, Dr.Khalid Latif, who has been a great source of inspiration for me during these years of research. Despite all the assistance provided by Dr.Khalid Latif and others, I alone remain responsible for any errors or omissions which may unwittingly remain.

Muhammad Mudassar

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1999 Tim Berners-Lee introduced the concept of Semantic Web which is an extension to traditional web. Semantic Web enables computers to seek out knowledge distributed through the web. RDF (Resource Description Framework) is the standard to encode metadata and other knowledge on the semantic web. RDF consists of statements which describe a resource, its properties and some specific values of the resource against those properties. These RDF statements are often referred as "triples". A simple RDF triple consists of subject, predicate and object. Currently semantic web has a solid base of literature and developed software. Due to simplicity of data representation using RDF its use is increased in knowledge management applications. This requires a storage infrastructure capable of storing and querying large RDF datasets. The storage structures needs to be scalable enough to handle billions of RDF triples as well as efficient enough while querying this large scale RDF data. These problems can be handled by partitioning and distributing RDF data and partitioning queries and then running them in parallel over distributed datasets.

1.1 Motivation

Hadoop is an open-source software framework for reliable, scalable and distributed computing [9]. HDFS is the Hadoop distributed file system developed after idea of GFS the Google File System [14]. It is designed to run on low cost commodity hardware while keeping track of high fault tolerance. It provides facility to handle large datasets while providing high throughput access to the data. In distributed environment machine failure is very common, Hadoop handles machine failure by copying data and operations at many machines so if any machine fails data and operations can be ob-

tained from other machine. Also parallel computing is performed to enhance performance.

HBase [11] is a column oriented database which runs in the distributed environment provided by underlying Hadoop framework [9]. HBase is an open-source, distributed, and versioned column oriented store modeled after Google's Bigtable [5]. HBase is designed to work on top of HDFS and provides Bigtable like capabilities. It works efficiently with large datasets and offer real time read write access to the data. Data is made available immediately after any update performed. HBase provides storage for very large dataset and handles data efficiently in distributed environment.

MapReduce is a programming model based on functional programming model originally designed and implemented by Google for large scale data management [8]. It consists of two simple functions, map and reduce. Along with simplicity of the model it provides high degree of parallelism with little overhead. Parallel operations are performed to achieve high performance while working on the low cost commodity servers. Operations are performed transparently and independently from each other in such a way that failure of one operation does not affect performance of other operation. Today Hadoop, Google and many other organizations are using MapReduce framework to process petabytes of data on network of few thousand computers.

Semantic web provides facility to integrate information from multiple resources. Currently available semantic web databases are designed to hold small data and offer to query semantic web data but when playing with large scale semantic web data efficiency of these databases goes away. As data is growing day by day the scalability issues are arising and this challenge is also faced by semantic web applications. While designing semantic web applications, users need some storage mechanism to handle semantic web data, as it will increase rapidly. So some scalable and efficient storage mechanism is required to handle increasing amount of semantic web data. This makes people to design semantic web application without worrying about large scale semantic web data. With respect to above discussions it shows that there is a need of some storage mechanism which enables on to store massive amount of RDF data i.e. semantic web data. As Hadoop and HBase both are designed to handle massive amount of data so these can be utilized as semantic web data store. And by using MapReduce framework which is already plugged with Hadoop we can design an efficient retrieval mechanism for RDF data stored in Hadoop. It is worth that this system would result an effective increment to the semantic web world.

1.2 Problem Definition

Challenge is there to provide cost effective and scalable system which can handle massive amount of RDF data efficiently.

1.3 Proposed Approach

We will use HDFS and HBase to store large amount of RDF data. We will utilize features of HDFS and HBase along with MapReduce framework to design a RDF repository capable to store and explore massive amount of RDF data.

1.3.1 RDF Data Storage

System will be designed to store RDF triples into HDFS and HBase tables in such a way that efficient query processing is possible. In HBase we can store everything in a single table with as many columns as required. Normally RDF data is sparse but it is not a problem here because HBase adopts various compression techniques and is very good for handling null values in the table.

A single HBase table can store all data with as many columns as required. So we can make a table in which we might make a row for each RDF subject and store all properties and values as columns in the table. This reduces costly self-joins in answering queries asking questions on the same subject.

1.3.2 RDF query processing

Query processor part of the system executes SPARQL queries on RDF data stored in HDFS and HBase tables. It translates RDF queries into API calls to HDSF and HBase or MapReduce jobs which can run in parallel in the distributed environment of Hadoop. Finally results would be gathered from distributed environment recompiled if required and then presented to the user.

MapReduce framework takes RDF query from the user and divides it in number of parts or jobs and runs these jobs in parallel to achieve efficiency. Beauty of MapReduce framework is that jobs are performed in transparent manner so processing of one job does not depend on outcomes of some other job which can enhance overall query performance.

1.4 Research Contributions

This research is aimed to design an efficient and scalable storage system for semantic web data. Research objectives set to follow above stated problem statement are given below.

- To design storage system capable of storing billions of RDF triples to overcome limitation of existing systems.
- Design a system for querying the stored RDF data.
- Compare the efficiency of different storage schemas for the system.

In the light of these objectives we initially design the system by using HDFS, HBase and MapReduce framework. Then we will test the scalability and efficiency of the system by large scale open source RDF dataset. Barton library dataset is selected for our evaluation.

1.5 Thesis Outline

Rest of this thesis is organized as follows: Chapter 2 gives background knowledge for understanding this research. It also includes analysis of currently available RDF storage systems. Chapter 3 contains details of methodology we adopted. Detailed analysis of system design and implementation is presented in chapter 4. At the end evaluation and validation of results is done. Chapter 5 presents conclusions of this thesis and provide our viewpoint for future research work.

Chapter 2

Background

This chapter describes technology used in this research work with purpose of providing background knowledge, so that one can easily understand rest of this document. The problem we worked upon consists of processing massive amount of Semantic Web data. We tried to overcome the problem by using distributed software frameworks. We have utilized RDF data for this purpose which is a W3C recommendation to represent Semantic Web data. In section 2.1 we describe semantic web and how it is different from traditional web. In section 2.2 we provide a brief description of RDF. Section 2.3 describes about RDF storage mechanism. In section 2.4 we describe Hadoop framework along with MapReduce programming model. Section 2.5 reports a brief description of HBase, and finally we discuss about what is HIVE and its use with Hadoop framework and HBase in section 2.6.

2.1 Semantic Web

Semantic web which is called Web 3.0 is aimed to provide a common framework for data interchange and reuse between semantic web applications and different communities [17, 20, 3, 6] . Current web structure which is called Web 2.0 consists of only some documents which are understandable by humans only not the computers. These documents are linked with each other to make a package as shown in Figure 2.1. For this package computer just understand that is has to shift control from one place to another place by following information encoded in HTML tags, but don't understand nature and contents of the document.

Semantic web is web of resources and things rather than being web of some linked documents. It defines relationship among different resources and properties for those resources. Semantic web helps to retrieve knowledge

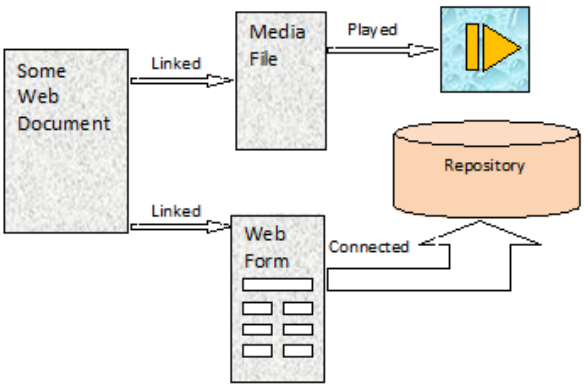

Figure 2.1: Web 2.0.

rather than shifting control from one document to another, thus allowing one to find, share and combine information more easily. This representation of data makes semantic web like an integrator which allows to interchange both data and information at same time between different applications without any lost of semantics of data.

Semantic web is targeting to provide sufficient structure to data to change it into information, so that reasonable work can be performed over data extract knowledge [11, 22]. Semantic web is targeting to represent data in such a manner that it is useful for both humans and computers at same time. Semantic web data representation techniques enable semantic web applications to extract hidden knowledge from semantic web data automatically. Semantic web languages are the universal information interchange formats used to give structures to data as well as metadata. These languages include Resource Description Framework (RDF), Web Ontology Language (OWL), and Resource Description Framework Schema (RDFS).

2.2 Resource Description Framework

Resource Description Framework (RDF) is a data model released by World Wide Web Consortium (W3C) in 1999 [1]. RDF allows to create links between semantic web data which provide inferencing facility to semantic web applications. RDF implements very minimum number of constraints to model the information. Hence we can say that, one can easily handle semantic web data in flexible manner by using RDF. With RDF information is encoded

[1]http://www.w3.org/TR/PR-rdf-syntax/

into statements called triples, these triples consists of resources their property and different values of each resource against each property. In RDF these are represented by Subject, Property and Object (SPO).

- **Resources or Subjects** are things which are described by RDF statements. A resource can be a part of a web page or whole web document or collection of documents. Each resource is assigned a unique identifier which is called Universal Resource Identifier (URI).
- **Properties or Predicates** are special kind of resources. A property describes relationship between two resources.
- Each subject contains some value against its property, these values are also known as **Objects or Values** in RDF terminologies.

Above three items are used to construct RDF triples which are also called RDF statements, and a set of such triples constitute a graph in which a triple is represented by a node-arc-node which is shown in Figure 2.2.

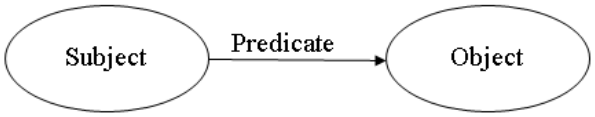

Figure 2.2: RDF Statement

This can be illustrated by an example of simple RDF statement like "Tim Berners-Lee is the director of W3C". In this statement Tim Berners-Lee is subject director is property and W3C represents value. The same triple can also be seen in terms of Subject-Predicate-Object graph as shown in Figure 2.3.

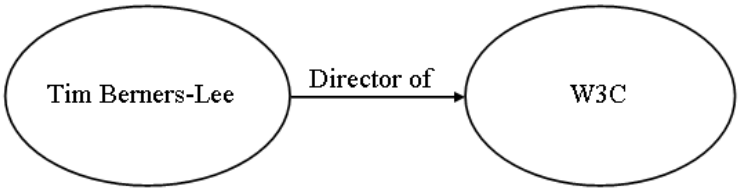

Figure 2.3: RDF Example

In RDF resource can be either a URIs, a literal or a blank node. In typical semantic web settings resource or subject of the statement Tim Berners-Lee could be represented by a URI (`http://www.w3.org/persons/TimBerners-Lee`). Choice of using URIs as standard identifiers for resources instead of simple text is because that an URI is supposed to be unique over web. Hence it is ideal to represent unique identifiers.

