Volume 2, No. 1, Jan-Feb 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Storage Schemes and Query Optimization Techniques for RDF Data

Chuttur Y Mohammad
Indiana University
Bloominton, Indiana, USA
mchuttur@indiana.edu

Abstract: With the rising interest in the semantic web, and the rapid proliferation of semantic data, there is an increasing need for efficient means of storing and querying data stored using the RDF data model. In this paper, the motivations for using RDF as a data model in relation to the semantic web are presented, following which, several popular available storage techniques for RDF data are described. Different proposed methods for optimizing queries on RDF datasets are also covered to illustrate the most recent state of the art in RDF data storage and query optimization techniques.

Keywords: semantic web, rdf data model, databases, query optimization.

I. INTRODUCTION

The Resource Description Framework (RDF) is a language to represent information about resources on the Web [1]. Originally designed as a metadata model, that represented metadata about web resources such as the title, author, copyright and licensing information, RDF has evolved into a more expansive concept with generalization of the concept of "web resources". It can be used for identification of resources on the web rather than just for retrieval purpose [4]. This equips RDF to represent information that can be processed by applications and not just be displayed on search. By providing a common framework, it thus provides for exchange of web resource descriptions between computer applications without the loss of meaning. That is why RDF is a key to the implementation of the 'semantic web', which according to the world wide web consortium (W3C) is the next evolutionary stage of internet activity enhancement where automated programs can store, exchange and make use of machine-readable information located throughout the web, making information handling activities on the web more efficient and certain [4].

RDF was first published as a data model with XML syntax as a W3C Recommendation in 1999 [7] and the newer, improved version of RDF was later published in 2004.Since then, there has been a growing interest in exploiting the benefits proposed by the RDF model. This paper adds to the growing literature of RDF by setting the roadmap on the methods used to store RDF data as well as different query optimizing techniques that have been proposed so far. First a review of the RDF data model is provided. Then, several use cases for RDF are described followed by a review of some of the popular models that have been proposed to store RDF data. Different techniques for optimizing queries on these storage models are then discussed, following which, a summary of the current state of the art in storage model and query optimization techniques is provided.

II. RDF DATA MODEL IN BRIEF

The RDF data model draws form the concept of Relational Database Management System (RDBMS). In fact, it makes use of a conceptual modeling approach similar to the Entity-Relationship Model as proposed over thirty years ago [5]. RDF statements consists of the subject-predicate-object (SPO) format of expressions to describe Web resources (see Figure 1). These statements are called 'triples' in RDF terminology. The RDF statement has either a Uniform Resource Identifier (URI) or a blank node as its subject. The predicate is a URI implying a relationship between the subject and an object, while the object itself can be a URI, blank node or Unicode string lateral [5]. Using this form of expression, i.e, subject-predicate-object statements, resources on the web can be meaningfully represented. Consider for example an expression having for subject the value: http://www.google.com; predicate, the value: "is-a", and for object, the value: "search-engine". It is easy for both a human and a system to recognize that http://www.google.com is actually a search engine.

Figure 1. RDF Data Model

RDF data can be serialized in several formats. The most common serialization format is by using XML syntax to write and exchange RDF graphs, referred to as RDF/XML implementations, but other serialization formats also exist and these include the Notation 3 or N3 format, which is a non-XML implementation claimed to be easier to follow as it is based on tabular notation making the triples easier to recognize [3]. N3 is also similar to Turtle and N-Triples formats.

RDF also supports several query languages and these include the SPARQL Protocol and RDF Query Language (SPARQL), as well as other query languages like RDQL, Versa, RQL and XUL [6]. SPARQL was published as a W3C recommendation in January 2008, while the other languages appeared much earlier.

