LARGE SCALE RDF STORAGE SOLUTIONS EVALUATION

Bela Stantic, Juergen Bock
Institute for Integrating and Intelligent Systems, Griffith University, Brisbane, Australia

Irina Astrova
Institute of Cybernetics, Tallinn University of Technology, Tallin, Estonia

Keywords: Resource Description Framework - RDF, RDF storage, Sematic Web.

Abstract: The increasing popularity of the Semantic Web and Semantic Technologies require sophisticated ways to store huge amounts of semantic data. RDF together with the rule base RDF Schema have proved themselves as good candidates for storing semantic data due to the simplicity and high abstraction level. A number of large scale RDF data storage solutions have been proposed. Several typical representative have been discussed and compared in this work, namely Sesame, Kowari, YARS, Redland and Oracle's `RDF_MATCH` table function. We present a comparison of those approaches with respect to consideration of context information, supported access protocols, query languages, indexing methods, RDF Schema awareness, and implementation. We also identify applicability as well as discuss advantages and disadvantages of particular approach. Furthermore, an overview of storage requirements and performance tests has been presented. A summary of performance analysis and recommendations are given and discussed.

1 INTRODUCTION

Originally designed for storing meta-data on the web, the *Resource Description Framework* (RDF) became increasingly popular for storing all different sorts of data. This is due to the simple but expressive triple structure, in which RDF data is organised. Typical RDF documents scale to millions of triples in RDF representations of data, such present in WordNet (WordNet, 2007), UniProt (UniProt, 2007), or Wikipedia3 (Wikipedia3, 2007).

Accessing large scale RDF data requires highly efficient storage and query management. There have been a number of solutions presented in the literature for storing RDF data. While the majority of presented methods have performance evaluation conducted in isolation, not enough attention was directed to compare different methods. In this work we evaluate the open-source solutions Sesame (Jeen Broekstra and Arjohn Kampman and Frank van Harmelen, 2002), YARS (Andreas Harth and Stefan Decker, 2005), Kowari (David Wood and Paul Gearon and Tom Adams, 2005), Redland (Dave J. Beckett, 2001) as well as the commercial Oracle Spatial 10g Release 2 table function (Eugene Inseok Chong and Souripriya Das and George Eadon and Jagannathan Srinivasan, 2005) for RDF storage. In order to identify strengths and shortcomings of particular method and future research direction we compare methods with respect to low-cost solutions, space usage, repository independence and efficiency.

The remainder of this paper is organised as follows. Section 2 recalls some basic introduction of RDF and RDF Schema. In section 3 several typical representative of RDF storage systems are presented. In section 4 we compare different approaches while in section 5 we present experimental results and analysis. Finally, in section 6, we present our conclusions and discuss possible future work.

2 BACKGROUND

The *Resource Description Framework* (RDF) has been a key concept in Semantic Web technologies and a W3C Recommendation since February 2004 (Frank Manola and Eric Miller, 2004). It is based on XML (Sperberg-McQueen et al., 2004) and designed

Stantic B., Bock J. and Astrova I. (2007).
LARGE SCALE RDF STORAGE SOLUTIONS EVALUATION.
In *Proceedings of the Second International Conference on Software and Data Technologies - Volume ISDM/WsEHST/DC*, pages 103-108
DOI: 10.5220/0001329001030108

to represent resources on the web in a triple structure. A triple consists of subject, predicate and object, where subject and predicate must be Uniform Resource Identifiers (URIs), and objects can be URIs or values (RDF literals). This structure allows graph representation of resources, if the object is used as another subject.

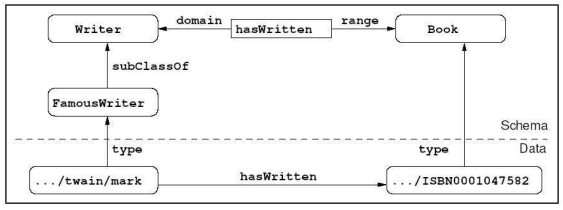

Figure 1: RDF Schema.

RDF does not define reserved terms or a vocabulary for the information represented. RDF Schema overcomes these shortcomings by introducing classes with sub-class relations, properties with domain and range restrictions, amongst others. An RDF sample with underlying Schema can be seen in Figure 1.