In XML and RDF syntax URIs used to represent unique resources are normally represented in abbreviations like someNameSpace:someThing. For example the URI "`http://www.w3.org/1999/02/22--rdf-syntax-ns#type`" is often abbrivated as *rdf:type*.

A simple RDF statement can be represented using different formats. Among most common formats are RDF/XML [2], N-Triple [3], N3 [4] and Turtle [5]. While representing RDF using XML, RDF inherits all the advantages of syntax interpretability present in XML. However RDF is not dependent on XML and can be represented in other simple formats including N3 or Turtle. Below we have given RDF information encoding of above example in different formats.

2.2.1 RDF/XML

```
<rdf:Description rdf:about=‘‘http://www.w3.org/persons/Tim Berners-Lee”>
<Director>W3C</Director>
</rdf:Description>
```

2.2.2 N3

```
< http://www.w3.org/persons/Tim Berners-Lee>
<Director>
  W3C
```

2.3 RDF Storage

RDF simplicity led to increase its use in knowledge management applications. As use of RDF increased it raised issue of persistent storage for RDF data and manipulation over data. Semantic web repositories are RDF stores where RDF data can be stored and it provides the facility to operate over stored data in order to get required results [19].

A triple store is "a system which provides persistent storage to RDF data and ease of access to the data". Main functions of such a system include storing and querying semantic web data.

[2] `http://www.w3.org/TR/rdf-syntax-grammar/`
[3] `http://www.w3.org/TR/rdf-testcases/ntriples`
[4] `http://www.w3.org/DesignIssues/Notation3`
[5] `http://www.w3.org/TeamSubmission/turtle/`

For persistent storage Relational Database Management Systems (RDBMS) can be used to store RDF data. Different approaches are being followed to design relational tables in order to store RDF data in RDBMS. In following a brief overview of each technique is given.

2.3.1 Triple tables

Triples table technique [1] is based on single table with three columns one for each component of RDF triple; subject, predicate and object. This technique is very simple. Triple tables can be created and managed easily also insertions, updating and deletions are easy to perform. Although this method is flexible but it has serious performance issues because having all the triples stored in one single RDF table requires several self-joins. Thus, as queries become more complex due to increased number of self joins the execution time will increase. Table 2.1 represents RDF triples in simple triple table approach, here if one wants to find books written by Thomsen in 2001 a three-way self-join over triple table is required. Hence performing queries over triple table is difficult.

Subject	**Predicate**	**Object**
ID1	type	BookType
ID1	title	Database
ID1	copyright	2001
ID1	author	Thomsen
ID2	owner	Thomsen
ID2	type	CDType
ID2	title	ABC
ID2	artist	Marcus
ID2	language	French
ID2	copyright	2006
ID3	type	BookType
ID3	title	signals
ID3	language	English
ID4	type	DVDType
ID4	title	Matrix
ID5	type	CDType
ID5	title	Songs
ID5	copyright	1999

Table 2.1: Triple Table

2.3.2 Property Tables

To reduce joins property table [13] technique has been proposed. Property tables denormalize RDF data to store it in wider tables. These property tables speed up the queries over triple store [13]. Two types of property tables have been proposed which are discussed below.

2.3.2.1 Clustered Property Tables

By using clustered property tables storage technique more than one tables are created to store RDF data. These tables are created based on cluster of properties that tend to be defined together [1]. For example, in Table 2.1 a cluster consisting of type, title, and copyright date properties tend to be defined as these are repeating properties of different subjects. Thus, a table called Property Table can be created containing these three properties as column headings along with subject as a primary key for the table. This table stores the triples from original data whose property is one of these three attributes. The resulting property tables, along with a table containing Left-Over triples that are not stored in this property table are shown as Table 2.2 and 2.3. More than one property tables can be created containing with different cluster of properties. But a particular property can not appear in more than one table.

Subject	Type	Title	copyright
ID1	BookType	Database	2001
ID2	CDType	ABC	2006
ID3	BookType	signals	Null
ID4	DVDType	Matrix	Null
ID5	CDType	Songs	1999

Table 2.2: Property Table

Subject	Predicate	Object
ID1	author	Thomsen
ID2	owner	Thomsen
ID2	artist	Marcus
ID2	language	French
ID3	language	English

Table 2.3: Left-Over Triples

2.3.2.2 Property Class Tables

Property-Class tables use type property of subjects to combine similar set of subjects in a same table[1]. A property may exist in more than one table. Table 2.4 and 2.5 shows property class tables and table 2.3 shows Left-Over Triples.

The most important advantage of property tables is that these can be used to reduce subject-to-subject self joins [1]. Property tables have not been widely implemented as these have number of disadvantages like more wider tables contain more NULL values for properties that are not defined for certain subjects. So these tables introduce a space overhead. But on other

hand, joins and unions are increased for short tables. Queries that have not involved select on class type are problematic for property-class tables. Moreover handling multi-valued properties is also a difficult task. There is no algorithm which always yields property tables of perfect width for all queries.

Subject	Author	Title	copyright
ID1	Thomsen	Database	2001
ID3	Null	signals	Null

Table 2.4: Class BookType

Subject	Author	Title	copyright
ID2	Marcus	ABC	2006
ID5	Null	Songs	1999

Table 2.5: Class CDType

Subject	Predicate	Object
ID2	owner	Thomsen
ID2	language	French
ID3	language	English
ID4	type	DVDType
ID4	title	Matrix

Table 2.6: Left-Over Triples

2.3.2.3 Vertical Partitioning

In vertical partitioning approach simple three column table is rewritten into n two column tables where n is number of unique properties in RDF data [1]. First column contain subjects for that particular property and second column contain objects of those subjects as shown in Figure 2.4.Each table is sorted by subject. It handles unstructured RDF data and null values efficiently.

2.3.2.4 RDF Data extraction through SPARQL

For every data repository there are mechanisms defined to extract desired results from the store. This needs some query standard to be defined. A semantic web query language is a language used to retrieve required results and manipulate data stored in semantic web language format [2]. RDF data model is somewhat different then other data models used for knowledge representation so its querying mechanism is also little different than ordinary query processing mechanism which is SQL. Semantic web query language is

Subject	Type
ID1	BookType
ID2	CDType
ID3	BookType
ID4	DVDType
ID5	CDType

Subject	Title
ID1	Database
ID2	ABC
ID3	signals
ID4	Matrix
ID5	Songs

Subject	copyright
ID1	2001
ID2	2006
ID5	1999

Subject	author
ID1	Thomsen

Subject	Artist
ID2	Marcus

Subject	language
ID2	French
ID3	English

Subject	owner
ID2	Thomsen

Figure 2.4: Vertical Partitioned Table

little bit more complex than normal SQL as underlying RDF data model is complex than relational data model.

Several design and implementations have been purposed for querying semantic web data [25, 26]. SPARQL is one of these which is also a W3C recommendation for querying RDF data [27, 15]. SPARQL is a graph matching query language. In order to query some RDF data there exists some pattern in SPARQL body which is matched with RDF data set, and successful mappings are returned as the result set.

An example SPARQL query to retrieve the name of director of W3C is given below:

```
SELECT ?name
WHERE
{ ?name <http://www.w3.org/persons/Tim Berners-Lee > ''W3C" }
```

In SPARQL variables are denoted with '?' or '$' sign placed before the word and result set constitute of those variables which are defined in SELECT part of the query. WHERE part of query also called pattern matching part includes triple patterns which are to be compared with RDF data set. This part might contain some optional triple patterns, union of patterns, filters and nested queries also. SPARQL also supports classical operators like projection, distinct, order, limit and offset. SPARQL also include yes/no queries

and queries for description of resources

2.4 Hadoop Framework

The Hadoop framework [2] is an open source software project hosted by Apache software foundation, and it is developed in Java. Hadoop provides facilities to handle massive amount of data in terms of storage and processing. For storage it uses its own Hadoop Distributed File System (HDFS) [10], and for processing massive amount of data stored in HDFS it has implemented MapReduce programming model [8].

Hadoop framework runs over a network of commodity computers and it uses HDFS for data storage. In distributed file system of Hadoop data can be stored in files which are replicated over different computers. This replication of files allows Hadoop to recover data in case of failure. So Hadoop is an ideal candidate for building massive data repositories. Hadoop features include high fault tolerance, scalability and reliability.

Hadoop also provides software framework for the famous parallel data processing model called MapReduce [8].The MapReduce framework allows to write applications to process massive amount of data in parallel on large clusters of commodity nodes. A typical MapReduce job reads the input from HDFS, process it and writes the results back to file system. HDFS does not only provide data storage but it is heavily integrated with Hadoop and is easy to install and configure.

Four different types of programming modules are running in Hadoop. Two of them are related to distributed data storage commonly known as "NameNode" and "DataNode", the other two are related to distributed processing called "JobTracker" and "TaskTracker". These are shown in Figure 2.5. NameNode acts as a master and take cares of the replication of the data blocks, organizing and keeping track of nodes activity and capacity of data storage. DataNode is responsible for actual storage of data blocks. DataNodes are also responsible for serving client requests related to read and write files from the file system. DataNode also performs block creation, deletion and replication upon instruction from NameNode. JobTracker also acts as master who submits and coordinates different jobs and their execution on different computational units. TaskTracker is the module that performs actual job computations. A TaskTracker accepts tasks that are assigned to it by the JobTracker and report results obtained back to him. Normally every node in the Hadoop network acts as a TaskTracker. We can say that it is a worker in the Hadoop network.

Distributed storage, replication, scalability and parallel execution of tasks

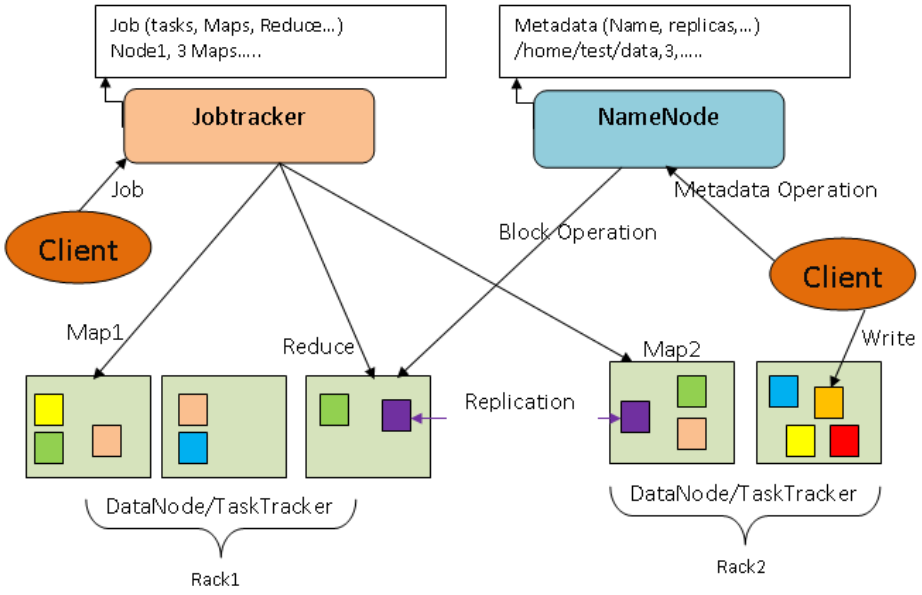

Figure 2.5: Hadoop Structure

make Hadoop framework an ideal candidate for building a storage system, especially when one has to handle massive amounts of data.