III. SOME RDF USE CASES

Since its introduction as a data model, RDF has been increasingly used in various applications. Some popular applications using the RDF model include Really Simple Syndication (RSS) – a collection of web feed formats used to periodically publish updated works, Friend of a Friend (FOAF) - designed to describe people with interests and interconnections, Haystack client – a semantic web browser developed by MIT, and MusicBrainz, which publishes information about Music albums. Since RDF is very suitable

for sharing vocabularies, it is also highly in use for life science data, for which there is a high need for data integration over the web.

Other application using RDF include Chandler, which is a personal information management (PIM) application, built as an open source technology that includes RDF and RDF/XML specifications. RDF Gateway is another application which makes uses of RDF and operates as an integrated web server and database built from the ground up rather than on top of an existing web server (such as Apache) or database (such as SQL Server or MySQL). Siderean Software's Seamark also uses RDF and it is a versatile application which provides resources for intelligent site querying and navigation. Similarly Plugged In Software's Tucana Knowledge Store (TKS) uses RDF so as to enable storage and retrieval of data that can be easily scaled to larger datastores.

Adobe, a major player in the publications and graphics industry, has also incorporated RDF in its application. Its RDF/XML strategy is known as XMP. XMP focuses on providing a metadata label that can be embedded directly into applications, files, and databases, including binary data.

In other words, these use cases significantly demonstrate the range of applications that RDF data can be applied into and the need for efficient storage models.

IV. POPULAR RDF STORAGE SCHEMES

Since the proposal of the RDF data model, various storage schemes have been proposed for storing RDF data. They can be classified into three main categories: File system, RDBMS and OODBMS. Other methods of storage also exist under these main categories as shown in Figure 2. Further information on each storage scheme follows.

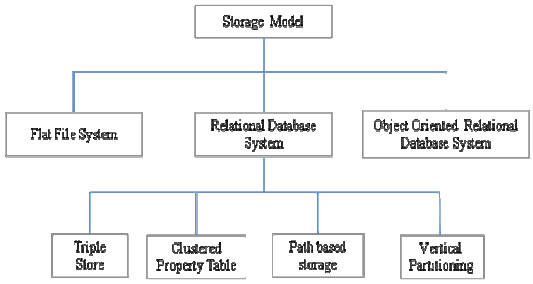

Figure 2. Different RDF Storage Schemes.

A. *Simple Flat File Systems*

Mostly used on the web for storing metadata information, RDF data is stored as RDF/XML notations in web pages or HTML files. This type of storage model typically requires an XML parser that can extract the different elements from the XML file in order to obtain information regarding RDF triples.

B. *RDBMS*

Since RDF data consists of triples in the form of a subject, a predicate and an associated object, many storage schemes exploit this property of the RDF data model to store RDF data in RDBMS as described below.

1) Triple Store: This is a simple three column table in RDBMS with attributes set as subject, property and object respectively [16]. Due to its scalability, this type of storage is very efficient for storing large amount of data. Another interesting feature of storing data as triples is that it just requires self joins in order to answer queries, since all data is stored in a single table in the former.

2) Clustered Property Table: Clustered property table is one type of property table. This technique de-normalizes RDF tables by physically storing them in a wider, flattened representation which is more similar to traditional relational schemas [18]. One way to do this flattening is by finding sets of properties that tend to be defined together; i.e., clusters of subjects that tend to have these properties defined. This flattened property table representation usually requires less number of joins during access as it eliminates self-joins on the subject column.

3) Property Class Table: Property class table is another type of property table. This type of table exploits the type property of subjects to cluster similar sets of subjects together in the same table. Unlike the Clustered Property Table method of storage, in the Property Class Table, a property may exist in multiple property-class tables. In Jena2 [22], for example, property-class tables were found to be particularly useful in storing reified statements. Oracle also adopted a property table-like data structure which they called "subject-property matrix" to increase the speed of queries over RDF triples. The most important advantage of the property table approach over the triple-store is that they reduce subject-subject self-joins of the triples table.