3 RDF STORAGE SOLUTIONS

Significant work has been directed towards solving the problem of storing RDF data. In this section we present some of the prominent solutions that have been proposed in the literature.

3.1 Sesame

Sesame has been introduced in (Jeen Broekstra and Arjohn Kampman and Frank van Harmelen, 2002). It represents a repository-independent framework for storing and querying RDF and RDF Schema data. According to the authors Sesame is the "first publicly available implementation of a query language that is aware of the RDFS semantics". This refers in particular to the RQL (Gregory Karvounarakis et. al., 2002) support, which has been designed for interpreting RDF descriptions with respect to RDF Schemata. Sesame exists in several variations, representing the storage paradigms *memory*, *database*, or *native*. Sesame Memory manages data only in a machine's main memory, where Sesame Database uses any underlying DBMS to store RDF data persistently. Sesame Native does not use a database, but its own proprietary storage system, explicitly tailored to RDF.

The Sesame framework is designed to be independent of the underlying database. It encapsulates functionality to process RDF data manipulation and querying requests, and handles communication with multiple clients via HTTP, RMI or SOAP protocols. The actual repository interface is implemented as database-specific in a so-called SAIL (**S**torage **A**nd **I**nference **L**ayer).

Sesame incorporates three different modules for dealing with RDF data. The *Query Module*, which handles queries in the supported query languages, the *Admin Module* to manipulate data, such as insertion or deletion, and the *Export Module*, which renders the data as proper XML/RDF files. These modules are invoked by the *Request Handler*, which itself communicates with the *Protocol Handlers* for client interaction.

3.2 Kowari

Kowari Metastore was introduced to handle large-scale storage of RDF and OWL (David Wood and Paul Gearon and Tom Adams, 2005). Instead of the original RDF triple structure, the system provides a native quad-store, where the fourth element in addition to the basic RDF triples represents a meta-element indicating which RDF model the triple belongs to. Kowari stores RDF data in two tables, called "Node Pool" and "String Pool", where the former contains IDs only, and the latter according mapping to URIs, RDF Literals and XML Datatypes.

Query access to RDF data in the Kowari Metastore is provided via a number of query languages. Apart from Kowari's native query language iTQL, other access methods, such as SOFA (**S**imple **O**ntology **F**ramework **A**PI) (Alishevskikh, A., 2007), Java RDF, Jena (Jena, 2007), RDQL (Andy Seaborne, 2004) and SOAP are supported.

Kowari is designed to store hundreds of millions of RDF triples while providing reasonably short response time for queries, due to the the hybridisation of indexing structure, more precisely, AVL trees and B^*-trees. The main advantage of AVL trees is, though they are binary and hence deeper, they contain less overhead and therefore fit more likely in the memory of adequate machines.

3.3 YARS

The main contribution of Harth and Decker introduced an index structure tailored to RDF data (Andreas Harth and Stefan Decker, 2005). They further proposed the RDF native storage framework YARS (**Y**et **A**nother **R**DF **S**tore) which implements this index structure.

Storage in the YARS system is done similarly to Kowari, where full string representation of RDF graphs is provided by a *lexicon*, where the actual quads are stored as IDs in a quad table. For indexing a B^+-Trees are used, adopted from JDBM (JDBM, 2007). The indexes cover all combinations of possible access patterns to the quad store, and are combined to reduce the actual number of indexes.

3.4 Redland

The Redland Framework was designed to be accessible from a number of applications with different interfaces (Dave J. Beckett, 2001). It provides API access for various languages, such as *Perl*, *Python*, or *C*. Redland itself is implemented in *C*. The latest version does not support access via HTTP, however, the intention is, that any web application could be developed for its particular need and use Redland as an RDF storage backend by API access.

The Redland RDF storage framework supports RDQL (Andy Seaborne, 2004) and SPARQL (Eric Prud'hommeaux and Andy Seaborne, 2006) query languages. Internal storage in Redland is indexed by three Hash indexes. Although Redland translates RDFS vocabulary in proper RDF triples it does not support any reasoning with RDFS rules.

