S-XML 데이터의 효율적인 X-Path 처리를 위한 색인 구조

장기*, 장용일*, 박순영*, 오영환**, 배해영*
*인하대학교 킴퓨터•정보공학과
**나사렛대학교 정보통신학과
e-mail : yulinzq@dblab.inha.ac.kr

An Index Structure for Efficient X-Path Processing on S-XML Data

Qi-Zhang*, Yong-Il Jang*, Soon-Young Park*, Young-Hwan Oh and Hae-Young Bae***
***Dept. of Computer Science & Information Engineering, Inha University**
****School of Information Science, Korea Nazarene University**

Abstract

This paper proposes an index structure which is used to process X-Path on S-XML data. There are many previous index structures based on tree structure for X-Path processing. Because of general tree index's top-down query fashion, the unnecessary node traversal makes heavy access and decreases the query processing performance. And both of the two query types for X-Path called single-path query and branching query need to be supported in proposed index structure.

This method uses a combination of path summary and the node indexing. First, it manages hashing on hierarchy elements which are presented in tag in S-XML. Second, array blocks named path summary array is created in each node of hashing to store the path information. The X-Path processing finds the tag element using hashing and checks array blocks in each node to determine the path of query's result. Based on this structure, it supports both single-path query and branching path query and improves the X-Path processing performance.

1. Introduction

The Extensible Markup Language (XML) is the standard for data storing and exchanging in the internet and has the growing popularity nowadays [1]. GML is the XML encoding for the transport and storage of spatial information [2]. And, S-XML is the extended GML for LBS applications [3]. There are several query languages proposed for XML, such as X-Path and X-Query which are the standard query languages [4, 5]. As X-Path being the common feature of XML query languages, indexing XML data for X-Path processing is concerned to be a challenge research [6]. Many conv-entional methods using tree index structure were proposed in this research filed [6, 7, 8]. However, considering the large query requirements of spatial information on S-XML data, because of general tree index's top-down query fashion, the unnecessary node traversal makes heavy access and decreases the query performance. And both of single-path query and node indexing need to be supported in proposed index structure.

The proposed method in this paper uses an index structure which is used to process X-Path on S-XML data. This method uses a combination of path summary and the node indexing [6, 7, 8]. First, it manages hashing on hierarchy elements which is presented in tag in S-XML data. Second, array blocks are created in each node of hashing to store the path information. Each node includes element's name, path summary array blocks and a pointer which points to the parent element. The X-Path processing finds the tag element using hashing and checks array blocks in each node to determine the path of query's result. Based on this structure, it supports both single-path query and branching path query.

The paper is organized as follows. Section 2 introduces the related work about XML query processing and discusses two previous approaches called path summary and node indexing. Section 3 presents the algorithm of hashing XML data for X-Path processing and gives the querying processing method for X-Path using this index structure. The conclusion and future work are given in the last section.

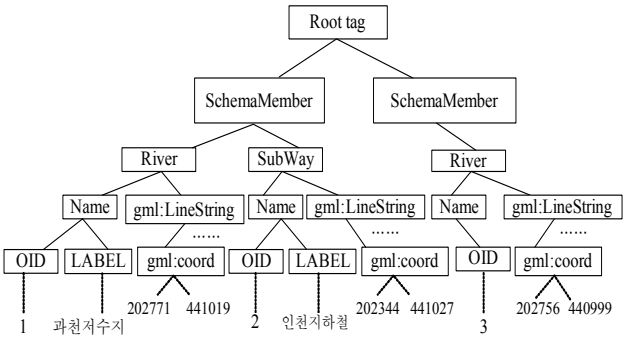
2. Related Work

This section will introduce the XML data model and the query processing, and then discuss two previous index approaches called path summary and node indexing related in our research.

* This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment)

2.1 XML data model and Query processing

XML data is generally modeled by a tree structure [8, 9]. The nodes of tree represent the tag elements, attributes and text data [8]. The nesting relationship in XML data is represented by the parent-child node pairs. (Figure 1) indicates a sample S-XML data which is modeled by a tree structure. To quickly determine the ancestor-descendant relationship between any pair of nodes in the hierarchy of XML data is the crucial work speeding up the X-Path processing [6].