4) Vertical Partitioning: Vertical partitioning [18] can be defined as creating two column tables based on unique properties, with one column representing subjects and the second column referring to objects. The following are the advantages of this approach over the Property Class table approach:

a) Support for multi-valued attributes: If a subject has more than one object value for a particular property, then each distinct value is listed in a successive row in the table for that particular property.

b) Support for heterogeneous records: Subjects that do not define a particular property are simply omitted from the table for that property. This avoids the explicit storage of NULL data, which is very critical when the data is not well-structured.

c) Fewer unions and fast joins: Property table approach reduces the need for union clauses in the queries since whole data is present in the same table. On the other hand the vertical partitioning approach requires more joins relative to the property table approach. But still the properties are joined using simple, fast (linear) merge join algorithms' which make vertical partitioning approach more preferable than the property table approach.

5) Path Based Storage: Proposed by Matano et al. [15], the path based relational RDF storage scheme parses RDF data and generates its own RDF graph. This RDF graph is decomposed into sub graphs which are then stored into distinct relational tables. The decomposition into sub graphs is based on the type of predicate, while the sub graphs are based on Class Inheritance, Property Inheritance, Type information, and Domain-range. Any remaining data, then, fall into Generic graphs. Based on the sub-graphs Matano et al. designed their relational schema which includes the relations class, property, resource, triple, path and type. Since this storage scheme

retains schema information and path expression for each data source it is highly efficient in processing path based queries.

C. OODBMS

RDF data can also be stored as entire graphs in an object oriented [23] or semi-structured database. The graph model consists of nodes and edges to represent RDF statements and it allows direct interpretation of semantics instead of building graph from triples as may be required by certain query language such as RQL. Furthermore, storing RDF data as graph offers two more advantages over other existing schemes since:

a) Storage design can be simplified as the graph can be stored directly without further reorganization, and

b) The RDF graph can be interpreted directly as it is already in the storage and no external mapping is required from triples to the graph model.

V. QUERYING RDF DATA

Several langauges exist for querying RDF data. Examples include RDQL, Versa, RQL and XUL but the most recent language proposed is SPARQL [6]. A full discussion of each query language is beyond the scope of this paper, and instead attention is given on the methods used to optimize queries based on the SPARQL language. In query optimization, the aim is to find the optimal plan for query execution. Similar to query optimization in relational databases, several ways for optimizing SPARQL queries exist as discussed below.

A. Use of Indices:

Use of indices have been successfully proposed and implemented in for relational databases. To achieve similar performance, several indices have been proposed based on the way RDF data is stored. For instance, Chong et al. [19] proposed the use of B-tree index when the data is stored in two tables, one of which stores all the triples, and the other table stores only the corresponding URIs. In Chong et al.'s proposal, the B-tree index is built on either the subject or the predicate. Similarly, Alexaki. et al [20] used B-tree indices when four tables (class, sub-class, property and subproperty) are used to store RDF data. In this case, B-tree indices are built on subject, predicate and object. Neumann et al. [18] on the other hand used B+ tree index on a single triple table, by considering all six combinations of triples as well as any frequently used projections, making the index compressed that reduces the amount of space utilization. This method works well when RDF data is stored in Triple table. Furthermore, Baolin et al. [21] combined three indexes to propose the mixed index structure consisting of a B+ tree, Path index, and Context index, which was found to work well for both RDF graphs as well as RDF tables, and which increased the performance of long path queries. Octavian et al. [17] further proposed the GRIN index, which is a balanced tree data structure optimized for RDF graph storage model. This was found to increase the performance of graph based queries. *Furthermore,* George et al. [13] proposed the Triple-T index, which consists of B+tree index built on all combinations of triples. This was found to be very efficient for RDF data stored in one single triple table.

B. By Selectivity Estimation:

Using selectivity estimation, SPARQL queries are optimized based on selectivity of basic graph patterns (BGP) [14]. Consider the case where the data is stored in memory. Join order optimization (i.e. static query optimization) can be achieved by knowing the selectivity of the underlying triple patterns. For example, for two triple patterns as follows: "?x", "rdf:type", "uv:person"; "?x", "uv:hasSSN", "324-09-7865", one should be able to easily state that the second triple pattern should be executed first because its result set is considerably small when compared to the first triple pattern. Thus, by reordering the queries, query performance can be significantly improved.

