Materializing Inferred and Uncertain Knowledge in RDF Datasets

James P. McGlothlin, Latifur Khan

The University of Texas at Dallas
Richardson, TX, USA
{jpmcglothlin, lkhan}@utdallas.edu

Abstract

There is a growing need for efficient and scalable semantic web queries that handle inference. There is also a growing interest in representing uncertainty in semantic web knowledge bases. In this paper, we present a bit vector schema specifically designed for RDF (Resource Description Framework) datasets. We propose a system for materializing and storing inferred knowledge using this schema. We show experimental results that demonstrate that our solution drastically improves the performance of inference queries. We also propose a solution for materializing uncertain information and probabilities using multiple bit vectors and thresholds.

Introduction

The RDF data format is the standard mechanism for describing and sharing data across the web. OWL (Web Ontology Language) specifies inference rules which allow additional knowledge to be derived from known facts. The goal of our research is to improve the efficiency and scalability of queries against this data. The core strategy of our research is to perform inference at the time the data is added rather than at query time. All inferred triples are persisted in our relational database. Queries requiring inference become both simpler and more efficient.

This would not be a viable solution unless the inferred triples can be added and stored without increasing the performance cost of queries. We have designed a bit vector schema that can store these triples with negligible impact to query performance. Our bit vectors enable joins and unions to be performed as bitwise operations, and our inference materialization reduces the need for subqueries, joins, and unions at query time. The end result is that query performance is improved. Our evaluation results show that our solution outperforms the current state-of-the-art solutions for RDF storage and querying.

We also present a framework for storing uncertain information and probability data. Every triple is persisted including its probability and an explanation of how it was inferred. We define a multiple bit vector schema involving thresholds. This solution materializes uncertain information and makes this knowledge available to queries.

Bit Vector Schema

We have designed a bit vector schema, RDFVector, specifically for RDF datasets. We store the RDF triples in three bit vector tables: the POTable, the SOTable and the PSTable. Each URI or literal is dictionary-encoded to a unique ID number which is the index into the bit vectors. As an example, the POTable includes four columns: the PropertyID, the SubjectID, the SubjectBitVector and a BitCount. The SubjectBitVector has a 1 for every subject that appears with that property and object in a RDF triple. For example, all subjects matching *<?s type text>* can be retrieved from a single tuple and returned as a single bit vector, indexed by the ID numbers. The BitCount is used to support aggregation queries and query optimization.

The bit vectors are the reason we are able to store inferred triples at little or no cost. There are bits in each vector for every known URI, therefore inferring additional triples just involves changing 0s to 1s. In the next section, we present an example of such an inference. Our solution incurs a performance penalty only when the dataset's vocabulary (number of unique URIs) is increased, and OWL inference usually adds only a few terms to the vocabulary. For our experiments, we utilize the LUBM (Lehigh University Benchmark, http://swat.cse.lehigh.edu/projects/lubm/) dataset with >44 million RDF triples. For this dataset, 20,407,385 additional triples are inferred, yet only 22 unique URIs are added by this inference. Our schema pays no performance penalty for the addition of 20 million inferred triples, only for the 22 new URI terms.

The bit vectors can become quite large. We have tested with datasets that have over 18 million unique URIs. We use compression to take advantage of the sparse nature of the vectors. We utilize D-Gap (http://bmagic.sourceforge.net/dGap.html) compression, an algorithm that has the key advantage that it allows bitwise operations to be performed against compressed data, thus eliminating the need to decompress the vectors.

In addition to the bit vectors, we provide a triples table. The triples table is the only table exposed to the inference rules and all user additions, deletions and updates are performed against this table. The triples table allows us to encapsulate our schema and to optimize processes for pushing updates across our other tables. The triples table also contains the inference count which is used to support deletions of inferred triples. The triples table is not used during queries; it is used only for manipulating the dataset.

Inference

Consider the triple *<Professor0 type FullProfessor>*. The LUBM ontology will allow us to infer 4 additional triples: *<Professor0 type Professor>*, *<Professor0 type Faculty>*, *<Professor0 type Employee>*, *<Professor0 type Person>*. Our strategy is to materialize and store all 4 inferred triples. As *Professor, Faculty, Employee* and *Person* exist elsewhere in the dataset, no new vocabulary is introduced. All that is required to add these triples to the bit vector tables is to change 0s to 1s in the vectors; the size of the bit vectors is not increased. Now, we can execute a query such as *List all persons* by reading a single bit vector. To query all persons with vertical partitioning (Abadi et al., 2007) or RDF-3X (Neumann&Weikum, 2008) would require 21 subqueries and 20 unions.

