XISS/R: XML Indexing and Storage System Using RDBMS *

Philip J Harding   Quanzhong Li   Bongki Moon

Department of Computer Science, University of Arizona, Tucson, AZ 85721
{harding,lqz,bkmoon}@cs.arizona.edu

Abstract

We demonstrate the XISS/R system, an implementation of the XML Indexing and Storage System (XISS) on top of a relational database. The system is based on the XISS extended preorder numbering scheme, which captures the nesting structure of XML data and provides the opportunity for storage and query processing independent of the particular structure of the data. The system includes a web-based user interface, which enables stored documents to be queried via XPath. The user interface utilizes the XPath Query Engine, which automatically translates XPath queries into efficient SQL statements.

1 Introduction

XML is quickly becoming the new standard for data representation and exchange in the Internet. An emerging issue is how to provide efficient storage and manipulation of XML data [1, 2, 3, 5]. Since relational technology is mature and well-developed, using relational databases to store XML data is an important direction to explore. The XISS/R system demonstrates an efficient approach for using relational database systems to store and evaluate queries on XML data.

The *extended preorder numbering scheme* [4] is an efficient method for relating semi-structured XML data with structured relational data. The numbering scheme encodes the nesting structures of XML data in such a way that ancestor-descendant relationships can be determined in constant time. Every XML node encoded in this way can be stored in a uniform manner inside a relational database.

The ability to quickly determine ancestor-descendant relationships is crucial to efficient query processing when dealing with regular path expressions. With the numbering scheme, translating regular path expressions to SQL statements is straightforward, without need for recursive SQL queries.

In the XISS/R system, we implement the XML Indexing and Storage System on top of a commercial relational database system. Several key issues involved in storing the XML data using the numbering scheme are identified and relational schemas are generated based upon the choices made in resolving these issues.

The features of the XISS/R system include

- A web-based user interface, which enables stored documents to be queried via XPath.
- An XPath Query Engine, which automatically translates XPath queries into efficient SQL statements.
- Multiple relational schemas for comparison.
- Reporting of performance statistics.

2 System Description

The XISS/R system consists of three components:

1. A mapping of XML data to relational schema.
2. An *XPath Query Engine*.
3. A web-based user interface.

The mapping of XML data to relational schema is accomplished by using the extended preorder numbering scheme. We have generated two relational schemas that make best use of this numbering scheme.

The XPath Query Engine allows XPath queries to be issued on the relational implementation of the mapping of XML data.

Universal access to the system is provided through a web-based interface which allows users to visually interact with the query engine and result set.

2.1 Mapping XML Data to Relational Schemas

Mature relational technology can be a useful mechanism for storing and querying XML data. Mapping of semi-structured XML data to a highly structured relational system can be accomplished by using the extended preorder

Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003

Figure 1: Tables in Schema A (Primary keys in bold)

numbering scheme. This numbering scheme provides a method for encoding tree-formed data into integer pairs irrespective of data content.

2.1.1 The Extended Preorder Numbering Scheme

The extended preorder numbering scheme [4] associates each node in an XML document with a pair of numbers, the *extended preorder* and the *range of descendants* ($<order, size>$), which should satisfy the following conditions:

- For a tree node y and its parent x, $order(x) < order(y)$ and $order(y) + size(y) \leq order(x) + size(x)$. In other words, interval $[order(y), order(y) + size(y)]$ is contained in interval $[order(x), order(x) + size(x)]$.
- For two sibling nodes x and y, if x is the predecessor of y in preorder traversal, $order(x) + size(x) < order(y)$.

For a tree node x, $size(x)$ can be an arbitrary integer larger than the total number of the current descendants of x. This allows future insertions to be accommodated gracefully. The ancestor-descendant relationship can be determined in constant time by examining these pairs of numbers. That is, for two given nodes x and y of a tree T, x is an ancestor of y if and only if $order(x) < order(y) \leq order(x) + size(x)$.

In a relational schema, these pairs can be stored in conjunction with other node information and used as part of join conditions during query processing. Using this numbering scheme, it is not necessary to attempt to use SQL statements to traverse tree structures in order to to process ancestor-descendant joins. This numbering scheme also enables the XISS/R system to store nodes in a uniform format as tuples inside relational tables without losing XML document structural information.

2.1.2 Relational Schema

The numbering scheme provides a unified way to store the structural relationships of XML data. However, there are a number of options for storing other necessary data from XML documents alongside such structure data. We investigate several key issues that can affect the storage and query performance:

- How to store element and attribute nodes.
- How to store tag name values.
- How to store value string information for text and attribute nodes.

Figure 2: Tables in Schema B (Primary keys in bold)

- For different schemas, what kind of indexes are needed.

XISS/R requires five pieces of information for each node stored in the system. They are the document ID, order (also referred to as the node ID) and size of a node in the numbering scheme, depth of a node in a document tree, tag-name and text value of a node. In an effort to improve efficiency in processing queries and during export of data, we also store the parent node ID, sibling node ID, first child ID and first attribute ID for each node.

