RDFMatView: Indexing RDF Data for SPARQL Queries

Roger Castillo

Humboldt Universitaet zu Berlin
{castillo}@informatik.hu-berlin.de
http://www.hu-berlin.de/

Abstract. The Semantic Web is now gaining momentum due to its efforts to create a universal medium for the exchange of semantically tagged data. The representation and querying of semantic data have been made by means of directed labelled graphs using RDF and SPARQL, standards which have been widely accepted by the scientific community. Currently, most implementations of RDF/SPARQL are based on relational database technology. But executing complex queries in these systems usually is rather slow due to the number of joins that need to be performed. In this article, we describe an indexing method using materialized SPARQL queries as indexes on RDF data sets to reduce the query processing time. We provide a formal definition of materialized SPARQL queries, a cost model to evaluate their impact on query performance, a storage scheme for the materialization, and an algorithm to find the optimal set of indexes given a query. We also introduce different approaches to integrate materialized queries into an existing SPARQL query engine.

Key words: SPARQL, Indexing, RDF, Materialized Queries, Semantic Web

1 Introduction

The *Semantic Web* aims to create a universal medium for the exchange of data where data can be shared and processed by automated tools as well as by people. The basis for this proposal is a logical data model called Resource Description Framework (RDF) [1]. An RDF data set 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. Each triple states the relation between subject and object. A set of triples can be represented as a directed graph where subjects and objects represent nodes and predicates represent edges connecting these nodes. The SPARQL query language is the official standard for searching over RDF repositories [2].

The increasing amount of RDF data have motivated the development of approaches for efficient RDF data management. Initially, most SPARQL implementations were built upon relational databases (e.g. Jena [3], 3Store [4, 5], or Sesame [6]).

Some other approaches have proved to be very efficient based on relational database technology but following a native data scheme (see for instance [7–9]). Basically, they propose data structures, which transform the triples table into several tables according to different conditions and create indexes by combining triple elements *(s,p,o)* in all possible ways.

In contrast to these systems, our approach fundamentally is based on the assumption that some patterns are combined more frequently than others, and that only indexing those combinations promises to provide large speed-ups at manageable space and maintenance cost. Note that we do not claim that our approach offers a particular fast SPARQL-processor, compared to systems such as [7–9]. Instead, we present a new technique to speed up query execution with SPARQL that is applicable to any SPARQL query processor. We want to showcase its potential and compare different ways to integrate it into query processing using one particular system (namely ARQ [10]), which has been chosen because of its widespread use. An extended version of this paper can be found in [11].

1.1 Contributions

In our research, we capture the following contributions:

- We propose an indexing method, which fully exploits the RDF graph-structure. We do not index single attributes or triples, but fractions of queries that occur frequently in an expected workload. Therefore, our approach is a *native RDF/SPARQL indexing* method whose concepts are viable for all possible implementations of RDF stores.
- We provide an indexing method using *Materialized Queries* for RDF data sets. Our method can be seen as an orthogonal indexing solution that acts as cache between SPARQL query processors and triple stores, which may be used in conjunction with other indexing methods.

2 The RDFMatView Approach

RDFMatView is a theoretical framework for using materialized SPARQL queries as indexes [11]. Formally, RDFMatView defines an index as follows:

Definition 1 (Index). *An* index I *over a data graph* G *is a pair* $I = (P, O)$*, where* P *represents a pattern*[1] *and* O *represents the set of all occurrences of* P *in* G*.*

Indexes are precomputed queries suitable to speed up other queries when the index pattern is entirely contained in the query pattern. We say, an index I is eligible for a query Q when the patterns of I are entirely contained in the patterns of Q [12]. However, it would be more advantageous when query processing uses more than one index. For those cases, we require that indexes overlap. Overlapping indexes are good candidates for reducing query processing

[1] Set of triple patterns in the body of I.

because the query engine can combine occurrences of these indexes and generate solutions without matching against the RDF dataset. To estimate which indexes *bring more savings in time* to query execution, the optimizer must choose an index, a set of indexes or decide to execute the query without using indexes.

We propose a simple model for estimating these savings by using the concept of *selectivity* of an index. It defines the relation of the number of occurrences of an index in a given graph to the possible total number of index occurrences in the graph. Having computed the selectivity of all indexes, the query optimizer must determine which index or set of indexes is the best for query processing.

3 Methodology

To integrate our approach into an existing SPARQL query processor we developed three strategies. Our first strategy (MatView-and-ARQ) uses ARQ to process the residual part of the query. RDFMatView extends the results of the chosen cover by joining the partial solutions with the solutions of the residual patterns. The second strategy (MatView-to-SQL) is based on a SPARQL-to-SQL algorithm to translate SPARQL queries into SQL queries. The idea is to directly access the native Jena storage tables and to combine those results with the index tables to generate the final solution. The last strategy (Hybrid) is built from a combination of the previous two strategies, i.e. ARQ and database execution engine. We present these strategies for the ARQ system [10]. However, we want to stress that general process would be the same for any other SPARQL query processors.