2.5 HBase

HBase [12] is a column oriented database which runs in distributed environment provided by underlying Hadoop framework [**?**]. HBase is open-source distributed versioned column oriented store modeled after Google's Bigtable [14]. HBase provides Bigtable like capabilities on top of HDFS. It works efficiently with large datasets and offer real time read write access to the data. Data is made available immediately after any update performed.

HBase provides storage for very large dataset and handles data efficiently in distributed environment. The underlying hardware of HBase consists of clusters of commodity servers. In distributed environment machine failure is very common, HBase handles machine failure by copying data and operations at many machines so if any machine fails data and operations can be obtained from other machine. Also parallel computing is performed to enhance performance. As it is column major database high compression rate can be achieved on per column basis.

HBase stores data in from of sort able row key and predefined column families and an arbitrary number of columns defined inside column families. More over data is time stamped to track the changes in data. A data row in HBase is composed of row keys which are sortable, and it falls under some column which is a member of some family. A column is represented as ColumnFamilyName:ColumnName. A data cell can contain multiple versions of data by attaching timestamp.

2.6 Hive

Hive [13] is a data warehouse infrastructure build on top of Hadoop to analyze the data stored in HDFS and HBase. It helps to format data according to some defined structure, it also provides capability to query large datasets stored in Hadoop. Hive defines simple SQL-like query language, which is called QL, by using QL user can perform queries to stored data. Hive also allows building custom defined MapReduce jobs which can be plugged in Hive QL.

Hive allows to structure data according to al-ready well known database concepts like tables, columns, rows and partitions. It supports all the major primitive data types including integer, float, double, and string. Hive stores data in tables as traditional databases, where each table consists of rows and columns. While tables are logical data units in Hive table metadata associates the data in a table to HDFS directories. HiveQL query language is very similar to SQL and therefore one can easily perform queries over massive data. Traditional SQL features like from clause sub-queries, various types of joins, cartesian products, group-bys and aggregations, union all, create table as select and many useful functions are very SQL like. This enables anyone who is familiar with SQL to start a hive CLI (command line interface) and start browsing data stored in Hadoop. There are some limitations e.g. only equality predicates are supported in a join and joins have to be specified using the ANSI join syntax such as

```
SELECT DISTINCT t1.subj, t2.prop
From barton_1 t1 JOIN barton_1 t2 ON (t1.Subj=t2.Subj)
Where
t1.Prop='http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND
t1.Obj='http://simile.mit.edu/2006/01/ontologies/mods3#Text'
```

While in SQL a join is performed as follows.

```
SELECT  t1.subject
From Triple_Table AS t1, Triple_Table AS t2
Where
t1.object=t2.subject
```

Another limitation is in how inserts are done, hive does not support inserting into an existing table, all inserts done overwrite existing data. This all is because till now Hive is no a mature query language and work is in progress on this project.

These restrictions have not been a so much problem, as in reality there are rare cases where a query cannot be expressed as an equi-join, and we simply loaded the data in to a new table every time.

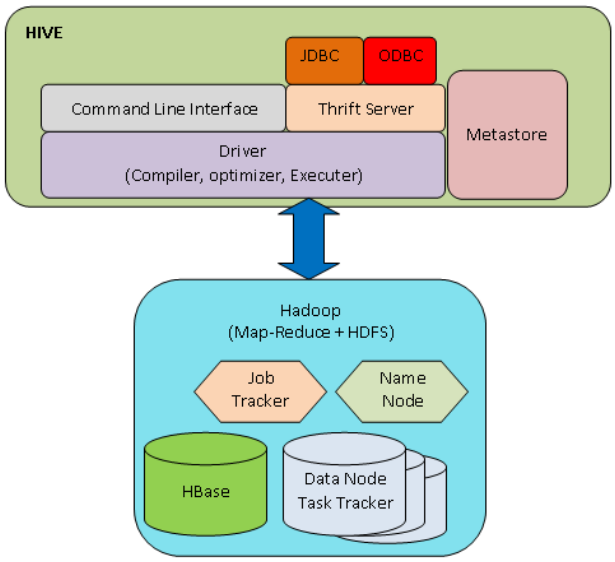

Figure 2.6: Hive System Architecture

Following are components are main building blocks in Hive:

- **Metastore** is used to store system catalog and metadata about tables, columns, partitions etc.
- **Driver** manages lifecycle of HiveQL statements as it moves through Hive.
- **Query Compiler** is a component which compiles HiveQL into a directed acyclic graph of MapReduce tasks.
- **Execution Engine** execute tasks produced by compiler in proper dependency order while interacting with underlying Hadoop instance.
- **Hive Server** provides the thrift interface and a JDBC/ODBC server to operate it from other applications.

- Client components like **Command Line Interface (CLI)**, JDBC and ODBC drivers.

Chapter 3

Methodology

As discussed in earlier chapters that by distributing storage and processing mechanism we can handle massive amount of RDF triples. This chapter will focus on how this can be achieved by using a distributed framework called Hadoop. We will describe how we will configure our distributed setup to achieve the goal, and after that how RDF data is arranged to store in distributed file system and distributed database i.e. HDFS and HBase respectively. Finally how we process data stored in Hadoop framework will be discussed. We will talk about the SPARQL transformation to query RDF data stored in HDFS and HBase.

3.1 Hadoop Setup and Data Storage

3.1.1 HDFS Setup and Storage

HDFS is distributed file system of Hadoop framework[10]. It provides the storage infrastructure to Hadoop applications. HDFS allows storing data as normal files like we can store on any operating system Linux or Windows. HDFS consists of one NameNode and multiple DataNodes, NameNode acts as a server of storage system that manages file system metadata and also regulates access to files by client. DataNodes are slave applications which run on ordinary commodity machines and storage devices attached to these commodity DataNodes are used to store data. All distribution of data is managed by NameNode so user is freed from keeping record of which data is stored on which node. Data given to HDFS is splitted into one or more blocks and these blocks are stored in a set of DataNodes. A block is replicated on more than one node for easy recovery and paralyzation of operation in case of any node failure. NameNode is responsible to execute file system operations

like opening, closing and renaming files and directories, NameNode also keeps track of mapping blocks to DataNodes. Datanodes are responsible for serving read and write requests, Datanodes also performs block creation and deletion when NameNode instructs them to do so.

HDFS provides reliable storage for very large files across different machines in a large cluster and it stores each file as a sequence of equal size blocks, and block size is configurable. Blocks of a file are replicated on multiple DataNodes (replication factor is also configurable) for fault tolerance. NameNode makes all decisions of data blocks replication. NameNode remains in contact with DataNodes in cluster by receiving blockreports and heartbeats from each of DataNode in Hadoop network. Heartbeat from a DataNode shows that node is working properly, while blockreport contains list of all blocks placed on specific DataNode.

We will configure the HDFS cluster on ordinary machines running Ubuntu as Linux operating system on them. One machine is used as NameNode configuration and it will act as the storage server of HDFS cluster, number of machines will be configured as DataNodes. All DataNodes will be configured to utilize all available space attached to them. HDFS is allowed to use default block size of Hadoop framework which is 64MB, as it works well because it is not too small not a larger one. Block size has a definite impact on performance.

HDFS offers to store data in any format like in form of tables, plain files or CSV files etc. when it comes to store RDF data in HDFS we have chosen CSV format to store RDF data in HDFS. These CSV files contain RDF statements in the form of triples table. By using Jena [4] we can extract RDF triples from RDF files and from these triples subject, predicate and objects can be extracted very easily. So CSV files are created which have stored RDF data in them like a triples table. For each file we have defined start of new triple pattern by setting parameter of row format as delimited at time of storing data, plus we have defined start of new field by putting some unique character between each field.

Hive provide ways to upload data to HDFS but one has to configure proper parameters in order to allow Hive to insert data. These configurations are given in below section. So we have used Hive bulk load commands to store RDF triples from CSV files to HDFS.

3.1.2 Hadoop MapReduce Setup

MapReduce is a programming model used to process large datasets in parallel environment. This programming model is originally presented by Google [8]. MapReduce can process data either stored in a filesystem of in a database.

MapReduce splits the problem into two stages map stage and reduce stage, in first stage map method is executed on TaskTracker. Master node i.e. JobTracker takes input job partitions it into smaller sub-problems, and distribute these sub problems among worker nodes i.e. TaskTracker in Hadoop network. Worker node execute job and answer back to JobTracker. After getting results of map execution master node instructs TaskTrackers to perform reduce step which is the 2nd stage of a MapReduce job. Reduce operation is performed on the output of map processing.

Hadoop framework follows rule of "moving computations is cheaper than moving data". This is especially true when input dataset is large enough. So a computation request by an application in form of MapReduce jobs is executed on that node where the required data is residing. That's why in Hadoop cluster DataNodes and TaskTrackers are configured on same machine.

Number of map and reduce operations are not fixed in a MapReduce job, so number of map jobs may vary than number of reduce jobs. During a MapReduce job all map operations executed on worker nodes are independent of each other so all maps can perform in parallel. Results of these map operations are stored on local hard disk. After completion of map operations TaskTracker informs jobTracker about completion and JobTracker then instructs to move results to nodes where reduce operations needed to be performed. After completion of map operations reduce operations are carried out on output of map operations. If no reduce operation is defined then after successful completion of map operations job is completed.

3.1.3 HBase Setup and RDF Storage Model

HBase is distributed, column oriented, versioned database of Hadoop framework. HBase provides Bigtable [5] like storage on top of Hadoop distributed computing environment. As HBase is inspired by Bigtable its data model is also very similar as that of Bigtable. As HBase is distributed over commodity machines but at same time it is also tolerant of machine failures. Data can be stored in labeled tables, data in a table consists of a sortable row key and some predefined "column families" and an arbitrary number of columns. In a table a row can have widely varying number of columns. A column name consists of "<family>:<label>". Number of <family>s ("column families") is pre defined at table creation time and can't be altered later on. New ¡label¿s can be created as required. Data stored in a same column family are physically close on hard disk while data in different column families might be distributed over underlying Hadoop network. Items in one column family contain similar data. Only a single row at a time is locked to insert, update

Row Key	Column Family "rdf:"	
Some unique value	"rdf:subject"	"some value"
	"rdf:predicate"	"some value"
	"rdf:object"	"some value"

Table 3.1: HBase Data Model

and retrieve operations. The HBase data model can be better understood by Table 3.1.