C. Conversion of SPARQL Queries to relational algebra:

As proposed by Cyganiac et al. [12], SPARQL queries can be converted to relational algebra onto which optimizing techniques can be applied. It is well known, within the database community, that relational algebra is an intermediate language for the expression and analysis of the queries that is widely used in the DBMS. A query represented in relational algebra helps the query engine to perform better. Similarly, considering RDF data stored in tables, the SPARQL query can be represented in relational algebra. Different operators like selection, projection and rename, inner-join and left outer-join and union of relational algebra can be mapped to the operations of SPARQL which can be further mapped to SQL. By doing this one can extend all the optimization techniques of SQL to SPARQL. However, some mismatches are bound to occur during this conversion emphasizing caution when using this method. In relational algebra, the missing variable is represented as a Null but in SPARQL it is represented as UNBOUND variable. Moreover, relational algebra rejects the tuple combination when there is a NULL value in the join attribute but in RDF relational algebra the tuple combination is rejected only when an attribute is bound on both sides with two different values. And, operators such as OPTIONAL and FILTER cannot be represented in relational algebra.

VI. SUMMARY OF FINDINGS

The choice for storing RDF data is not confined to one scheme. As illustrated in Figure 1, there are several options available and to decide which one is most suitable for a task is highly dependent on many factors such as cost, expertise of staff, dataset, etc. Each scheme has distict benefits and drawbacks and Table I list some of the immediate ones that were observed. This comparison table is not exhauxtive and further research is required to evaluate each and every feature of different available storage schemes.

Table I. Comparison of different RDF Storage Schemes

Storage Method	*Benefits*	*Drawbacks*
Triple Table	Simple to construct	Requires large no. of self-joins
Vertical Partitioning	Works well when the no. of properties is small	Does not scale well
Property Table	Reduces no. of self-joins	Requires fully structured data
Path-based Table	Schema data and instance data are easily distinguished from each other.	Requires large no. of tables

Regarding query evaluation, it is seen that triple table storage works well for all kinds of queries and also supports Triple-T, B+ tree indexes. Null values are however not supported by this storage model and the model does not work well for queries that require more number of joins. On the other hand, in the vertical partitioning storage scheme, instance queries are best supported. But similar to triple tables, null values are not supported by vertical partitioning storage schemes and it also does not scale well for queries with the * operator. Indexes built on predicate are however well supported. Property table on the other hand supports null values, and provides good support for index structures built upon subjects and predicates. However, it does not perform well for queries with the *?* operator. Path based storage, however, is appropriate for schema queries but does not work well with any other query type, and does not support null values. For path based storage, indexes built upon path are best supported. Table II shows a comparison of the query support for each data storage model discussed above. It should be noted that the table is only indicative of some aspects of querying and further analysis is required to obtain a full evaluation of query support for each data storage method.

Table II. Comparison of Query Support for different RDF Storage Schemes

Storage Method	*Null Values*	*Queries Suported*	*Queries not well Suported*	*Indices well Suported*
Triple Table	No	Any	Those having large no. of joins	B+ Trees, Indices on combination of S,P,O
Vertical Partitioning	No	Instance Queries	Queries with *	Index built on P
Property Table	Yes	Any	Queries with ?	Index built on P
Path-based Table	No	Schema Queries	Non-schema related	Index built on paths

VII. CONCLUSION

This paper has presented the RDF data model and some use cases, thus, emphasizing on the benefits of storing data in RDF. Several schemes proposed for storing RDF data has been covered and it is determined that few studies have focused on benchmarks to evaluate which scheme performs better demanding further research in this area. Several optimization techniques for query evaluation have also been discussed and summarized.

