International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878, Impact Factor (2014): 3.05

# Review of Querying RDF Techniques with SPARQL in Semantic Web Technologies

**Vijayanathan Senthooran**

Faculty of Applied Science, Vavuniya Campus, University of Jaffna, Sri Lanka

**Abstract:** ***Efficient querying RDF triples with SPARQL plays an important role in Semantic Web data management. This paper presents a short review of an efficient RDF query to evaluate SPARQL queries, where the reversed index structure is engaged for indexing RDF triples. First, we review the design and implementation of operators on query optimization and evaluation then how it transforms a SPARQL query graph into the optimal query plan by effectively reducing the search space to determine the optimal joining order. This research study is to investigate the performance of the Semantic Data models and patterns for querying RDF data with SPARQL and enrich the techniques to improve performance of the models and patterns efficiently.***

**Keywords:** RDF, SPARQL, semantic web, Jena framework, ontology

## 1. Introduction

The Semantic Web, which is intended to establish a machine-understandable Web, is currently changing from being an emerging trend to a technology used in complex real-world applications. Semantic data is represented in Resource Description Framework (RDF), the standard language for annotating resources on the Web, and queried using the SPARQL query language for RDF that has been recently proposed by the World Wide Web Consortium. RDF data is a collection of statements, called triples, of the form <s, p, o>, where s is a subject, p is a predicate and o is an object, and each triple states the relation between the subject and the object. Such collection of triples can be represented as a directed graph, in which nodes represent subjects and objects, and edges represent predicates connecting from subject nodes to object nodes. SPARQL allows the specification of triple and graph patterns to be matched over RDF graphs. Increasing amount of RDF data on the Web drives the need for its efficient and effective management. Clearly, querying performance has become a key issue for Semantic Web applications. The objective of this research is to investigate the performance of the Semantic Data models and patterns for querying RDF data with SPARQL and enrich the techniques to improve performance of the models and patterns efficiently.

## 2. Background

Methods for query processing are an essential part of database and information systems. Queries are not only used to access information, but also for structuring and integrating information. On the World Wide Web, effective query processing used to be impossible due to the lack of data structures and scheme information. The ability to efficiently access information in a query-like fashion had been sacrificed for the ease of authoring and publishing information. When we talk about the Semantic Web today, then we mainly refer to an effort of bringing back structure to the Information that is available on the World Wide Web. This time, structures do not come in the shape of well-defined database schemas but in terms of semantic annotations that conform to a specific, often loosely defined schema or even to an explicit specification of the intended meaning of a piece of information, called ontology. The first real results of semantic web research are languages for encoding these three components: The resource description framework RDF provides a language to encode semantically annotated information, the RDF vocabulary (formerly RDF schema) [Brickley and Guha 2002] is a language for capturing schema information in terms of a typing system with inheritance and typed relations. Finally, the Ontology Web Language OWL [McGuiness and van Harmelen 2002] is a logical language that can be used to describe ontologies that further constrain the possible interpretations of terms used to annotate information. This return of structure to the World Wide Web enables us to re-consider the issue of query processing. Having a closer look at the nature of the information on the web and the languages used to impose structure on this information, we have to recognize that conventional database techniques are not sufficient to make query effective and efficient processing on the Semantic Web possible. In this research, we have to discuss the specific characteristics of the semantic web that force us to consider new techniques for query processing. We base the discussion of these characteristics on experience gained when experimenting and theoretically investigating the issue of query processing on the semantic web. We mention some approaches to overcome existing problems whenever they exist.

## 3. Short Review of RDF with SPARQL

The Semantic Web as an evolution of the World Wide Web aims to create a universal medium for the exchange of semantically described data. The idea of representing this information by means of directed labeled graphs, RDF, has been widely accepted by the scientific community. However querying RDF data sets to find the desired information often is highly time consuming due to the number of comparisons that are needed. In this article we propose indexes on RDF to reduce the search space and the SPARQL query processing time. Our approach is based on materialized queries, i.e., pre computed query patterns and their occurrences in the data sets. We provide a formal definition of RDFM at View indexes for SPARQL queries, a cost model to evaluate their potential impact on query performance, and a rewriting algorithm to use indexes in SPARQL queries. We also develop and compare

different approaches to integrate such indexes into an existing SPARQL query engine. Our preliminary results show that our approach can drastically decrease the query processing time in comparison to conventional query processing.

