Efficient Query Processing for RDF data Using XML Repository

Win Lai Hnin, Khin Nwe Ni Tun
University Of Computer Studies, Yangon, Myanmar
winlaihnin.84@gmail.com, knntun@gmail.com

Abstract

The Semantic Web is an extension of the current Web that will allow to find, share, and combine information more easily. To harvest such power requires robust and scalable data repositories that can store RDF data. Most of the existing RDF storage techniques rely on relation model and relational database technologies for these tasks. The mis-match between the graph model of the RDF data and the rigid 2D tables of relational model jeopardizes the scalability of such repositories and frequently renders a repository inefficient for some types of data and queries. In this paper, we propose a system that can store RDF data in the XML repository and perform the efficient XML query processing. We discuss the basic idea of serializing RDF data into RDF/XML and mapping of RDF/XML to XML document and then algorithm for the efficient XML query processing to the performance of XML query evaluation.

1. Introduction

Most of the web sites today are designed for human reading, not for computer understanding. Computers essentially play a role in parsing web pages for displaying and processing jobs. They have no reliable way to draw the semantics from a page. The Semantic Web will improve the meaningful content of the web pages. It is not completely a new generation of web, but an expansion of the current one. The meaning in the Semantic Web is mostly represented by Resource Description Framework (RDF). RDF encrypts these meanings in the sets of triples that build meaningful webs about related things. These are recognized by the Universal Resource Identifiers (URIs) which tie meanings to a unique definition so that users can easily find them and their relationships on the web.

As the W3C standard document format for writing and exchanging information on the Web, XML (eXtensible Markup Language) is mostly concerned about syntax. XML is textual language quickly gaining popularity for data representation and exchange on the Web. Nested, tagged elements are the building blocks of XML. Each tagged element has a sequence of zero or more attribute/value pairs, and a sequence of zero or more subelements. These subelements may themselves be tagged elements or they may be "tagless" segments of text data. XPath is a declarative query language for XML that provides simple syntax for addressing parts of an XML document. XPath can specify sets of nodes and sets of paths in an XML document tree. XML queries are significantly different from the conventional *RDBMS* queries in that the former routinely involve a tree-shaped pattern that is to be matched against the database, and the queries are commonly referred to as *TPQ*s (Tree Pattern Queries). Furthermore, TPQs often contain redundancies, especially when constraints such as those induced from the DTD are additionally considered. Redundancies are detrimental to the performance of XML query evaluation. Therefore, studying efficient mechanisms for TPQ minimization is of great importance for XML query processing.

The needs to develop applications on the Semantic Web and support search in RDF data call for RDF repositories to be reliable and robust. As in the context of RDB and XML, the selection of storage models is critical to a data repository as it is the dominating factor to

determine how to evaluate queries and how the system behaves when scales up.

The rest of this paper is organized as follows: In the next section, we discuss the translation from RDF to XML. In section 2.1, we discuss the RDF data model and then in section 2.2, we describe the serializing from RDF to RDF/XML and then we describe the XML document in section 2.3. And then we describe XPath query language for XML in section 3 and we describe the algorithm for identify and remove redundant nodes in section 4.1 and the conditions for remove redundant nodes are described in section 4.2. We discuss the experimental results in section 5. Finally, we conclude our paper.

1.1 Related Work

Most of the existing RDF data repositories [2, 4, 5] rely on relational models for data storage and evaluate SPARQL queries by rewriting them into SQL queries and then executing them in the RDB engine. Among them there are two major directions:(1) keeping the simple triple data model of RDF data, e.g. *triple store* [6]; and (2) decomposing RDF triples into relations, either based on predicates, e.g. *vertical partition* or based on semantics, e.g. *property table* [4].

The *triple store* does not scale well as the evaluation of a complex SPARQL query invokes many self-joins. Various indexing techniques [1] were proposed as remedies, at the cost of huge increase in storage space and decrease in the scalability and update efficiency. The *vertical partition* [2] works well for SPARQL queries when all predicates in the WHERE clause are known. Otherwise, all tables have to be accessed and results unioned. For example, the RDF data in Fig. 1(a) are stored in five tables. All need to be accessed to evaluate the SPARQL query above. The *property table* incurs small number of joins for some queries because a selection in one property table can match multiple simple access patterns. However it suffers storage redundancy and large overhead in query evaluation [2].