VIII. ACKNOWLEDGMENT

The author extends his sincere thanks to Prabhu, Poornima, Asim and Rekha from Indiana University, USA, for their valuable input during the writing up of an earlier version of this paper.

IX. REFERENCES

[1] Bizer, C., Cyganiak, R., & Gaub, T. 2007. The RDF Book Mashup: From Web API's to a Web of Data. Free University of Berlin.

[2] Bekett, D. 2002. The Design And Implementation Of The Redland RDF Application Framework. Computer Networks, 39(5):577-588.

[3] Berners-Lee, T. 2006. Notation 3. Ideas About Web Architecture. Retrieved from http://www.w3.org/DesignIssues/Notation3.html

[4] Guha, R., McCool, R., & Fikes, R. 2004. Contexts for the Semantic Web. In Proceedings of the 3rd International Semantic Web Conference, Hiroshima.

[5] Harth, A., & Decker, S. 2005. Optimized Index Structures for Querying RDF from the Web.

[6] Powers, S. 2003. Practical RDF. O'Reilly. CA:USA.

[7] W3C. 2004. RDF Primer:W3C Recommendation 10 February 200. Retrieved April 15 2009 from <http://www.w3.org/TR/REC-rdf-syntax/#rdfxml>

[8] Decker S., Melnik S., Fensel, D., Klein M., Broekstra J., Erdmann, M. 2000. The Semantic Web: the roles of XML and RDF, Internet Computing, Vol.4, nr. 5, pp. 63-73, IEEE

[9] W3.org. 2004 . RDF Description: W3C Recommendation, http://www.w3.org/TR/rdf-schema/ (Accessed April 2, 2009).

[10] W3.org. 2007. Semantic Web, [Online]. http://www.w3.org/2001/sw/ (Accessed April 2, 2009).

[11] Xml.com, 2000, The Semantic Web: A Primer, http://www.xml.com/pub/a/2000/11/01/semanticweb/ (Accessed April 2, 2009)

[12] R. Cyganiak. 2005.A Relational Algebra for Sparql. HP-Labs Technical Report, HPL2005-170. Retrieved from http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html

[13] George H. L. Fletcher and Peter W. Beck. 2008. A role-free approach to indexing large RDF data sets in secondary memory for efficient SPARQL evaluation, arXiv:0811.1083

[14] Markus Stocker et al. 2008. SPARQL basic graph pattern optimization using selectivity estimation. In *Proceeding of the 17th international conference on World Wide Web* (WWW '08).

[15] Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. 2005. A path-based relational RDF database. In *Proceedings of the 16th Australasian database conference - Volume 39* (ADC '05).

[16] Daniel J. Abadi et al., .2007.Scalable semantic web data management using vertical partitioning, Proceedings of the 33rd international conference on Very large data bases, September 23-27, 2007, Vienna, Austria.

[17] Octavian Udrea, Andrea Pugliese, and V. S. Subrahmanian. 2007.GRIN: A Graph Based RDF Index. In AAAI, pp. 1465–1470, 2007

[18] Thomas Neumann and Gerhard Weikum. 2008. RDF-3X: a RISC-style engine for RDF. *Proc. VLDB Endow.* 1, 1 (August 2008), 647-659.

[19] Chong, E. I. , Das,S., Eadon, G. and Srinivasan, J.2005. An efficient SQL-based RDF querying scheme. In VLDB, 2005.

[20] Alexaki, V;.et al. 2001.The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, Proceedings of the Second International Workshop on the Semantic Web (SemWeb'2001) May 2001.

[21] Baolin, L. and Bo, H. Hprd: 2007.A high performance rdf database. In NPC, 2007.

[22] Wilkinson, K., Sayers, C., Kuno, H. and Reynolds, D. 2003. Efficient RDF Storage and Retrieval in Jena2, HP Laboratories Technical Report HPL-2003-266

[23] Bonstrom, V., Hinze, A. and Schweppe, H. 2003.Storing RDF as a graph. In Proc. of LA-WEB, 2003