(Figure 1) A sample S-XML data

X-Path is the standard common feature of XML query languages. Furthermore, there are two main queries in X-Path which is called single-path query and branching query [8]. Take an example of the S-XML data in (Figure 1), single-path query has the format like */SchemaMemeber/River/Name/LABEL(Q1)*. It means finding all of the *LABEL* elements which are under the path of */SchemaMemeber/River/Name*. The branching query is the query which has more conditions, for example, for the query "*find River under SchemaMember with both LABEL and LineString*", i.e. */SchemaMember/River [Name/LABEL and LineString](Q2)*

2.2 Path summary and Node Indexing for X-Path processing

Many previous approaches concern the research of indexing X-Path. They can be classified into two categories which are Path indexing and Node indexing. Path indexing creates a path summary from XML data [7, 8]. For the single-path queries, it speeds up the evaluation of queries. However, expensive join operations are needed for processing queries with multiple branches in path summary. Also the path summary is unable to answer branching queries. Node indexing creates indexes on each node by its positional information within an XML data tree [6]. Because of using node as a basic query unit, it provides great query flexibility. Both of these two approaches use the tree based structure. Because of the disadvantage of the tree index's top-down query fashion, we propose an improved solution based on hashing S-XML combining both of the path summary and node indexing.

3. Hashing S-XML Data for X-Path Processing

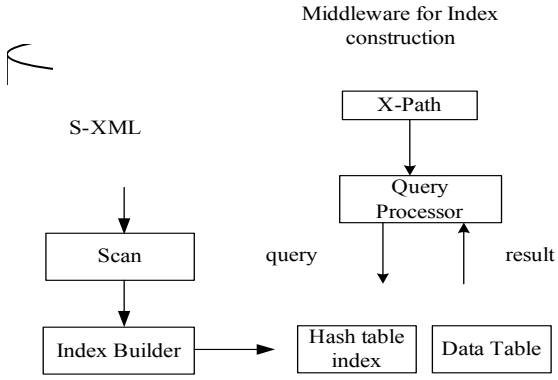
In this section, we first present a framework for building Hashing S-XML data. Then briefly discuss the proposed index structure and the method for processing X-Path.

3.1 Framework of Hashing S-XML data

S-XML data contains not only tag elements (e.g. SchemaMember, River) which purpose the different hierarchies, but also heterogeneous values (e.g.OID, LABEL).

(Figure 2) shows the system framework of Hashing S-XML data. Hashing S-XML data can be built in three steps.

a. The *Scan* module collects the structure and value characteristics from an S-XML file.
b. The *Index Builder* constructs the index structure for the elements and data value in S-XML, based on the algorithm of index configuration.
c. X-Path query is inputted and processed through the *Query Processor*. After querying on the index structure, the middleware returns the query result to the user.



(Figure 2) System framework of Hashing S-XML Data

3.2 Building Hashing S-XML data and Data organization

This section introduces the node structure of the proposed index, and then gives the algorithm of building hashing S-XML data.

3.2.1 Node structure

We distinguish two types of elements in S-XML data: *structure elements*, consisting of tag elements which dedicate the structure of S-XML data and *value elements*, consisting of text node. And they are modeled in different ways in the node structure.

There are two relationships between the elements in S-XML data, i.e. parent-child relationship and the ancestor-descendant relationship. The parent-child relationship is the relationship in which one element includes another element directly. On the other hand, the ancestor-descendant relationship indicates the relationship between the descendant elements in any hierarchy and the ancestor element. Take the S-XML data in (Figure 1) for example, the element *Name* and *OID* is in parent-child relationship and *River* and *OID* has the ancestor-descendant relationship. To benefit from these two relationships in X-Path, the index structure should include the information with two relationships. In this proposed index structure, we express these two relationships in the node structure.

(Figure 3) shows the node structure of structure element. It includes three parts, name of element, parent element pointer and the path summary blocks. Parent element pointer points to the parent element in the bucket and indicates the

parent-child relationship. Path summary blocks include the information of the path which each element traverses. The path summary blocks show the relationship between the ancestor and the descendant. Every element has the unique path summary blocks.