An alternative approach [8] preserves the graph nature of RDF data by storing RDF graphs in an object-relational database. However, this separates the RDF schema and RDF primary data, which brings difficulties in evaluating queries containing both schema and data instances.

The proposal of serializing RDF graph into XML trees to utilize existing XML technologies [3, 7] focused on representing all RDF features such as blank node in XML, but pays less or no attention to the efficiency of RDF data storage and query evaluation. It either leads to XML data [8] with large redundancy or flat XML data that cannot take full advantage of XML query evaluation techniques.

Mo Zhou and Yuqing Wu [6] proposed the two RDF-to-XML decomposition algorithms for the decomposition in two steps: (1) the schema-level decomposition which maps an RDF schema to a set of XML schemas and (2) the data-level decomposition which maps RDF data to a set of XML documents conforming to the XML schemas which brings inefficient in mapping RDF data to a set of XML documents conforming to the XML schemas in some applications.

1.2 Overview of the Proposed System

Specifically we propose to serialize RDF data into RDF/XML and map RDF/XML to XML documents in an XML repository and XPath queries to be evaluated against the XML data using the latest XML query evaluation techniques. The desired properties of an XML storage model to be as follows: (1) preserving semantics to facilitate efficient evaluation of XPath queries (2) high performance in evaluating all XPath queries rather than only some types of XPath queries (3) high scalability powered by no or small storage redundancy.

Our contribution can be summarized as follows:

- We propose an XML-based RDF storage that doesn't depend on the XML schema.

- We propose serializing from RDF into RDF/XML and the mapping from RDF/XML to XML documents.
- We discuss algorithm for efficient XML query processing to extract information from XML repositories.

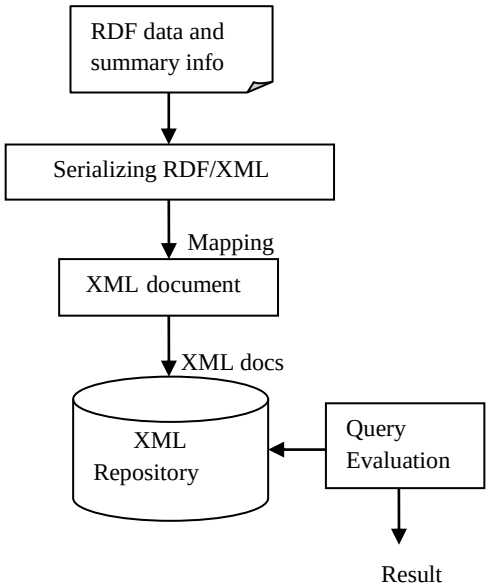

Figure 1. System Architecture

RDF data are significantly different from XML data in syntax and data model: RDF data and schema are directed graphs with both nodes and edges labeled, while XML data are trees with only nodes labeled. Although our work, as other RDF storage approaches, is syntax independent, the difference between the data models brings substantial challenges to storing and querying RDF data using XML techniques, in transforming graphs into trees, keeping storage efficiency and mapping graph pattern queries into tree pattern queries.

2. Knowledge Representation

There are three essential requirements for arbitrary language used for data interchange on the web:

- Language should have the ability to describe any form of data to satisfy all the potential need.
- The represented data should be easily accessed by other organizations and its supported software, such as parsers or query APIs, should be reusable (syntactic operability).
- It should have definitions for mappings between terms in the data (semantic interoperability).

2.1 RDF

The vision of the Semantic Web is to allow everybody to publish interlinked machine-processable information with the ease of publishing a web page. The basis for this vision is a standardized logical data model called Resource Description Framework (RDF). 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; each triple states the relation between the subject and the object. A collection of triples can be represented as a directed typed graph, with nodes representing subjects and objects and edges representing predicates, connecting subject nodes to object nodes. Basic RDF data model consist of three objects:

Resources : an element, a URI, a literal,…
Properties : directed relations between two resources
Statement : combination of a resource, a property and a value.