3.5 RDF Support in Oracle

Oracle Spatial 10g provides sophisticated means to store and query RDF data. The main contribution is the newly introduced `RDF_MATCH` table function (Eugene Inseok Chong and Souripriya Das and George Eadon and Jagannathan Srinivasan, 2005) with the following signature:

```
RDF_MATCH (
        Pattern    Varchar,
        Models     RDFModels,
        RuleBases  RDFRules,
        Aliases    RDFAliases
          )
        RETURNS AnyDataSet;
```

`Pattern` is an RDF Graph pattern in a SPARQL-like syntax (Eric Prud'hommeaux and Andy Seaborne, 2006), `Models` is a list of RDF Models, where an RDF Model represents a "view" to a triple table according to some user privileges. `RuleBases` is an optional argument stating some user defined rule bases. Note that RDF Schema is implicitly available as a rule base. Lastly, `Aliases` optionally include user defined namespace aliases. However, the `RDF` namespace is implicitly available. The return value of `RDF_MATCH` is of type `AnyDataSet`, since the number and types of the columns in the return table vary according to the graph pattern.

Similarly as for Kowari and Yars RDF data are stored in two relational database tables. This general approach has the advantage of storage efficiency for repeatedly occurring URIs. Furthermore, simple IDs can be indexed easier than bulky URIs. Oracle ensures high efficiency in querying RDF data, by changing its table function infrastructure to allow complete translation of the `RDF_MATCH` function into an SQL query prior to execution.

4 COMPARISON

In this section we identify main factors that influence quality and efficiency of any approach and discuss advantages and disadvantages of particular approach.

Context Information: Context information is important, when RDF data from different sources are involved or user privileges have to be considered. Kowari and YARS operate on quads, which is an augmentation of the original RDF triple structure. The additional component refers to this context. Oracle provides RDF models to accommodate the idea of contexts, and Redland provides an option, to enable context information on demand. In contrast context information is not supported in Sesame approach.

Query Languages: YARS only supports methods to formulate queries by two extensions to the N3 notation. Kowari introduced the iTQL query language, which accommodates features for Kowari's OWL support, as well as traditional RDF query features. Kowari further implements subsets of RDQL and SPARQL, which are also supported by Redland. Oracle provides seamless integration of its table function in SQL, where the table function uses SPARQL-like syntax to specify graph patterns. Sesame currently supports RQL, RDQL, and Sesame's SeRQL query language.

Access: The Kowari introduction paper (David Wood and Paul Gearon and Tom Adams, 2005) somehow does not distinguish between access protocols and query languages. They are treated equivalently in so-called Access APIs. These access methods include SOAP, SOFA (**S**imple **O**ntology **F**ramework **A**PI), JRDF (**J**ava **RDF**, an API to access RDF data from within Java™ applications), Jena, a Java Server Pages tag library and RDFS JavaBean. YARS supports only HTTP as an access method, while Sesame provides HTTP, SOAP and Java RMI. Oracle's table function

can be used by any means provided to access the Oracle database, like the Oracle Application Server, OBDC, etc. The Redland framework is not designed as a standalone solution, which accepts common access methods. It rather provides API interfaces for use from within other applications.

Indexing method: Oracle implements two three-column B^*-Tree indexes on the triple table, based on typical query patterns. The first one is *Predicate, Subject, Object*, the second one is *Predicate, Object, Subject*. While Sesame DB uses the underlying database in an abstract manner, all storing and indexing issues are handled by this database. However, Sesame's native version uses separate indexes on subject, predicate, and object respectively. Kowari uses a combination of AVL trees and B^*-Trees to incorporate advantages of both approaches. Frequently used subsets of graph patterns are indexed directly. Redland uses three Hash indexes, each mapping two triple elements to the third one. YARS implements B^+-Tree indexes for all combinations of quad patterns.

Implementation: Sesame, Kowari, and YARS are all implemented in Java™, where Redland is implemented in *C*. The Oracle table function, however, is seamlessly integrated in the table function interface of the Oracle Kernel.

RDFS Semantics: YARS does not support RDFS semantics or other rule bases for reasoning on the RDF data. Oracle implicitly supports RDF Schema rules, but is able to load user defined rule bases in addition. Kowari further supports OWL constructs. The Redland framework supports RDF Schema only in terms of translation of specific vocabulary into pure RDF, and does not support any RDFS reasoning.

In Table 1 we show an overview of the features discussed in this section for all RDF large scale storage approaches considered in this study.