An application views a HBase table as list of tuples which are sorted in ascending order of row key. Physically tables are broken up into row ranges, these broken tables are called "regions". Each region contain some specific number of rows, HBase identifies a region by table name and start key. Figure 3.1 describes this process.

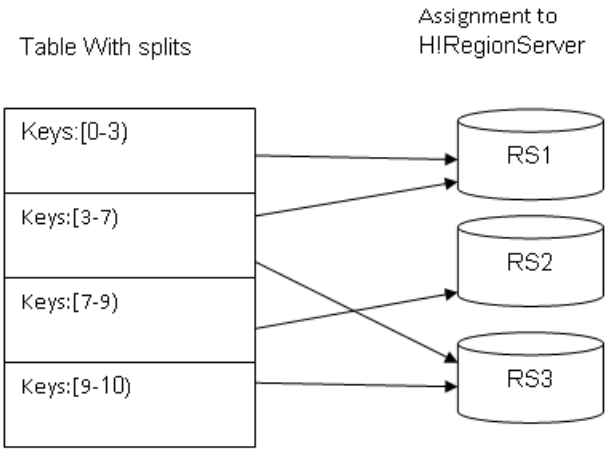

Figure 3.1: HBase Tables and Regions

Regarding the HBase cluster setup there are three major components of HBase H!BaseMaster, H!RegionServer and HBase client. H!BaseMaster is responsible to assign data in form of regions to worker nodes i.e. H!RegionServers. H!BaseMaster also monitors health of each H!RegionServer and if a H!RegionServer is dead the H!BaseMaster will reassign regions served by dead H!RegionServer to some other H!RegionServer. If H!BaseMaster dies cluster will shut down. H!RegionServer is responsible for handling read and write requests of clients. It remains in contact with H!BaseMaster by using heart beat messages. When it needs to process some request it contacts H!BaseMaster to get a list of regions to be served. These requests are either read requests or write requests.

To store RDF data in HBase a single table based on single column family

RDF Table		
ROW Key	**Column Family "rdf:"**	
1	"rdf:subject"	"http://libraries.mit.edu/barton/MCM/000000504"
	"rdf:predicate"	"http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
	"rdf:object"	"http://simile.mit.edu/2006/01/ontologies/mods3#Record"
2	"rdf:subject"	"info:isbn/0442215312."
	"rdf:predicate"	"http://simile.mit.edu/2006/01/ontologies/mods3#language"
	"rdf:object"	"http://simile.mit.edu/2006/01/language/iso639-2b/eng"

Table 3.2: RDF Data Model in HBase

will be created. This column family will contain three columns further one for each subject, predicate and object. Each RDF triple will be assigned a unique number while loading RDF triples to HBase, these unique numbers will act as row key for against each triple. HBase allows querying to data stored in tables by using the row key. RDF triples stored in HBase consists of full URI along with value. So there is no need to store long URIs separately which results in simple schema, single table updations and retrievals, no need to care extra tables, it will also reduce number of joins with other tables to get long or short URI values.

Logical view of our RDF table after storing RDF data in it will look like Table3.2. For Table 3.2 one can access data by using table name its row key and ColumnFamily plus column name. So to access "info:isbn/0442215312." one can give command on HBase Shell *get 'RDF Table', '1','rdf:','subject'*

3.2 Querying

As discussed in Chapter 2 that Hive provides facility to query data stored in Hadoop framework including HDFS and HBase. Hive allows running SQL like queries. As we are working with RDF dataset so the query standard recommended by W3C is SPARQL. We have to transform SPARQL queries to Hive QL. This also involves checking that either the transformed queries are correct one or not for this purpose we have to analyze the results of SPARQL and hive queries

3.2.1 Hive Setup

Hive provides CLI in order to execute queries, it also allows to configure Hive server. By using this server and Hive drivers one can also perform queries from any Java based client. Hive automatically set number of MapRreduce jobs against Hive queries submitted. Number of MapReduce operations depends on complexity of query like numbers of joins in a query effects number of MapReduce operations, number of string comparisons effects numbers of

map operations. Simple select queries with no join condition present are solved by simply executing map operations.

Query execution time depends on total number of MapReduce jobs, queries having more number of jobs take more time. It also depends on HDFS replication factor, if replication factor is higher more number of parallel operations can be performed which results in decreased execution time.

Hive requires a pointer to be defined in hive configuration about current NameNode instance running in Hadoop cluster along with its port. At same time hive directories needed to be created in HDFS where Hive will store its tables and views of HBase tables.

3.2.2 Querying HDFS

In order to perform queries to the files stored in HDFS one have to write his own MapReduce application and then run these application on Hadoop (HDFS+MapReduce) cluster. While Hive provides facility to perform SQL like queries to data stored in HDFS. At the same time Hive also provides facility to insert data to HDFS, but for this hive needs some serialization format to be defined while inserting data to HDFS. Over these serialized files Hive allows to execute select, overwrite, aggregation, union all and other SQL like queries. To query RDF data we will first store the RDF data to HDFS as described in Section 3.1.1. After loading RDF data to HDFS we can execute Hive queries over RDF data, these Hive queries are counterparts of SPARQL queries.

3.2.3 Querying HBase

HBase does not provide any query language to explore data stored in HBase tables. Instead it offers "get" and "insert" commands from HBase shell and from "get" and "put" methods if someone is using HBase client API. These commands or methods are restricted to work on a single row at a time, which means that one can obtain or post only single row at a time. Also if someone needs to perform join over HBase tables he has to write his own application to get desired results.

Hive allows to execute SQL like queries over HBase tables. But for Hive requires HBase table's views to be defined in Hive. For this purpose Hive comes with HBase handle which contains jars of running HBase instance plus configurations of HBase along with pointer to current H!BaseMaster along with its port on which it is running. Hive provides the commands to create views of HBase tables but one have to be careful about mappings of HBase column families and columns to tables of Hive.

Chapter 4

Implementation and Results

This chapter describes implementation details of our work, tests performed and results achieved. Section **??** contains an overview of dataset used. Section **??** covers the SPARQL queries we have used. In Section 4.3 Hive queries are given. Section 4.4 describes the testing environment setup and configurations. Results have been discussed in Section and at last Section 4.6 consist of comparisons.

4.1 Barton Data Set

Purpose of this research is to design a scalable semantic web repository where massive amount of RDF data can be stored and processed efficiently. For this purpose some large size real and public data set is required. There are different RDF datasets are available for testing purpose like Barton libraries dataset [23], DBLP [16] and DBpedia [7] datasets. These RDF datasets are very commonly used for semantic web repositories evaluation.

The Barton libraries dataset [23] which is provided by the Simile Project [24], contains the records that compose and RDF-formatted dump of MIT Libraries Barton catalog. This dataset is a large size dataset and contains irregular structures. Barton library dataset was derived from multiple sources and it follows a semantically rich ontology. This dataset contains more than 25 million triples. Similar string at start of properties belongs to predefined schemas like RDF, OWL and some other schemas. In a document these are declared as namespaces. The Barton dataset namespaces are given in Table 4.1.

We have created three different size partitions of Barton data set. These partitions contain 1 Million, 5 Million and 25 Million triples.

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#"

Table 4.1: Barton Prefix Against URI

4.2 SPARQL Queries

To query RDF data stored in distributed database we have executed the queries already defined [21]. These queries were performed to analyze performance of semantic web stores against Barton Libraries dataset. The SPARQL queries are given below with description of each query. First three queries are the simple queries, these queries consists on single triple pattern given in WHERE clause.

- **Query1**

  Query 1 selects different type of data by comparing the resultant triple with the given predicate which is *rdf:type* the query is given below.

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

- **Query2**

  The Query 2 selects all distinct subjects of any property containing the value of *mods:Person* in the object part of the RDF triple. This query returns a lot of triples in the result set.

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

- **Query3**

  The Query 3 returns all the predicates against the give subject value.

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

- **Query4**

  The Query 4 involves more than one triples in the *WHERE* claus of the query, this query finally returns all items of *Text* type. This query also returns a large number of result set.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>
SELECT ?recordID
Where
{
?recordID mods:records ?item.
?item rdf:type ?type.
FILTER(?type = mods:Text)
}
```

- **Query5**

  Query 5 is similar to Query 4 but it selects all items of *NotatedMusic* type also the result set of the query is normal.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>
SELECT ?recordID
Where
{
?recordID mods:records ?item.
?item rdf:type ?type.
FILTER(?type = mods:NotatedMusic)
}
```

- **Query6**

  Query 6 emphases to select items of type *StillImage*, the result set for this query contains a small number of records.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mods:<http://simile.mit.edu/2006/01/ontologies/mods3#>
SELECT ?recordID
Where
{
?recordID mods:records ?item.
?item rdf:type ?type.
FILTER(?type = mods:StillImage)
}
```

- **Query7**

  Query 7 is the selectivity estimation query which returns translated titles of all *text type* records. Join performance is also checked by this query.

```
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
}
```

- **Query8**

  This query involves a union of two sub-queries. This query also return huge size of output.

```
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
}
}
```

4.3 Hive Queries

This section contains the SQL like queries of above SPARQL queries which we have executed using Hive CLI over RDF data stored in HDFS and HBase. For given queries we have assumed the table name as *TripleTable*.

To test that either our Hive queries are exactly equal to their respective SPARQL queries, we have compared the results obtained by running SPARQL queries over plain RDF files using Jena [4] with the results obtained using Hive over RDF data stored in HDFS and HBase. For each result set comparisons we found that these are same, this testing confirmed that our Hive queries are exactly equal to their respective SPARQL queries.

- **Query1**

```
SELECT DISTINCT object
From TripleTable
WHERE
property='http://www.w3.org/1999/02/22-rdf-syntax-ns#type'
```

- **Query2**

```
SELECT DISTINCT subject
From TripleTable
WHERE
object='http://simile.mit.edu/2006/01/ontologies/mods3#Person'
```

- **Query3**

```
SELECT DISTINCT property
From TripleTable
WHERE
subject='info:isbn/0525070893'
```

- **Query4**

```
SELECT t1.subject
From TripleTable t1 JOIN TripleTable t2 ON (t1.object=t2.subject)
WHERE
t1.property='http://simile.mit.edu/2006/01/ontologies/mods3#records' AND
t2.property='http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND
t2.object='http://simile.mit.edu/2006/01/ontologies/mods3#Text'
```

- **Query5**

```
SELECT t1.subject
From TripleTable t1 JOIN TripleTable t2 ON (t1.object=t2.subject)
WHERE
t1.property='http://simile.mit.edu/2006/01/ontologies/mods3#records' AND
t2.property='http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND
t2.object='http://simile.mit.edu/2006/01/ontologies/mods3#NotatedMusic'
```

- **Query6**