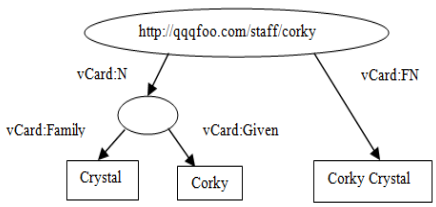

Figure 2. RDF data

In this example, one could create triple with the subject http://qqqfoo.com/staff/corky, the predicate vCard:FN, and a value as object which is identified by a literal "Corky Crystal". RDF offers the concept of blank nodes (also known as anonymous resources). Blank nodes allow the creation of resources without needing a URIref, since the node itself provides the necessary connectivity between the various other parts of the graph. In Figure (2), one could create another triple with the subject http://qqqfoo.com/staff/corky, the predicate vCard:N and the blank node that represents the data type properties vCard:Family and vCard:Given. The range of Family and Given properties is of type string. The RDF database integrates vocabulary from different namespaces, i.e. the standard RDF namespace rdf, a user-defined namespace vCard, as well as the namespaces foaf, dc and dcterms. The standard namespace rdf provides some basic vocabulary with predefined semantics, such as rdf:type used for typing URIs. Next, foaf provides domain-specific vocabulary to describe persons in a uniform way and dc and dcterms provide predefined vocabularies for describing bibliographic entities.

2.2 Serializing from RDF to RDF/XML

RDF/XML is the widespread serialization format for RDF graphs. RDF/XML is the normative syntax for writing RDF. The success of RDF/XML lies in its early availability and the number of tools that support RDF/XML processing. Therefore, RDF/XML is the recommended syntax for applications to exchange RDF information. The basic principle of RDF/XML files is the mapping of RDF nodes and arcs into XML elements, attributes, element content, and attribute values. Probably the most prominent serialization format is RDF/XML which allows encoding RDF databases as XML trees. The basic idea behind RDF/XML is to split the RDF graph into small, tree-structured chunks, which are then described in XML with the help of predefined tags and attributes. The RDF/XML format was primarily designed to be processed by computers. We propose pseudo code for serializing from RDF into RDF/XML is:

```
Input ←— Subject, Predicate, Object
Output ←— Serializing RDF/XML

IF (Subject is root node)
  THEN display root node in the about attribute
of the Description element and then display
predicate and object.
    ELSEIF (Object is BagID)
       THEN display Bag Element and then
         display its properties and value
       ELSE IF (Object is SeqID)
            THEN display Seq Element and then
                  display its properties and value
          ELSE IF (Object is AltID)
               THEN display Alt Element and then
                     Display its properties and
                       value
          ENDIF
        ENDIF
     ENDIF
   END
```

Pseudo code for serializing RDF/XML

An RDF graph can be considered as a collection of paths of the form- node, predicate arc, node, predicate arc, node, predicate arc … node, which cover the entire graph. In RDF/XML, these turn into sequences of elements inside elements which alternate between elements for nodes and predicate arcs. This has been called a series of node/ arc stripes, where the node at the start of the sequence turns into the outermost element; the next predicate arc turns into a child element, and so on.

Example :

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf =
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:vCard =
"http://www.w3.org/2001/vcard-rdf/3.0#">
```

```
    <rdf:Description rdf:about =
"http://qqqfoo.com/staff/corky" >
        <vCard:FN> Corky Crystal </vCard:FN>
        <vCard:N rdf:parseType="Resource">
            <vCard:Family> Crystal
</vCard:Family>
            <vCard:Given>  Corky
</vCard:Given>
        </vCard:N>
</rdf:Description>
</rdf:RDF>
```

RDF/XML for Figure (2)

In this example, the outer rdf:RDF XML element encloses the scope of the RDF/XML. The inner rdf:Description element is the "frame-style" block of properties, all about the resource with URI http://qqqfoo.com/staff/corky. Here the element vCard:FN represents the property with the value "Corky Crystal". This element encodes for the URI reference that is defined by the namespace name (URI) for "vCard" which in this case is http://www.w3.org/2001/vcard-rdf/3.0#/ concatenate with the local name of the element (FN) giving the URI http://www.w3.org/2001/vcard-rdf/3.0#/FN.
When a property has a URI value, an rdf:resource attribute is used on the empty property element with the URI as the attribute value. A property value can have an XML content when the parseType="Resource" attribute is used on the property element

2.3 XML

XML is a meta-language that enables designers to create their own customized tags to provide functionality not available with HTML. XML is a restricted version of SGML, designed especially for Web documents. SGML allows document to be logically separated into two: one that defines the structure of the document (DTD), other containing the text itself. XML retains key SGML advantages. XML is not intended as a replacement for SGML or HTML. It is a data format for exchanging data on the web, between databases and elsewhere. Elements or tags are most common form of markup. First element must be a root element, which can contain other (sub) elements. XML document must have one root element. Element begins with start-tag and end-tag. XML element is case-sensitive. Attributes are name-value pairs that contain descriptive information about an element. A given attribute may only occur once within a tag, while (sub) elements with same tag may be repeated.

In this paper, we map the RDF/XML to XML document. RDF/XML has an XML syntax that has a specific meaning. Every Description element describes a resource. Every attribute or nested element inside a Description is a property of that resource. Tags and attributes have a specific meaning and we can refer to resources by using URIs. The following is an example of XML-tagged document, contained in the file **staff.xml**.