More formally, this paper reviews the present solutions to the following problems.

**Generation** of execution plans to cover a query. Given a query Q for a data graph G and the set I of all indexes on G, the first step is to define which indexes are usable for speeding up Q. To this end, we need to generate all possible mappings between the index pattern and the query.

**Definition** of a cost model to assess different query execution plans. Each plan differs from the others according to the parts of the query that are covered by indexes, which in turn leads to different sizes of intermediate results and different parts of the query that need to be executed and combined with the materialized parts. The objective of the cost model is to assess all possible plans and to find those with minimal estimated cost.

**Rewriting** of a SPARQL query to substitute covered query patterns by RDFMatView indexes. When a combination of RDFMatView indexes is selected, its materialized results must combine to occurrences of the covered query pattern.

There are two different cases how this may happen:

1. The combination of indexes completely covers the query pattern. Thus, the solution to the query can be generated completely by joining the indexes.
2. The combination of indexes only partially covers the query pattern. Then, the query solution needs to be generated by joining the results of the chosen indexes with the residual parts of the query pattern.

We presented a theoretical framework for using materialized SPARQL queries as indexes in [10]. In the present work, we show the practical applicability of this framework by describing and comparing several ways to integrate materialized views into an existing SPARQL query processor and by providing an evaluation on the speed-ups that can be achieved using our methods. Clearly, in a setting such as ours it is also important to provide algorithms to keep materialized queries up-to-date, and to give a user hints on which queries should be materialized to best (best in terms of space/cost-efficiency) support a given workload. However, these questions are out-of-scope of our current research.

## 4. Methodology

**Step 01**: Provide a formal representation of RDF to reduce space and time complexity.

First formal study of the basic principles of RDF statements and how they can define data graphs then how RDF data models are semantically encoded using RDFS and OWL then build own basic ontology gradually. This is leading to enrich more semantics efficiently.

**Step 02**: Define physical operators as novel to implement SPARQL efficiently.

Developing SPARQL queries step by step then executing SPARQL queries on RDF data set then investigate and optimize the query processing on RDF dataset with SPARQL.

We focus on three areas to accomplish the above steps: physical management of the RDF data, its indexing, and query optimization.

### 3.1 Native Storage of RDF Data

A native RDF database has the advantage that it can exploit the specific characteristics of the RDF data model, the RDF data, and the query language. We develop an RDF database that takes advantage of the following characteristics:

- A resource is uniquely identified by its URI
- A resource is typically subject of many statements, e.g., it has many properties
- Most queries contain a star-like pattern or a set of connected star-like pattern

The main principle of the native RDF store is to achieve high data locality, e.g., we arrange statements on database pages so that a query engine needs to access only a few of them to answer a query. The placement works as follows:

- All statements with the same subject can be found on the same page
- A page contains only resources of the same type
- A page contains an “optimum” number of resources

Additional placement strategies can be applied to reduce further the number of read pages: If it is known that pieces of information are likely to be accessed together (e.g., the address of a person) then these pieces are stored together on the same database page – we refer to it as semantic closure.

### 3.2 Constructing a Query Execution Plan

Storing the RDF data effectively is only one aspect of improving query execution. Another one is to optimize the query itself and to construct a good query execution plan (QEP). Typically the process of query execution is divided into four phases: query parsing, query rewriting, QEP generation, and QEP execution. Although we focus our research on query rewriting we developed the SPARQL Query Graph Model (SQGM) for representing SPARQL queries and supporting all phases of query processing. This model forms the key data structure for storing information that is relevant for query optimization and for transforming the query. The basic elements are the following:

- Operators consist of a head (provided variables), a body (performed operations), and additional annotations.
- Data flows indicate that an operator consumes the output of another one.