5 PERFORMANCE EVALUATION

For testing RDF representation of UniProt data (UniProt, 2007) are used, which scales up to 80 million triples. Oracle's `RDF_MATCH` function (Eugene Inseok Chong and Souripriya Das and George Eadon and Jagannathan Srinivasan, 2005) has been tested and demonstrated reasonable performance on large-scale RDF data. The tests showed that the query performance is highly scalable, since query runtime does not change significantly for scaling the data size from 10 to 80 million triples. In fact, the longest runtime in these tests took a query, matching 6 triples with 5 variables and results limited to 15,000 rows, which returned the answer in about one second.

Testing of YARS, Sesame and Redland was performed on 4 different queries representing typical query patterns. The results show, that YARS generally outperforms the other approaches, except for queries when simple Hash lookup is possible. In this case, Redland has better performance, due to its Hash indexing method. That means, that Redland is optimised for getting the sources, getting the targets, or getting the arcs in an RDF graph, provided all other information. Note that only triples are considered in this test, although YARS and Kowari operate on a quad structure. This, however, only adds context information and can be regarded as constant for these test purposes.

Because Kowari could not be implemented, performance analysis can only be discussed in isolation based on the authors' testing results (David Wood and Paul Gearon and Tom Adams, 2005). The test consisted of building data and index structures for up to 235 million triples, as well as simultaneous requests by 250 simulated clients. The authors state that Kowari compares with MySQL using a simplistic schema. Apart from that, no comparison to competing systems, and no qualitative query performance results are given.

5.1 Analysis

Large scale testing has been carried out using the UniProt (UniProt, 2007) data of about 80 million triples. Oracle's `RDF_MATCH` function, YARS, Redland, Sesame MySQL, and Sesame Native were compared. In Table 2 we present index space requirements for different approaches to store and manage large scale RDF data. A reason for those differences is that Sesame Native uses 4 byte IDs, where Kowari and YARS uses 8 byte IDs. Redland does not use IDs at all, and operates directly on URIs, which results in a relatively large index.

All discussed approaches for large scale RDF storage but Redland use variations of the *B*-Tree as an index structure. Redland uses three Hashes for fast lookup of graph patterns with only one variable in the triple. In the comparison Redland performed best in a query, where only the subject was requested for constant predicate and object, which Redland carries out as a simple Hash lookup. Since Redland only provides three Hash indexes, all other queries involving more complex graph patterns need cumbersome joins and combinations make Redland's performance

Table 1: RDF storage and querying features of Oracle's RDF_MATCH table function, Sesame, Kowari, YARS, and Redland.

	Oracle	**Sesame**	**Kowari**	**YARS**	**Redland**
Context	yes	no	yes	yes	yes
Access	Application Server, ODBC, etc.	HTTP SOAP RMI	SOAP APIs	HTTP	APIs
Query Lang.	SQL (graph patterns SPARQL-like)	RQL RDQL SeRQL	iTQL RDQL SPARQL	N3	RDQL SPARQL
Index	*B*-Tree (2 indexes)	DBMS specific	hybrid (AVL, B^*-Tree) (6 indexes)	B^+-Tree (6 indexes)	Hash (3 indexes)
Impl.	Oracle-Kernel	Java™	Java™	Java™	C
Rules	RDFS user defined	RDFS	RDFS OWL	–	–

Table 2: Index size space requirements for 80 millions triples.

RDF Storage Method	Size (MB)
YARS	2,857
Sesame MySQL	9,068
Sesame Native	1,095
Redland	57,141
Oracle	4,915

worst. Sesame Native maintains three indexes on each, subject, predicate, and object. Query patterns, specifying a single triple element perform quite well. For queries, that specify more than one triple element, a join must be performed, which results in poor runtime results for those kinds of queries. Since YARS keeps all possible combinations indexed, it performs well for all kinds of query patterns. However, it also runs slightly faster in those test cases, where proper indexes *are* available for Sesame Native. Sesame with underlying DBMS relies heavily on the internal storage and optimisation mechanisms of those databases. However, a comparison of Sesame using the Oracle SAIL and Oracle itself is not appropriate, since Sesame uses the Oracle RDF_MATCH table function, and just adds additional overhead. Therefore, Sesame DB can be seen as the preferred RDF storage system, when database independence is an important aspect.