```
SELECT  t1.subject
From TripleTable t1 JOIN TripleTable t2 ON (t1.object=t2.subject)
WHERE
t1.property='http://simile.mit.edu/2006/01/ontologies/mods3#records' AND
t2.property='http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND
t2.object='http://simile.mit.edu/2006/01/ontologies/mods3#StillImage'
```

- **Query7**

```
SELECT t4.object
From TripleTable t1 JOIN TripleTable t2 ON (t1.object=t2.subject) JOIN TripleTable t3 ON (t
WHERE
t1.property='http://simile.mit.edu/2006/01/ontologies/mods3#records' AND
t2.property='http://simile.mit.edu/2006/01/ontologies/mods3#title' AND
t3.property='http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND
t3.object='http://simile.mit.edu/2006/01/ontologies/mods3#TranslatedTitle' AND
t4.property='http://simile.mit.edu/2006/01/ontologies/mods3#value'
```

- **Query8**

```
SELECT *
From (
SELECT DISTINCT t1.subject, t2.property
From TripleTable t1 JOIN TripleTable t2 ON (t1.subject=t2.subject)
WHERE
t1.property='http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND
t1.object='http://simile.mit.edu/2006/01/ontologies/mods3#Text'

UNION ALL

SELECT DISTINCT t1.subject, t2.property
From TripleTable t1 JOIN TripleTable t2 ON (t1.subject=t2.object)
WHERE
t1.property='http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND
t1.object='http://simile.mit.edu/2006/01/ontologies/mods3#Text') unionResult
```

4.4 Test Environment Setup

This section describes experimentation setup including hardware and software. We have performed our experiments in two environments, stand-alone mode and distributed mode. In stand-alone mode Hadoop Framework including HDFS and HBase are configured on single machine while for distributed setup we have used three machines of same configurations connected with each other over 1 GBPS network. For this purpose we have utilized ordinary machines running Linux (Ubuntu) as operating system. Each machine is a Dell Optiplex 760 Core 2 Due @3 GhZ with 2 GB RAM.

For each Ubuntu machine Java version 6 is installed which is required by Hadoop framework. Each node is configured such that it can perform password-less ssh to localhost as well as other nodes running Hadoop framework, this is also required by Hadoop framework in order to communicate with localhost as well as to other Hadoop nodes. Hadoop also uses DNS server setup to map machine names with their IP addresses as Hadoop framework uses host names rather than IP addresses to communicate with other machines, we have configured master node to provide this service also.

We have used Hadoop version 0.20.2 which is the most stable release of Hadoop framework. Each Hadoop release consists of HDFS and MapReduce instances, we have configured both of these instances. We have used HBase version number 0.89 the most latest version in which almost all bugs of previous versions are fixed. In order to perform query analysis we have use Hive 6.0 version, we have downloaded the source code from given SVN repository and complied the code at our side to build the project. This compilation also includes the HBase handle so that by using Hive we can execute queries on data stored in HBase tables.

4.5 Results and Discussions

4.5.1 Data Load Results

In This section contains results of data loading in HDFS and HBase in both single node and distributed setup. Figure 4.1 shows the load times analysis of different sized datasets. The experiment showed that in standalone mode it takes a bit more time to load the data in HDFS because all of the data has to be stored in single instance of HDFS while in distributed setup data is stored in form of chunks on some of nodes but the replication factor slows down the whole process so not making a too much difference. While on the other hand HBase is showing a little opposite response for small and medium

size dataset but it showed little performance gain in case of large dataset.

It is clear from the given results as more nodes will be added to the Hadoop cluster the loading performance will increase and also more scalability will be achieved. For large number of nodes issues of node failures will be there but this can be handled by increasing the replication factor. Currently the replication factor is 3.

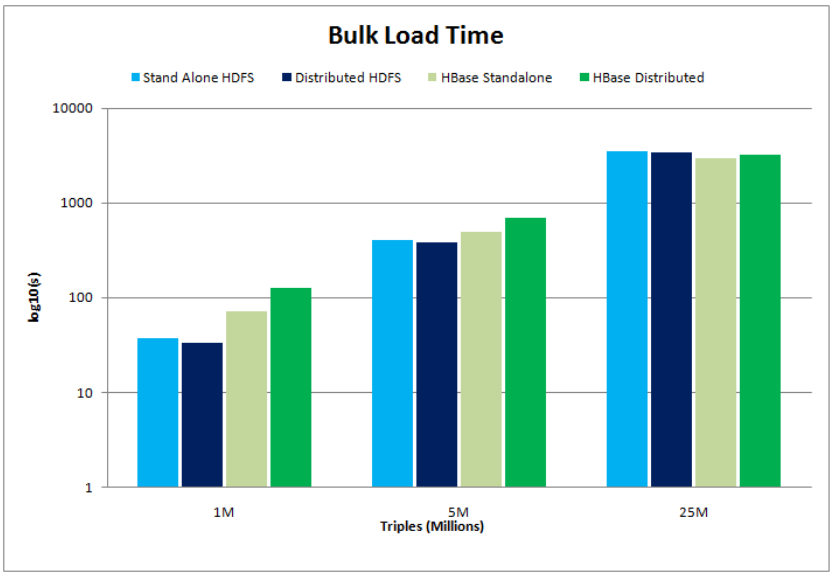

Figure 4.1: Bulk Load Time

4.5.2 Queries Results

Query response times of the queries discussed in Section 4.3 for HDFS and HBase are described here and a brief discussion of each query's behavior is given below. From the results it is clear that HDFS on distributed setup gives better results than on single node setup. Because in distributed environment more replicas of the data are present so jobs can run in parallel on different nodes hence increase in performance. While HBase showing some poor performance for simple queries because there is an extra interface present between HBase and query processer Hive, this extra interface is Hadoop File System.

First, second and third queries are simple queries based on single triple pattern search against some specific values of predicate, object and subject respectively. Query results show clearly that both HDFS and HBase are