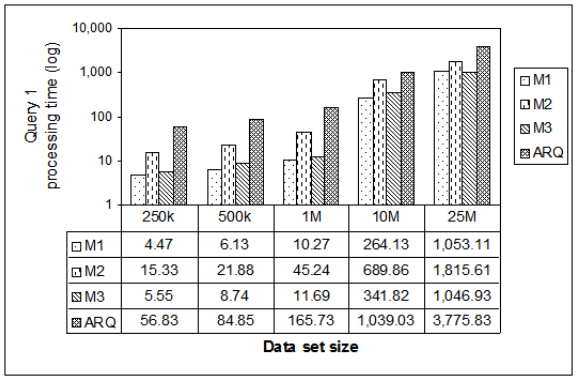
4 Evaluation

Because of space reasons, we present an evaluation of our approach by executing one query on five data sets [13] ranging from 250K to 25M triples using our three RDFMatView methods and plain ARQ (without indexes). A description of the tested query is shown in Listing 1.

Listing 1. Query1 finds products for a given set of generic features.

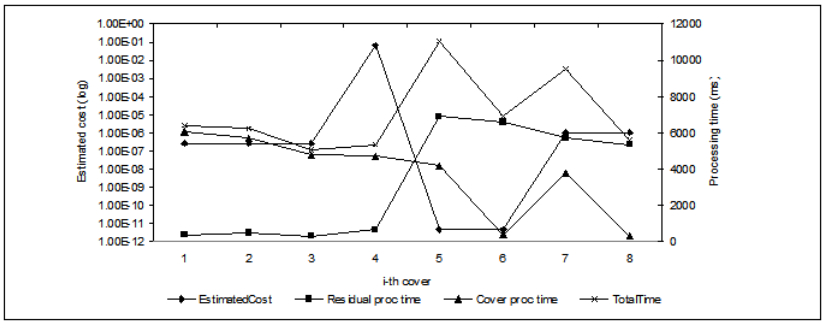
```
SELECT * WHERE {
    ?product rdfs:label ?label .
    ?product rdf:type ?ProductType .
    ?product bsbm:productFeature ?ProductFeature1 .
    ?product bsbm:productFeature ?ProductFeature2 .
    ?product bsbm:productPropertyNumeric1 ?value1 .}
```

Figure 1(a) illustrates the average processing time of Query1 in 5 different datasets. Clearly, processing time significantly improves when using MatView-and-ARQ (M1) and Hybrid (M3). The improvements are the higher, the larger the database. However, processing time does not significantly improve when using MatView-to-SQL (M2). The reason for this is the Jena native storage schema. Since the values are encoded following the Jena layout, our process needs to parse



	250k	500k	1M	10M	25M
M1	4.47	6.13	10.27	264.13	1,053.11
M2	15.33	21.88	45.24	689.86	1,815.61
M3	5.55	8.74	11.69	341.82	1,046.93
ARQ	56.83	84.85	165.73	1,039.03	3,775.83

(a) Methods comparison



(b) Processing Query1

Fig. 1. Figure 1(a) shows the processing time for Query1 on five data sets using three rewriting methods. M1: MatView-and-ARQ; M2: MatView-to-SQL; M3: Hybrid; ARQ: plain ARQ (y-axis in logarithmic scale). Figure 1(b) shows estimated and real costs for query1.

the stored values and extract the required information. Figure 1(b) show that the costs estimated by our model roughly correlate with the real processing time and that it manages, in all cases, to prevent the selection of exceptionally bad plans. Actually all plans improve the total execution times when compared to those without using indexes. However, they also show that our model can be improved. Especially, it does not yet reflect that using less indexes is advantageous as this requires less joins at runtime. This fact is captured only indirectly by our model as we concentrate on the number of covered patterns.

5 Conclusions and Future Work

We have proposed a novel logical and physical framework to speed-up the execution of SPARQL queries by using materialized queries as indexes. The use of

RDFMatView indexes focuses on minimizing query pattern comparison against the RDF data set and on minimizing the number of self-joins to answer a query. We analyze query and index patterns and provide three rewriting methods to use indexes and get the final query result set. We also provide a simple cost model to estimate the savings in time these indexes may bring to query execution. Our preliminary results have shown that the performance gains are considerable. As future directions, and in addition to the previous described ideas, we plan to use variable bindings stored in the index structures to restrict uncovered query patterns. Another promising topic to extend our approach is index selection for SPARQL queries taking into account the particularities of RDF and SPARQL, especially in the area of cost models and scope of selectable indexes.

References

1. Manola, F., Miller, E.: RDF Primer (February 2004) W3C Recommendation.
2. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (April 2008) W3C Recommendation.
3. 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)
4. Stephen Harris, N.G.: 3store: Efficient Bulk RDF Storage. In: 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03). (2003)
5. Harris, S.: SPARQL Query Processing with Conventional Relational Database Systems. In: International Workshop on Scalable Semantic Web Knowledge Base System. (2005)
6. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: International Semantic Web Conference. (2002) 54–68
7. 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
8. Weiss, C., Karras, P., Bernstein, A.: Hexastore: Sextuple Indexing for Semantic Web Data Management. Proc. VLDB Endow. **1**(1) (2008) 1008–1019
9. Neumann, T., Weikum, G.: RDF-3X: a RISC-style engine for RDF. Proc. VLDB Endow. **1**(1) (2008) 647–659
10. ARQJena: ARQ - A SPARQL Processor for Jena. http://jena.sourceforge.net/ARQ/ (2010)
11. Castillo, R., Leser, U., Rothe, C.: RDFMatView: Indexing RDF Data for SPARQL Queries. Technical Report 234, Humboldt Universitaet zu Berlin (2010)
12. Halevy, A.Y.: Answering Queries Using Views: A Survey. The VLDB Journal **10**(4) (2001) 270–294
13. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal On Semantic Web and Information Systems - Special Issue on Scalability and Performance of Semantic Web Systems, 2009 (2009)