### 3.3 Query Rewriting

In the query rewriting phase, an SQGM is transformed into a semantically equivalent one to achieve a better execution strategy when the query optimizer generates query execution plans. For instance, transformation rules may aim at simplifying complex queries by merging graph patterns, e.g., avoiding join operations, and eliminating redundant or contradicting restrictions. Transformation rules change an SQGM only locally, i.e., an operator and its immediate neighbors are affected. For example, a transformation rule can merge two graph pattern operators and a join operator into a single graph pattern operator (see figure below). Rewrite strategies are more complex and affect the complete SQGM. Every rewrite strategy follows a certain goal, e.g., merge as many graph pattern operators as possible. To reach a goal it may be necessary to apply a single or a sequence of transformation rules several times. For example, query rewriting can be used to support the fast path algorithm implemented in the Jena Semantic Web Framework or the selection of indexes.

### 3.4 Indexing

We assume that users create indexes for a basic graph pattern. Creating an index means that all occurrences of the pattern in the data graph are materialized. Thus, parts of a query that are equivalent to an existing index need not to be computed at run time and information about the frequency of matches of sub graphs may be used to determine an optimal plan for query evaluation.
Involving indexes the query processing works as follows:

- Determine all eligible indexes I for a query.
- Determine all maximal sets of overlapping indexes contained in I.
- For each set estimate its selectivity.

### 3.5 Experimental design

We will implement the SQGM on top of Jena and ARQ.

ARQ is a query engine for Jena that supports the SPARQL RDF Query language. SPARQL is the query language developed by the W3C RDF Data Access Working Group.

Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine.

Jena is open source and grown out of work with the HP Labs Semantic Web Programme.

The Jena Framework includes:

- A RDF API
- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API
- In-memory and persistent storage
- SPARQL query engine

**Steps**

- The ARQ query processor parses the query and generates a query model specific to ARQ.
- Then, the generated query model is translated into an SQGM
- Heuristics are applied to provide a good basis for an efficient query execution plan (QEP).
- Finally, the restructured SQGM is translated back into an ARQ query model which is executed.

## 5. Conclusion

In this paper we presents a review of l RDF query engine for efficient SPARQL query processing and the key points of our work are: (1) an IR based solution for indexing triples and a set of highly-efficient operators for query optimization and evaluation, (2) a set of RDF statistics for estimating the execution cost of the query plan, and (3) a main-tree-shaped optimization algorithm for identifying the optimal query plan. Currently we mainly focus the query optimization problem on SPARQL basic graph patterns, and union and optional patterns. In the future we plan to extend to develop an RDF engine to support filter clause and named graph in the SPARQL by extending the existing statistics, indexing scheme and operators.

## References

[1] Manola, F., Miller, E.: RDF Primer (February 2004) W3C Recommendation.
[2] Wilkinson, K., Sayers, C, Kuno, H., Reynolds, D, Efficient RDF storage and retrieval in Jena2. In: Proc. First International Workshop on Semantic Web and Databases. (2003)
[3] Stephen Harris, N.G Efficient bulk rdf storage, in 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03). (2003)
[4] Groppe, S., Groppe, J., Linnemann, V.: Using an Index of Precomputed Joins in order to speed up SPARQL Processing. In Cardoso, J., Cordeiro, J., Filipe, J., eds.: Proceedings 9th International Conference on Enterprise Information Systems (ICEIS 2007 (1), Volume DISI), Funchal, Madeira, Portugal, INSTICC (June 12 - 16 2007) 13–20
[5] Stocker, M., Seaborne, A., Bernstein, A., Kiefer C., Reynolds .D. Sparql basic graph pattern optimization using selectivity estimation. In: Proc. of WWW. (2008)
[6] Zhang, L., Liu, Q., Zhang, J., Wang, H., Pan, Y., Yong, Y.: Semplore: An ir approach to scalable hybrid query of semantic web data. In: Proc. Of ISWC. (2007)
[7] Harth, A., Umbrich, J., Hogan, A., Decker, S.:Yars2: A federated repository for querying graph structured data from the web. In: Proc. of ISWC.(2007)
[8] Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: Scalable semantic web data management using vertical partitioning. In: VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment (2007) 411–422
[9] Marcelo Arenas, Claudio Gutierrez and Jorge Perez, Foundations of RDF Databases, Department of Computer Science, Pontificia Universidad Catolica de Chile 2 Department of Computer Science, Universidad de Chile
[10] Marcelo Arenas and Jorge Perez, Querying Semantic Web Data with SPARQL: State of the Art and Research Perspectives, Pontificia Universidad Catolica de Chile, Universidad de Chile