Utilizing the above information we have created two relational schemas, *Schema A* and *Schema B*, that are best suited to implementing XISS/R.

Schema A

XISS/R divides nodes into three categories, element, attribute and text. Since the set of information that each type of node requires to be described is different, this separation saves space by storing only necessary fields. Schema A, which is shown in Figure 1, separates nodes along these lines and is defined as follows:

1. The *Document Table* consists of the Name of a document and a unique numerical Document_ID .
2. The *Element Table* stores all element nodes.
3. The *Attribute Table* stores attribute nodes. The Value stores the attribute value.
4. The *Text Table* stores text nodes (not text values) within the system. Value stores the actual text.

In this schema, a Document Table is a simple way to separate the document name from the element, attribute, and text relations. The element, attribute and text relations store a reference to the numerical ID of the document for each node. In the Element, Attribute, and Text tables, Order and Document_ID uniquely identify any node within the system. Since all attribute nodes have a corresponding text value (or empty) string, this value is stored with the attribute node, further reducing query time.

Schema B

Schema B, which is shown in Figure 2, goes further than Schema A in separating nodes into different tables. Like Schema A, Schema B separates nodes by their type (element, attribute or text). In addition to this, Schema B horizontally partitions element and attribute by tag-name. An element or attribute table is created for each unique tag-name. All nodes of the same type, from all stored documents, with the same tag name are stored in the same table. For instance, all element nodes with the tag-name `ACT` are stored in one table. This strategy serves to reduce overall query time by making SQL statements simpler and over smaller tables than Schema A. Schema B also contains a table that has information for reconstruction of XML documents.

2.2 XPath Query Engine

The XPath Query Engine (Figure 3), which is located between the user interface and the database server, is the core component of the XISS/R system. The query engine accepts XPath queries and generates SQL statements to send to the database server. Query results from the database server are then formated and forwarded to the user interface by the query engine.

Upon receiving an XPath query, the *Query Parser* first parses the query and translates it into an intermediate tree-structured format. For example, the query `A[B]/C` will be translated into a tree with node `A` as the root, and `B` and `C` as two children of `A`.

This tree-structured intermediary is then simplified by the *Query Transformer* based on current cache information in the *Cache Directory*. The *Cache Directory* stores result information from previous queries. Each entry in the directory consists of a query pattern in tree-structured intermediate format, and a pointer to it's associated result set.

The *Query Transformer* matches the current query tree against patterns in the cache directory. If a match is found, the matched part in the query tree will be replaced by a super node. This super node is similar to a normal node except that when parsed into SQL, the *SQL Generator* will evaluate it based on cached results. For example, if the result of the query `A[B]` is in the cache, the subtree representing `A[B]` can be replaced by a super node representing cached results. The query is thus simplified into a two-node query tree. In addition, other optimizations can be implemented inside the *Query Transformer*.

The simplified query tree is translated into SQL statements by the *SQL Translator* and communicated to the database server. For example, the query `media[@media-type="image"]` will be translated into the following SQL statement when using Schema A:

```
SELECT
    et0.DID as Document, et0.NID as Node_ID
FROM
```
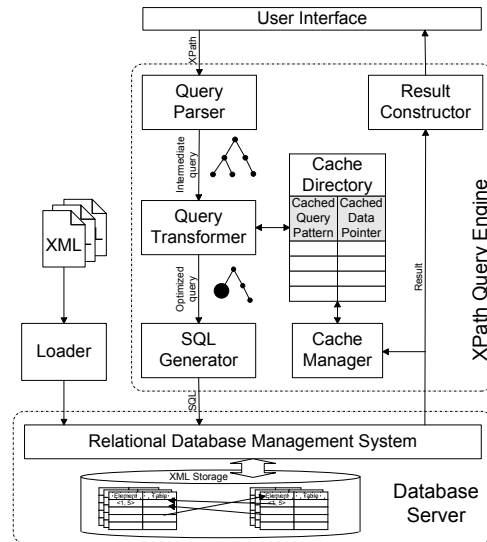

Figure 3: XISS/R System Architecture

```
    elem_tab et0, attr_tab at0
WHERE
    et0.NAME = 'media' and
    at0.NAME = 'media-type' and
    at0.VALUE = 'image' and
    at0.DID = et0.DID and
    at0.PARENT_ID = et0.NID
```

The database server processes the SQL query and returns the result to the XPath Query Engine, whose *Result Constructor* formats the result and sends it to the user interface for display. The result is also sent to the *Cache Manager*, which makes a caching decision based on information such as result set and data size, query pattern or sub-pattern frequency, query processing time, etc.

2.3 Web-Based User Interface

The Web-Based Interface allows users to issue XPath queries to the XISS/R system from any Internet terminal via `http://xiss.cs.arizona.edu`. It accepts queries through the HTML form construct, sends these queries to the XPath query engine and receives results in return. Users can choose the amount of results displayed at one time and then page through the result set. In addition to the requested portion of the result set, the web query interface also returns usage statistics to users such as the XPath query, the translated SQL query(ies), execution time for each query, and total time spent server side.