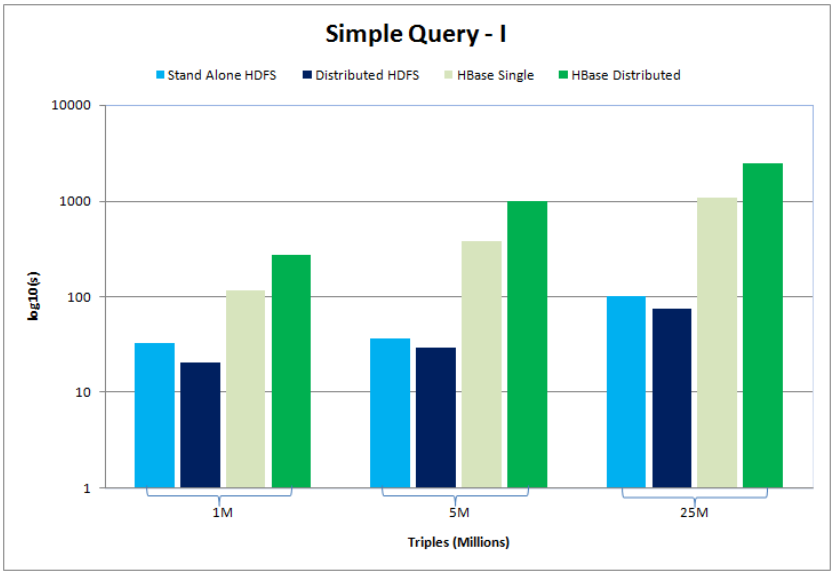

Figure 4.2: Query 1 Response Time

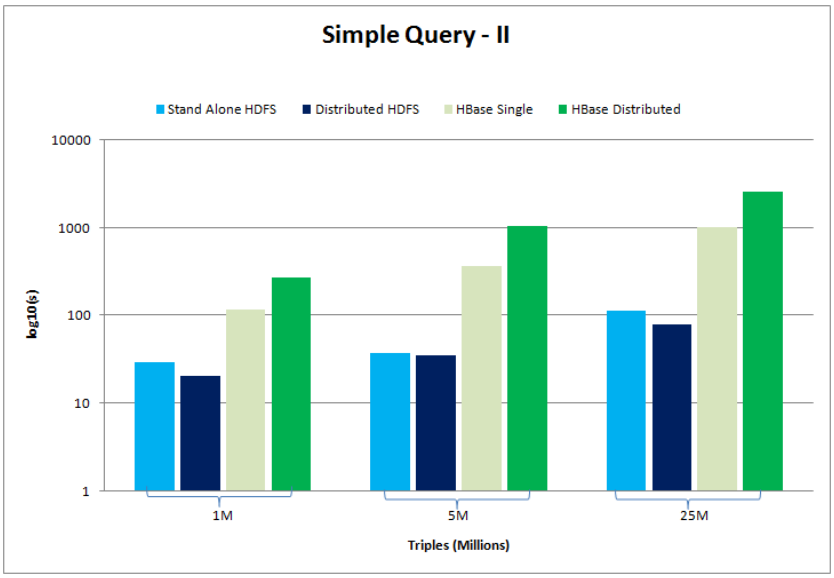

Figure 4.3: Query 2 Response Time

showing similar response for different variable search. Figures 4.2,4.3 and 4.4 shows this clearly that query response time in distributed setup of HDFS is better than single node setup. Also there are less number of MapReduce

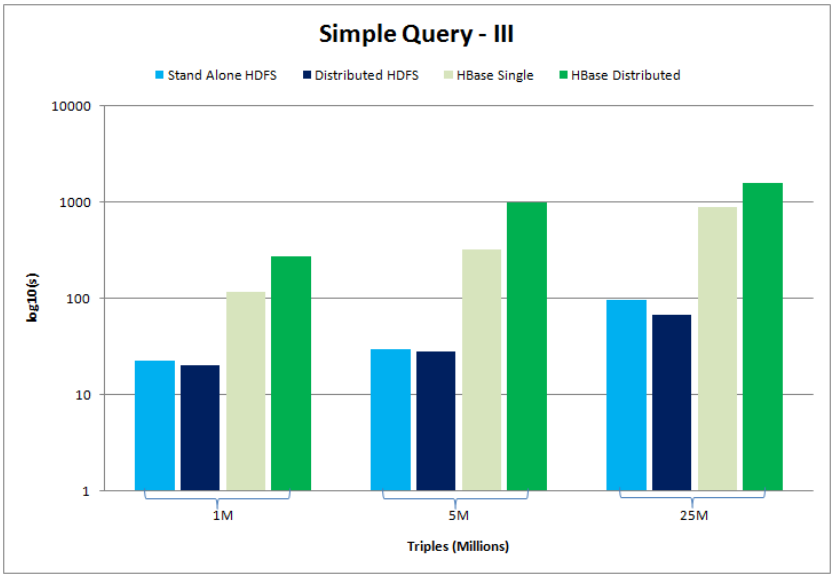

Figure 4.4: Query 3 Response Time

operations are involved in the query as it contains only a single triple in its WHERE clause. We also observed that these simple queries were completed by running map functions only, but when we instructed our system to store output results to some other table then join function was called to store the outputs of query to said table.

Queries 4,5,and 6 are used to check system against output size of each query hence we named these queries as "Result Size". All three queries i.e. Result Size-I,Result Size-II and Result Size-III have returned large medium and small results respectively. Figures 4.5,4.6 and 4.7 shows that HDFS and HBase shows almost similar behavior either output size are large, medium or small respectively. If we look specifically HDFS results graph shows that for large data sets HDFS gives better results in distributed environment.

Query 7 checks join performance of system it involves a number of joins which filter out the results, hence the result set of the query is a small. But as compared to previous queries the number of MapReduce jobs for this query are increased which which measures efficiency of Hadoop file system. Results showed that both HDFS and HBase successfully executed complete query. While running this query we observed that some times some map function failed to execute on one node, but Hadoop managed to launch similar map function on some other node and at the end finishing query successfully. So we can say that HDFS and HBase both, always managed to complete the query successfully. Figure 4.8 shows results of this query.

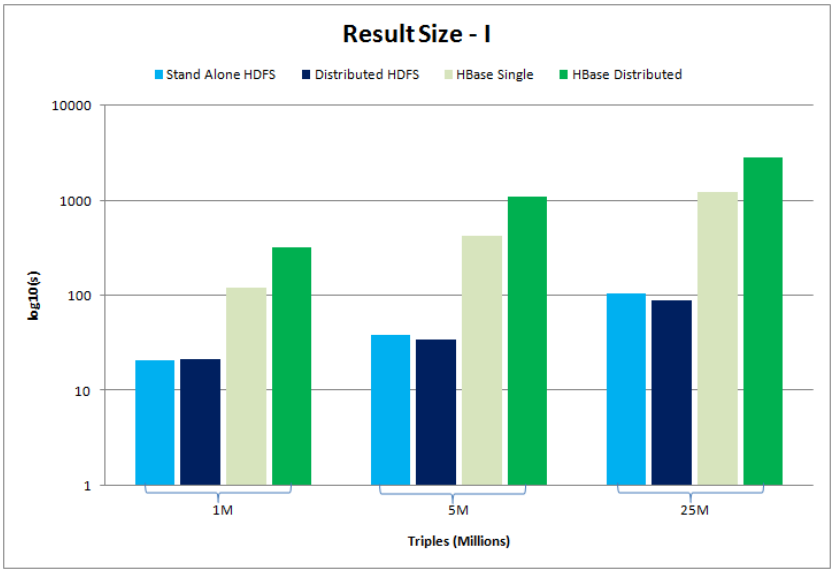

Figure 4.5: Query 4 Response Time

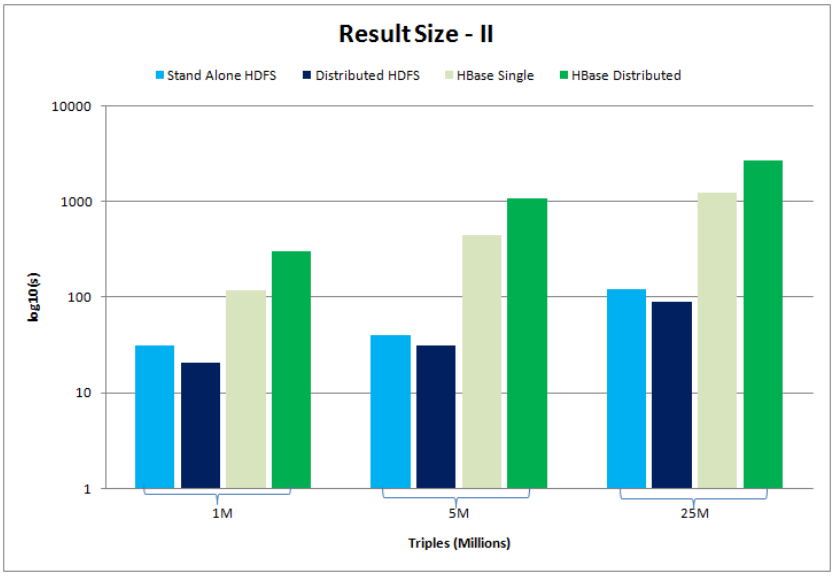

Figure 4.6: Query 5 Response Time

Query 8 is the real test of Hadoop MapReduce framework as it runs 6 number of MapReduce jobs which is the higher one among all the queries we have executed. Plus this query also consists of union of two sub queries.

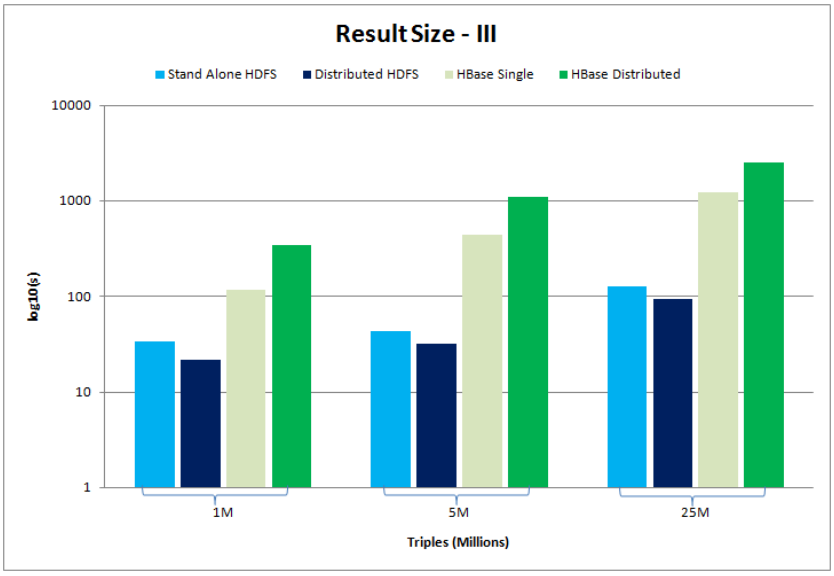

Figure 4.7: Query 6 Response Time

Figure 4.8: Query 7 Response Time

Results in Figure 4.9 showed that this query managed to run successfully with comparatively very less response time for large data set. By observing health of nodes while running this query we find that all the nodes are work-

ing at their full capacity, because each time a node completes some task, JobTracker assigns it some other task. So we can verify that JobTracker efficiently manages all jobs assigned to it, and that it is capable to utilize all distributed nodes in parallel. Hence we can say storing massive RDF data sets in HDFS and HBase results in achievement of scalability factor as well as efficiency also.

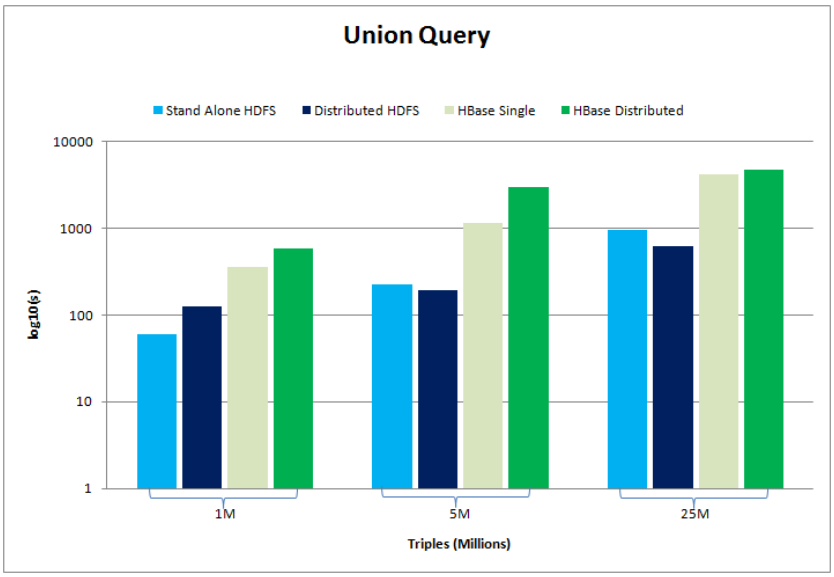

Figure 4.9: Query 8 Response Time

4.6 Comparisons

When it comes to compare our results we have chosen the parameters of RAM memory used and the scalability factor are also considered. According to benchmarks related to RDF stores available at [28]. Some specific results of some stores are given below. By using Virtuoso the RDF data loaded at 160,739 triples per second on a 2 x Xeon5520 machine using 72G RAM. Another similar benchmarking present at [28] of BigOWLIM shows 43,914 triples per second on a 2xXeon5520 2.5 GHz Quad core machine with 64G RAM. AllegroGraph also provides 303K triples per second RDF data load time on 2-4 cores Intel E5520 @ 2.2Ghz with 48G RAM. All the above bench marks show excellent RDF data load times but on the other hand these systems required costly servers with excessive amount of RAM memory installed in them.

For our approach we have used ordinary low cost machines with 2G RAM installed in them to provide RDF data load of 7,687 triples per second which is very cost effective solution.

For more comparisons we have chosen a recent benchmarks performed for different semantic web stores by [21], we have selected three benchmarks including famous jena [4] memory based approach and database approach and SesameRDB [18]. First one process RDF triples in main memory and other two use RDBMS to store and process RDF triples. Test environment used by [21] is as follows, Experiments were performed on a single server machine with 64bit Enterprise edition of Microsoft Windows Server 2003 installed over ACPI Multiprocessor (16 processors) X5550@ 2.67GHz CPU with 8 GB RAM and 8 GB virtual memory configured on 120 GB PERC 6/I SCSI Disk Device.

Figure 4.10 shows bulk load comparisons with HDFS and HBase. Results showed that bulk load outperform database approach of famous RDF processing systems, jena memory model technique shows better results but it is not scalable, as from figure it is clear that it failed to load large data set. Both HDFS and HBase are capable to handle large datasets showing scalability and efficiency at same time.