Comparing the actual response time, Oracle was able to deliver 15,000 rows out of 80 million triples for a moderately complex graph pattern of 6 triples and 5 variables, in roughly one second. In contrast to that, the YARS testing results show a runtime of about 10 seconds. Redland's fastest result for a simple query with only one triple and one variable in the query pattern, returned about 160,000 rows in about 10 seconds. Oracle's table function returned same query result in almost same time like Redland, however with respect to the more complex query patterns, Oracle seems likely to outperform all other approaches.

The choice of a particular RDF storage solution may heavily depend on factors, such as cost (commercial vs. open-source), access methods, supported query language, OWL support, context support, portability (platform and database), or general performance. (Refer to table 1 for details.) While Oracle seems to perform best in a number of aspects, it is only commercially available. However, YARS is a reasonable alternative with respect to performance, but currently does not support popular query languages.

If (future) OWL support is important, Kowari may be a reasonable consideration. Redland performs well for queries with limited query patterns (and where proper adjustments are made due to specific applications). The biggest advantage of Sesame is that it is database independent. However, using Sesame with the Oracle SAIL imposes additional overhead, and one should consider using Oracle directly for RDF storage.

6 CONCLUSION AND FUTURE WORK

In order to meet the growing requirements of large scale semantic data storage, which are often represented in RDF and RDF Schema, a number of approaches have been developed. Several of these RDF storage solutions have been presented, namely Oracle's `RDF_MATCH` table function, Sesame, Kowari, YARS, and Redland. Although they all strive for the same goal, there are some differences in terms of database storage, supported query languages, access protocols, indexing methods, implementation, and reasoning support. In this work we present a comparison of those approaches with respect to supported access protocols, supported query languages, indexing methods, RDF Schema awareness, and implementation and we identify applicability of particular approach as well as advantages and disadvantages. Furthermore, an overview of storage requirements and performance tests has been presented. Depending on the feature comparison presented in Table 1, a certain storage system may be chosen according to use-cases, application, or knowledge bases.

We intend to further compare large scale RDF storage solutions on more complex queries and large scale RDF data in order to reveal strengths and weaknesses of particular approach. This will help us to identify the applicability of particular approach and the need for distinct research directions to rectify shortcomings.

REFERENCES

Alishevskikh, A. (2007). SOFA Simple Ontology Framework API. `https://sofa.dev.java.net/`.

Andreas Harth and Stefan Decker (2005). Optimized Index Structures for Querying RDF from the Web. In *LA-WEB*, pages 71–80.

Andy Seaborne (2004). RDQL - A Query Language for RDF. W3C Member Submission, W3C. `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`.

Dave J. Beckett (2001). The Design and Implementation of the Redland RDF Application Framework. In *WWW*, pages 449–456.

David Wood and Paul Gearon and Tom Adams (2005). Kowari: A Platform for Semantic Web Storage and Analysis. In *Proceedings of xtech*.

Eric Prud'hommeaux and Andy Seaborne (2006). SPARQL query language for RDF. W3C working draft, W3C. `http://www.w3.org/TR/2006/WD-rdf-sparql-query-20061004/`.

Eugene Inseok Chong and Souripriya Das and George Eadon and Jagannathan Srinivasan (2005). An Efficient SQL-based RDF Querying Scheme. In *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*, pages 1216–1227, Trondheim, Norway. VLDB Endowment.

Frank Manola and Eric Miller (2004). RDF Primer. W3C Recommendation, W3C. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

Gregory Karvounarakis et. al. (2002). RQL: A Declarative Query Language for RDF. In *WWW*, pages 592–603.

JDBM (2007). `http://jdbm.sourceforge.net`.

Jeen Broekstra and Arjohn Kampman and Frank van Harmelen (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *International Semantic Web Conference*, pages 54–68.

Jena (2007). `http://www.hpl.hp.com/semweb/jena.htm`.

Sperberg-McQueen, C. M., Maler, E., Yergeau, F., Paoli, J., and Bray, T. (2004). Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, W3C. `http://www.w3.org/TR/2004/REC-xml-20040204`.

UniProt (2007). `http://www.isb-sib.ch/~ejain/rdf/`.

Wikipedia3 (2007). `http://labs.systemone.at/wikipedia3`.

WordNet (2007). `http://www.semanticweb.org/library/`.