An example of the query web interface is shown in Figure 4. Users can choose the dataset and schema to query. The XPath query and the desired number of results per page are sent to the web server, which interacts with the XPath query engine to process the query. The appropriate portion of the result set, received from the web server is displayed in an additional frame. Additional queries can be issued at any time.
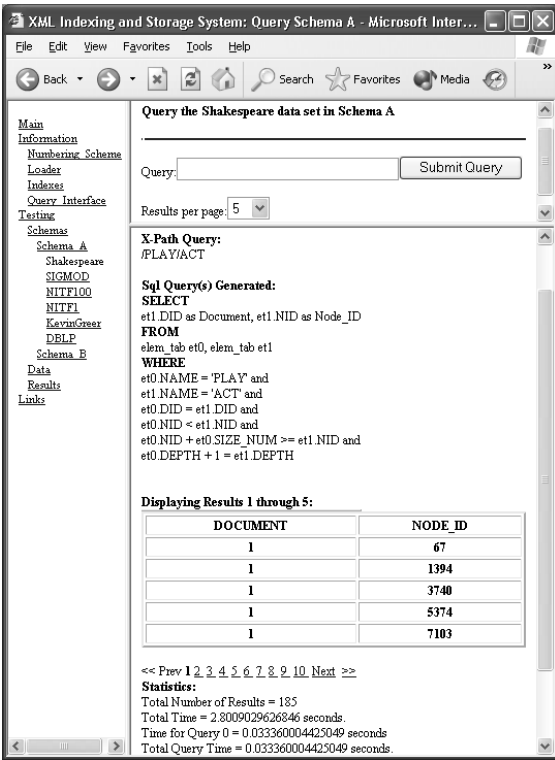
Figure 4: XISS Query Interface Example

2.4 Implementation

Our current implementation of XISS/R uses MySQL but has the capability to use any commercial database system. Documents are parsed and loaded into the database by a program that uses MySQL's C interface. This loader uses the LibXML [6] library to access XML documents such that their structural information can be encoded with the extended preorder numbering scheme.

The web-based interface is implemented with Apache 1.3.24 compiled with PHP 4.1.12. The XPath Query Engine is implemented by PHP scripts that communicate with the RDBMS in use.

Currently the web interface accepts queries expressed in XPATH 2.0 Abbreviated syntax. The XPath operators, $a//b$, $a[b]$, a/b, $a[@b = "c"]$ and $a@b$ are supported, and can be combined as needed to address stored XML documents. As per the XPath specifications result sets are composed of unique pointers to addressed nodes.

We created several schemas to determine whether to store elements in a large node table or store them in separate node tables divided by node name. The overall trend in performance was that schemas using horizontal partitioning were faster. As for whether to store tag names in the node tables or store them in a separate tag name table, we found that since the number of distinct elements and attributes is usually small, the join time between a tag name table and node tables is inconsequential compared to the common total query time.

Unlike a tag name, an XML dataset can and often contains large amounts of distinct value-string information. In worst case scenario, this can be "$2 \times (the\ number\ of\ element\ nodes) - 1 + the\ number\ of\ attribute\ nodes$". With this high percentage of distinct textual values, the amount of space saved is trivial compared to the extra time it takes to access this data. For this reason, the value string information is stored directly in the attribute and text tables.

In addition to the tables in the schemas described before, we also utilized database indexes to accelerate query processing. There are B+-tree indexes on all the primary keys of tables, name, value and document text information, `Parent_ID` and `Document_ID` of all text nodes and `Order`, `Size` and `Document_ID` for all nodes.

3 Demonstration

In the XISS/R demonstration, an Apache Web server and a MySQL database server will be available to serve requests from the web user interface. Query processing is provided on Schema A and B with several datasets. We will demonstrate our implementation of the XISS/R system and justify the choices we made in designing the system. The audience will be able to interact with the system during the demonstration. Users can choose different query combinations to highlight performance differences between different relational schema. In addition, performance figures and statistics for large scale testing will be exhibited to show the advantages of the XISS/R system.

References

[1] Philip Bohannon, Juliana Freire, Prasan Roy, and Jrme Simon. From xml schema to relations: A cost-based approach to xml storage. In *Proceedings of the 18th Inter. Conference on Data Engineering*, San Jose, California, February 2002.

[2] Alin Deutsch, Mary Fernandez, and Dan Suciu. Storing semistructured data with STORED. In *Proceedings of the 1999 ACM-SIGMOD Conference*, pages 431–442, Philadelphia, PA, June 1999.

[3] Daniela Florescu and Donald Kossmann. Storing and querying xml data using an rdmbs. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[4] Quanzhong Li and Bongki Moon. Indexing and querying xml data for regular path expressions. In *Proceedings of the 27th VLDB Conference*, Rome, Italy, September 2001.

[5] Igor Tatarinov, Stratis Viglas, Kevin S. Beyer, Jayavel Shanmugasundaram, Eugene J. Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *Proceedings of the 2002 ACM-SIGMOD Conference*, Madison, Wisconsin, USA, Jun 2002.

[6] XMLsoft. The XML C library for Gnome. http://xmlsoft.org/, March 2003.