```
<?xml version="1.0"?>
<Staff id="corky"

xmlsn:vCard="http://www.w3.org/2001/vcard-
rdf/3.0#"
    <vCard:FN> Corky Crystal </vCard:FN>
    <vCard:N>
        <vCard:Family> Crystal
</vCard:Family>
        <vCard:Given> Corky </vCard:Given>
    </vCard:N>
</Staff>
```

staff.xml document for Figure (2)

3. XPath Query Language

XPath is designed for XML documents. It provides a single syntax that we can use for queries, addressing and patterns. Fundamentally, an XPath is an expressing. Evaluating an XPath expression results in one of the following:

- A node set
- A Boolean
- A floating-point number
- A String of Unicode character

Specifically, identity constraints require the resultant node set to contain only elements or attributes. Fragment identifiers restrict the resultant node set to contain only elements.

Location paths nominally provide the grammar for typical XPath expressions for XML schemas. In an XML schema, all location paths are either relative to an enclosing component (for identity constraints) or relative to an entire XML document (for locating schema components). One of the general features of a location path is the ability to navigate along a number of axes. An *axis* specifies a direction of movement in the node tree. For example, you might specify a *child* node, an *attribute* node, an *ancestor* node, or a *descendant* node. The XPath Recommendation defines 13 axes. An identity constraint is limited to containing only the axes *child*, *attribute*, and *descendant-or-self*. Furthermore, an identity constraint can only use the shortcut notation for these axes. Predicates are very powerful, but slightly confusing when first encountered. A predicate is strictly a filter. A predicate filters out desired nodes from a node set.

The easiest way to demonstrate a predicate is to discuss two similar expressions along multiple axes. Examples of XPath queries against staff.xml document are the following:

(1) /Staff/N/Family (selects Family element that children of N element that is children of the root element Staff).
(2) ///Family (selects Family element in the document.
(3) /Staff/* (selects all child elements of the root element Staff
(4) /Staff[@id] (selects the id attribute of the Staff element

4. Identify and Remove Redundant Nodes

4.1 Algorithm

XML queries are significantly different from the conventional *RDBMS* queries in that the former routinely involve a tree-shaped pattern that is to be matched against the database, and the queries are commonly referred to as *TPQ*s (Tree Pattern Queries). Furthermore, TPQs often contain redundancies. Redundancies are detrimental to the performance of XML query evaluation. So, we consider the query minimizing algorithm. To abstract from existing query languages for XML, we express queries as tree patterns where nodes are types and edges are *parent-child* or *ancestor-descendant* relationships. Among all the nodes of a query *Q*, one is designated as the output node, denoted by *output (Q)*, corresponding to the output of the query.

Our query minimizing algorithm is given below:

Algorithm *query-minimization Q*
input: *Q*
output: minimized query – *Q*
begin
1. **for** *i*= 1 to *n* **do**
2. {**_if** *i*= output(Q) **then** c_{ii} := 1;
3. **else if** *i* is a leaf **then{ for** *j*= 1 to *n* **do if** $\lambda(i)=\lambda(j)$ **then** c_{ij} := 1;}
4. **else**
5. { let i_1, $i_{2,\ldots,}$ i_k be the children of *i*;
6. **for** *j*= 1 to *n* **do**
7. **if** *j* exists **do**
8. {**_if** λ (i)=λ(j) **then**
9. { **_if** for each child edge $(i,i_l)(1\leq l\leq k)$, $f(i,i_l,j)$ *return true* and
10. for each descendant edge $(i,i_l)(1\leq l\leq k)$, $g(i,i_l,j)$*return true*
11. **then** *ci j* := 1;}}}
12. let $j_1,j_2,..,j_h$ be the nodes that cover *i*;
13. set d_{ipl} = 1 for each p_l, where p_l is the parent of j_l ($1\leq l\leq h$);