We provide an inference engine to register and manage inference rules. When a triple is added to the triples table, the inference engine iterates through the inference rules and allows each rule to add inferred triples. We have implemented inference rules for all OWL constructs.

Deletions offer a special challenge for inferred triples. If the base triple is deleted, then any inferred triples should be deleted as well. However, a triple might be inferred from more than one base triple. We solve this problem by maintaining an inference count in the triples table, and only deleting the inferred triple when the count becomes 0.

Uncertainty Reasoning

OWL has provable inference. All inferred triples are known to be true with absolute certainty. Uncertainty reasoning addresses the issue of knowledge that might be true. Our goal is to associate probability numbers with facts (RDF triples) in the database. There is no standard for representing probability in an ontology. Our goal is not to restrict the ontology representation. Therefore, as with provable inference, we allow inference rules to register.

To support probability, we modify the triples table and the bit vector tables. For the triples table, we add extra columns for the probability and an explanation of the inference. For the bit vector tables, we add a threshold column. The vectors become bit vectors with 1s for every triple known with probability>=threshold. For vectors without probability, the threshold is 1. Below is an example POTable with probabilities and thresholds:

Property	Object	Threshold	SubjectBitVector	Count
hasDisease	LungCancer	1.0	101001010100100000	6
hasDisease	LungCancer	0.75	101101010101100001	9
hasDisease	LungCancer	0.5	111101010101101011	12
hasDisease	LungCancer	0.25	111101110101101011	13

We can now retrieve all subjects with >0.5 probability of having lung cancer by reading a single tuple, and we can still use bitwise operations to join vectors. If a query requests a different probability than provided by the threshold, we use the bit vectors for efficient selection and then we access the triples table for precise comparison.

We propagate the probabilities throughout the system based on the simple Bayesian logic rule that $P(A)=P(A|B)*P(B)$. We recognize that there can be ambiguities when 2 inference rules attempt to add the same triple with different probabilities. Our goal in this framework is not to solve such ambiguities. We support and register exception handlers to enable flexibility and to allow existing reasoning technologies to be used. Our contribution is that once the probabilities are calculated they can be persisted and made available for efficient querying.

Evaluation Results

Due to space limitations, we only present a summary of our results for one dataset (LUBM) for the queries involving inference. We compared with RDF-3X and vertical partitioning. All of our experiments here were performed on a Dell M6400 laptop with 8GB RAM. Our run times for LUBM (for the 44 million triples dataset) are:

	Q3	Q5	Q6	Q11	Q13
Time(sec)	0.15	0.23	0.14	0.24	0.21
%Improvement	67.4%	98.0%	60.8%	84.7%	81.1%

The %improvement is in comparison to the next fastest solution for that query. In most cases, this is RDF-3X.

LUBM Query 5: *List persons who are members of a particular department.* This query invokes five kinds of inference rules. Since our solution materialized the inferred triples, we can just query the POTable. We retrieve the tuples for *property=type, object=Person* and *property=memberOf, object=Department0,* and execute the bit operation *and* between the bit vectors. We are able to perform this query in 0.23 seconds. RDF-3X requires 2.88 seconds and vertical partitioning requires 56.28.

Future Work

In the future, we plan to identify or create large benchmarks to test our uncertainty solution for scalability and efficiency. We also plan to improve our framework for handling ambiguities and to integrate with available ontology tools such as BayesOWL and PR-OWL.

References

Abadi, D.J.; Marcus, A.; Madden, S.; Hollenbach, K.J. 2007. Scalable Semantic Web Data Management Using Vertical Partitioning. *In Proc. of VLDB*, 411-422.

Weiss, C.; Karras, P.; Bernstein, A. 2008. Hexastore: sextuple indexing for semantic web data management. *In Proc. of VLDB*, 1008-1019.

Neumann, T.; Weikum, G. 2008. RDF-3X: a RISC-style engine for RDF. *In Proc. of VLDB*, 647-659.

Costa, P.C.G.D.; Laskey, K.B.; Laskey, K.J. 2008. PR-OWL: A Bayesian Ontology Language for the Semantic Web. *In Proc. of URSW*, 88-107.

Ding, Z.; Peng, Y. 2004. A Probabilistic Extension to Ontology Language OWL. *In Proc. of HICSS.*