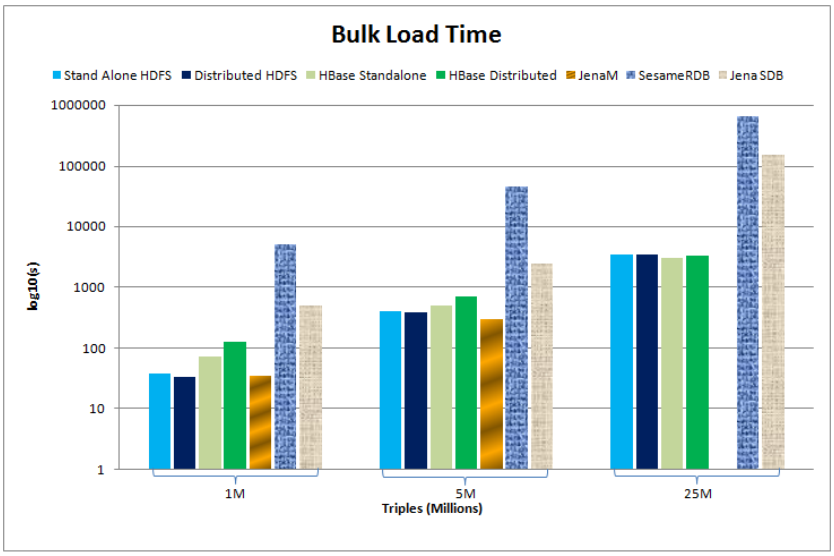

Figure 4.10: Bulk Load Comparisons

Figures 4.11,4.12 and 4.13 describe performance of HDFS and HBase against famous RDF systems for single triple search pattern. Experimental results showed that famous RDF systems show better performance for small

dataset, but while processing large dataset HDFS in distributed environment showing much better response time from all other approaches including JenaSDB and SesameRDB.

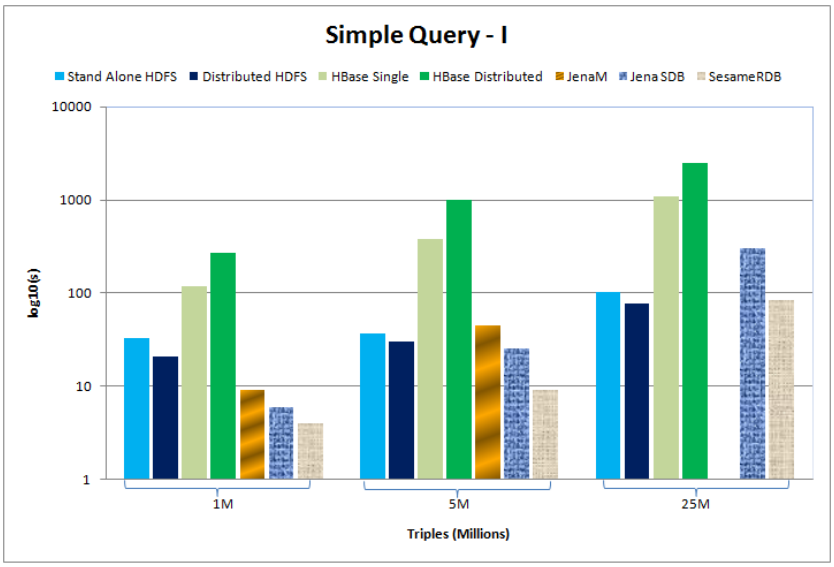

Figure 4.11: Query 1 Comparison

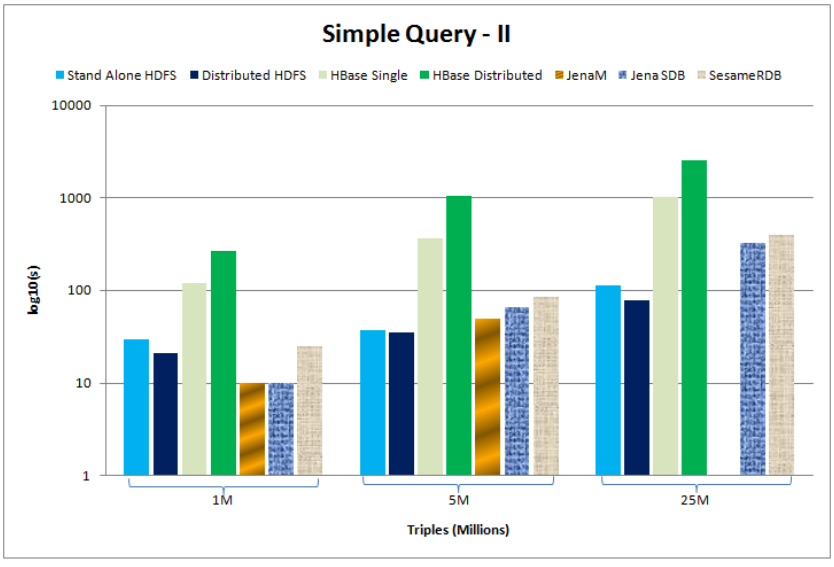

Figure 4.12: Query 2 Comparison

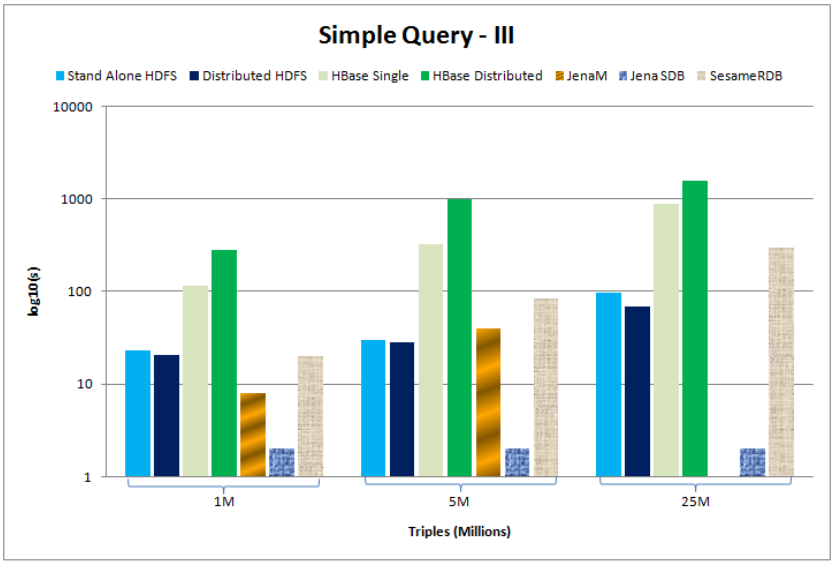

Figure 4.13: Query 3 Comparison

ResultSize-I 4.14, ResultSize-II 4.15 and ResultSize-III 4.16 are the experimental results of queries for large, medium and small output respectively. Figure 4.14 shows that JenaSDB takes much time with respect to all other techniques when result size is large and input data is also large. In other two cases JenaSDB and SesameRDB showing better response then HBase but HDFS takes less time to execute these queries.

Figure 4.17 shows join performance comparisons. Again JenaSDB takes more time with respect to all other techniques, and distributed HDFS performed best large dataset.

For union query all famous RDF stores fail to execute query to generate results for large dataset, at same time even JenaSDB and Jena memory model approach failed to execute over medium size data. Figure 4.18 shows that HDFS and HBase both executed this complex query successfully, and HDFS again showing better response time for medium and large data.

Comparative evaluation show that HDFS and HBase can execute all types of queries involving simple, complex or result size based queries efficiently in an cost effective manner. Detailed analysis of bulkload and query execution experiments reviled strengths or weaknesses of HDFS and HBase. HBase requires performance improvement, by introducing indexes into HBase for RDF data this can be achieved.

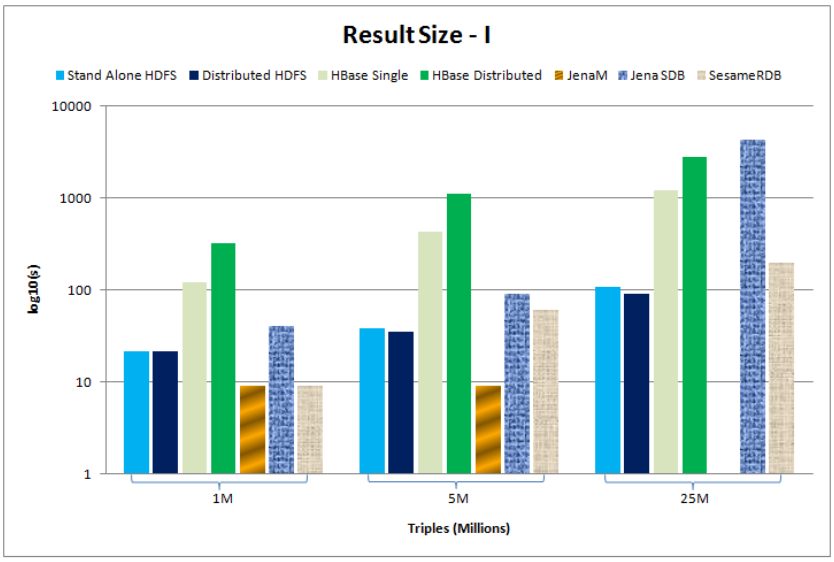

Figure 4.14: Query 4 Comparison

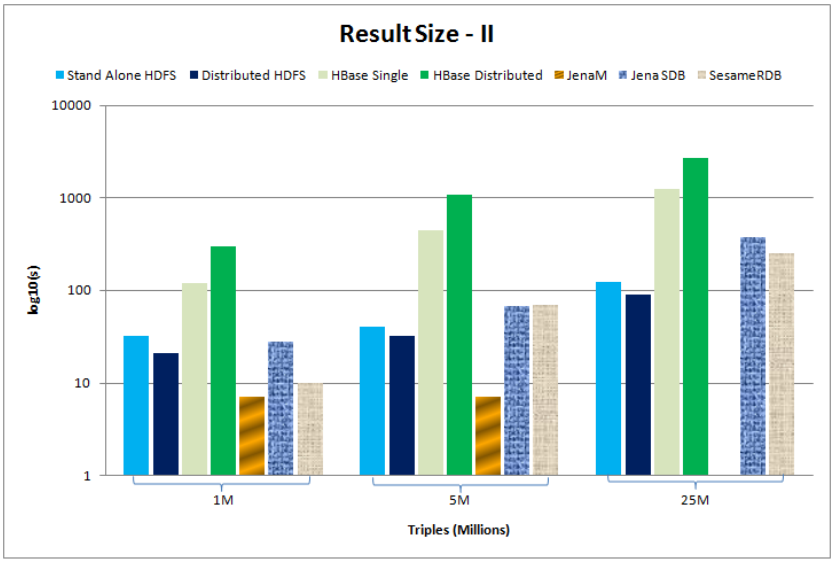

Figure 4.15: Query 5 Comparison

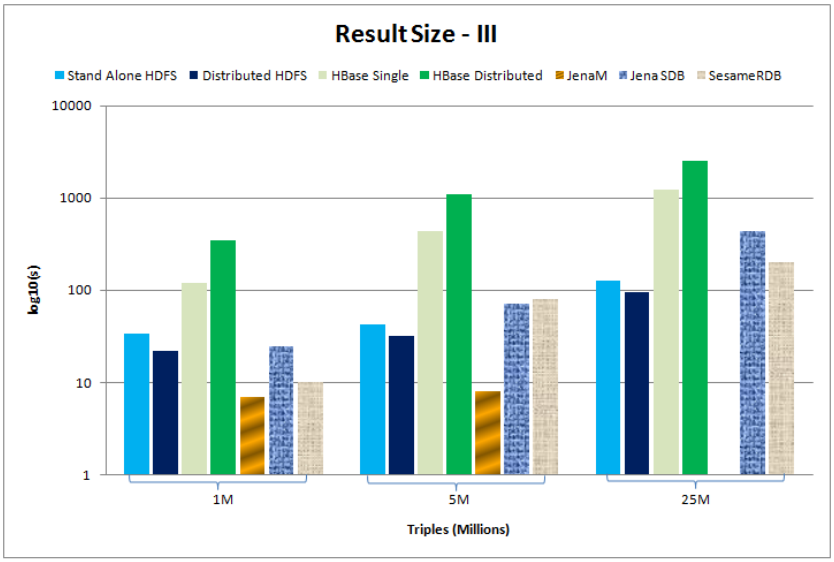

Figure 4.16: Query 6 Comparison

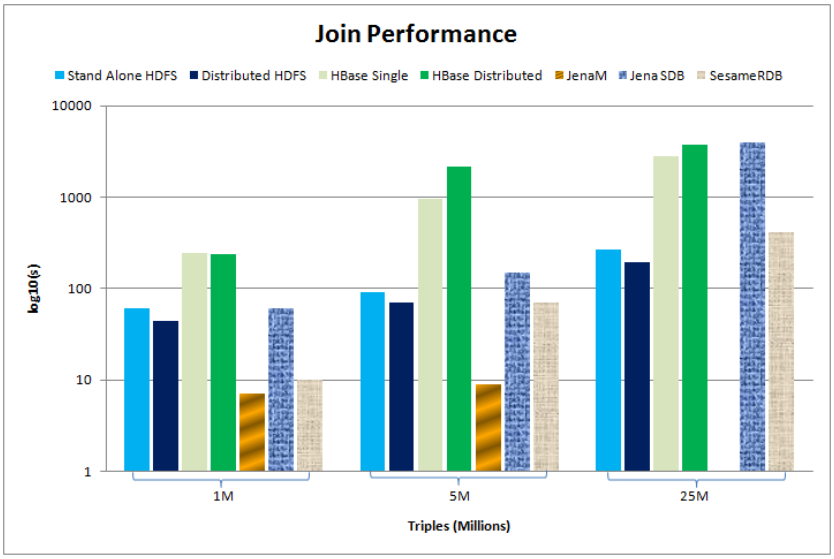

Figure 4.17: Query 7 Comparison

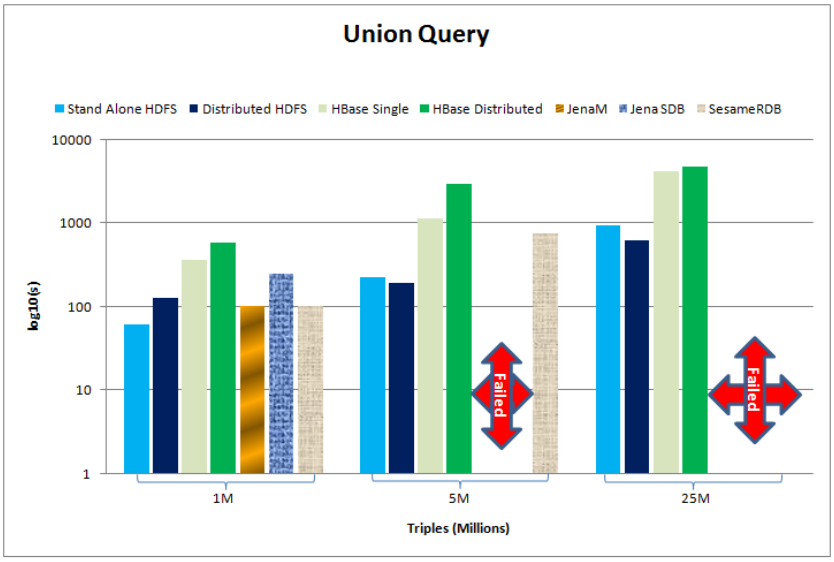

Figure 4.18: Query 8 Comparison

Chapter 5

Conclusion and Future Directions

In previous chapters we have described about RDF, RDF data storage, RDF data retrieval and limitations of currently available system for this purpose. Major problem being faced by the semantic web community is scalability in storing large volume of RDF data and retrieving this data efficiently in a cost effective way. Currently available RDF data stores are either not very efficient and scalable or their prices go so high that they are not affordable. To resolve this issue we have worked on Hadoop framework to store RDF data and retrieve it in an efficient and cost effective manner.

This chapter provides conclusions, summarizes our contributions and does provide a look at the future enhancements and extensions.

5.1 Conclusion

Semantic web is Tim Berner Lee's dream to enable computer to understand meaning behind simple text present over the internet. Semantic web is basically an extension to current web and RDF is W3C standard to represent semantic web data so that semantics can be involved in data to enable computers to understand data over internet. Main object required here is scalable storage and efficient retrieval mechanism for dealing with this huge RDF which will be generated as a result of representing the current data in RDF format. The RDF data is in form of Subject, Property, Object (SPO) triples. To store this huge and ever increasing RDF dataset researchers have started building the RDF stores and some are utilizing already available technology of RDBMS. To retrieve data from RDF data stores a standard querying language aka SPARQL is already in the market, which is also W3C

recommendation to to query semantic web data. SPARQL provides its own statements which provide a good fit for retrieving the RDF data from these stores.

Hadoop is an open source framework for handling voluminous data storage in a distributed environment using HDFS and HBase. It provides HIVE utility to query the data from its distributed structure with the help of MapReduce framework. With scalability of Hadoop and power of HiveQL we provided an RDF data store. In this, we have utilized scalability of Hadoop and ease of HiveQL to provide a cost effective, efficient and scalable RDF data store. With all this we have successfully loaded Barton Laboratory dataset of 25 million triples on an ordinary machines cluster with only 2 GB RAM. Results achieved and discussed in the previous chapter are also encouraging and show clearly that we have found a highly scalable, very efficient and cost effective RDF data store.

5.2 Contributions

5.2.1 Hadoop as Scalable, Efficient and Cost Effective RDF Data Store