14. let $\{q_{1.....}q_m\}$ be a set such that each node in it is an ancestor of some j_l. Set $a_{iql=}$ 1 for each q_1 ($1\leq l\leq m$);
15. if there is a sibling of *i* satisfying the condition specified below in (ii), remove *i* and its descendants;
end

In line 7, we check whether a node is already deleted. If it is the case, the corresponding computation will not be performed, leading to sometime reduction. In addition, some work in line 13 and 14 can also be saved. In line 15, we remove *i* if it can be removed according to the condition (ii) given below.

4.2 Conditions for Remove Redundant Nodes

The query Q can be minimized by doing the following conditions with each node $v \in Q$:

(i) Let $v_1, v_2, \ldots, v_k$ be the children of v;

(ii) For each v_i,

if (v,v_i) is a child edge and there exists v_j $(j \neq i)$ such that (v,v_j) is a child edge and $c_{vivj}=1$, then remove the subtree rooted at v_i if v_j has not been removed;

if (v,v_i) is a descendant edge and there exists v_j $(j \neq i)$ such that (v,v_i) is a child or descendant edge and $c_{vivj}=1$ or $a_{vivj}=1$, then remove the subtree rooted at v_i if vj has not been removed.

5. Experimental Results

In this section, we show and discuss the results obtained from five queries, as implemented in the query minimization algorithm. We show the performance of the algorithm in Table 1 with five queries.

As a result of the table, percentage for the performance of the query is depended on the number of the redundant nodes. When the number of redundant nodes is increased, the percentage for the performance of the query is decreased. And also the number of nodes is important role to calculate the percentage of the performance. Table 1 shows the performance of the query minimizing algorithm with five queries.

Table 1. Performance of the algorithm

	Query 1	Query 2	Query 3	Query 4	Query 5
Number of nodes	8	8	10	9	9
Number of redundant nodes	3	2	4	4	3
Number of remaining nodes	5	6	6	5	6
Percentage for the performance of the query (%)	63%	75%	60%	56%	67%

6. Conclusion

To answer the increasing demands on RDF repository, we carefully studied the existing RDF data management systems, identified the preferred properties of an RDF repository and proposed to take advantage of the latest XML data storage and efficient query processing techniques. We identified the system that serializing from RDF into RDF/XML and the mapping from RDF/XML to XML documents. In addition, our approach is efficient for time consuming in translation from RDF to XML documents for supporting Semantic Web applications in various domains.

References

[1] C. Weiss, *et al.* "Hexastore: sextuple indexing for semantic web data management", *PVLDB*, 1(1):1008–1019, 2008.

[2] D. J. Abadi, *et al.* "Scalable Semantic Web Data Management Using Vertical Partitioning", In *VLDB*, 2007.

[3] D. Beckett. "RDF/XML syntax specification (revised)", W3C Recommendation, February 2004.

[4] J. J. Carroll, *et al.* “Jena: implementing the semantic web recommendations” , 2004.
[5] L. Sidirourgos, *et al.* “ Column-store support for RDF data management: not all swans are white”, *PVLDB*, 1(2):1553–1563, 2008.
[6] M. Zhou and W. Yuqing. “XML-Based RDF Data Management for Efficient Query Processing”, 2010.
[7] Norman Walsh. “Rdf twig: accessing rdf graphs in xslt. *In Proc. Extreme Markup Languages*”, 2003.
[8] S. Alexaki, *et al.* “The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases”, 2001.
[9] S. Bischof, S. Decker, K. Thomas, N. Lopes, A. Polleres. “Mapping Between RDF and XML with XSPARQL”, April 2011.
[10] S. Battle.“Gloze: XML to RDF and back again”, http://www.hpl.hp.com/personal/steve-battle.
[11] T. Neumann, *et al.* “The RDF-3X engine for scalable management of RDF data”, *VLDB J.*, 19(1):91–113, 2010.
[12] Y. Chen and D. Chen. “Efficient Processing of XML Tree Pattern Queries”, 2006.