As discussed earlier Hadoop is an opensource distributed datastore. We have utilized it to store RDF data of upto 25 million triples with a cluster of ordinary machines each having only 2GB RAM.

5.2.1.1 HDSF As RDF Store

Hadoop Distributed File System (HDFS) facilitates to store vast amount data in distributed environment by ensuring reliability, scalability, efficiency and cost effectively. In this research we showed that HDFS can be used to store and process massive amount of RDF data without losing any of the property described before. We defined the storage format for RDF data in HDFS and the easiest way to explore RDF data by querying it afterwards.

5.2.1.2 HBase As RDF Store

HBase the distributed, column oriented, multidimensional store of Hadoop framework can also be utilized to store RDF data. For HBase we have defined the way to store RDF data. we have defined the way to perform query analysis on HBase tables by using HiveQL.

5.2.2 SPARQL to Hive query conversion

SPARQL is standard language for querying RDF stores whereas Hive component on Hadoop framework provides Hive Query Language (HiveQL) for data retrieval. We found that SPARQL queries can be converted to Hive queries to interact with underlying RDF data in HDFS and HBase. We convert these SPARQL queries to their HiveQL alternatives, which in result produced MapReduce jobs which run in parallel over distributed Hadoop network to achieve efficiency.

5.3 Future Direction

We are planning to build a couple of components utilizing our research work.

5.3.1 SPARQL Querying Interface for Hadoop framework

Hadoop framework is famous for its distributed data storage capabilities. For this it provides two components including HBase and HDFS. Currently we have utilized an extra component of Hive for providing the data retrieval interface. Building a native interface through which the SPARQL queries can be run on Hadoop is one of our future targets.

5.3.2 JENA Plug-in for SPARQL to HiveQL

JENA is one of the very strong APIs available to store and retrieve RDF data. JENA works very well with SPARQL but as described above we have to utilize HiveQL for data retrieval services over Hadoop. Our next mission is to make a plugin for JENA through which user can work on SPARQL queries and the plugin converts all those queries to equivalent Hive queries to retrieve data from the Hadoop framework.

Bibliography

[1] Abadi, D., Marcus, A., Madden, S., and Hollenbach, K. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases* (2007), VLDB Endowment, pp. 411–422.

[2] Ashok Malhotra (IBM), N. S. I. Rdf query specification, December 3, 1998. `http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html`.

[3] Carroll, J. An Introduction to the Semantic Web: Considerations for building multilingual Semantic Web sites and applications. *Multilingual Computing 68*, 19–24.

[4] Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (2004), ACM, pp. 74–83.

[5] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS) 26*, 2 (2008), 1–26.

[6] Consortium., W. W. Semantic web and other technologies, October 2008. `http://www.w3.org/2008/Talks/1009-bratt-W3C-SemTech/Overview.html`.

[7] DBPedia. The dbpedia data set, 2011. `http://wiki.dbpedia.org/Datasets`.

[8] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[9] Foundation., T. A. S. Hadoop, 2011. `http://hadoop.apache.org`.

[10] Foundation., T. A. S. Hadoop hdfs, 2011. http://hadoop.apache.org/hdfs/.

[11] Foundation., T. A. S. Hbase, 2011. http://hbase.apache.org.

[12] Foundation., T. A. S. Hbase, 2011. http://hbase.apache.org.

[13] Foundation., T. A. S. Hbase, 2011. http://hive.apache.org.

[14] Ghemawat, S., Gobioff, H., and Leung, S. The Google file system. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 29–43.

[15] Harris, S., and Shadbolt, N. SPARQL query processing with conventional relational database systems. In *Web Information Systems Engineering–WISE 2005 Workshops* (2005), Springer, pp. 235–244.

[16] Laboratory, K. D. Dataset: Dblp, 2008. http://kdl.cs.umass.edu/data/dblp/dblp-info.html/.

[17] Lee, T., Hendler, J., Lassila, O., et al. The semantic web. *Scientific American 284*, 5 (2001), 34–43.

[18] openRDF. Sesame, 2011. http://www.openrdf.org/.

[19] Owens, A. An Investigation into Improving RDF Store Performance.

[20] Reynolds, D., Thompson, C., Mukerji, J., and Coleman, D. An assessment of RDF/OWL modelling. *Bristol, UK: Hewlett Packard. Retrieved 12* (2009), 2005–189.

[21] Sahar, A.): Analysis of Semantic Web Databases, 2010.

[22] Sini, M., Salokhe, G., Pardy, C., Albert, J., Keizer, J., and Katz, S. Ontology-based Navigation of Bibliographic Metadata.

[23] TECHNOLOGY, M. I. O. Dataset, 2007. http://simile.mit.edu/wiki/Dataset:_Barton.

[24] TECHNOLOGY, M. I. O. Simile, 2011. http://simile.mit.edu/.

[25] W3C. Sparql query language for rdf, 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/.

[26] W3C. Rdf query survey, 2001. http://www.w3.org/2001/11/13-RDF-Query-Rules/.

[27] W3C. Rdf query survey, 2010. http://en.wikipedia.org/wiki/RDF_query_language.

[28] W3C. Largetriplestores, 2011. http://www.w3.org/wiki/LargeTripleStores/.