Element number	Parent element pointer	Path Summary blocks

(Figure 3) Node Structure of structure element

There is a difference between the *structure elements* and *value elements* in node structure which can be seen in (Figure 4). The *value elements* use a pointer pointed to value table instead of path summary blocks in the node. Because the *value elements* only include the text value, no path is included in the element. Since the value data in S-XML can be modeled in relational tables so that it can be implemented in relational database. For the various value types in S-XML data, multiple value indices can be implemented based on the value table. Because of large amount of spatial data in the S-XML, several general indexing for spatial data can be used to fast the query performance such as R-tree.

Element number	Parent element pointer	Value table pointer

(Figure 4) Node Structure of value element

In the *Scan* step, the information of relationship between elements is collected and the path summary of each element is built. The node is built in *Index Builder* using the node structure above during the index's building time.

3.2.2 Building Hashing S-XML data

The proposed index structure hash different tag elements in hash bucket, i.e. *structure elements* and *value elements*. The hash function H(m) is implemented to compute the bucket value for the same tag elements. Same tag elements are put in the hash bucket with the same bucket value. A link list is built to link all the element nodes in the same bucket.

(Figure 5) gives the algorithm of the index structure building processing.

```
Algorithm: Building the Index Structure

Input: {E1, ..., Em}, Ei is the element in the S -XML data
       H(m), the hash function
IndexBuilding({E1, ..., Em}, H(m))
  For each Ei in the S -XML data ..... ①
      use H(m) to compute the hash value h
   //elements with same name have the same h
  For each Ei, do NodeBuilding(Ei) ..... ②
      If Ei is the structure element
          Store path summary in array blocks
          and parent pointer
      Else Ei is the value element
          Store value in value table
      End If
   Input the node in bucket using hash value h
   as bucket number ..... ③
End
```

(Figure 5) Algorithm of Index Building

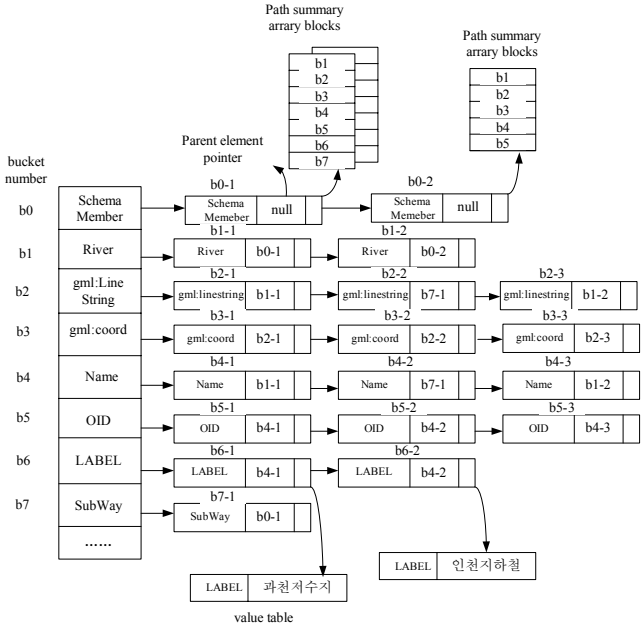
Take the S-XML data in (Figure 1) for example, the procedure of building index structure is explained as below:

① From the root tag of the S-XML data, scan the tag element, use the hash function H(m) to transform the variable-size tag element to the hash value *h*. Same tag elements have the same hash value. And use this hash value *h* as the bucket number.

② Meanwhile, collect the information of the tag element including the path summary which it traverses and the parent element. And build the node for tag element in different ways of structure element and value element as discussed before. Store the path summary in path summary blocks for structure element, otherwise store the text value in value table and make a pointer pointed to value table.

③ Input the node in the bucket it belongs to. And add it to the link list of the same bucket.

Follow the above steps for the next element, until the ending symbol of the root tag element is reached. (Figure 6) indicates the index structure of Hashing S-XML data.



(Figure 6) Index structure of Hashing S-XML Data

3.3 X-Path processing using Hashing S-XML data

Concerning the two main queries discussed in section 2, we explain the X-Path query processing based on those two query types below.

3.3.1 X-Path query modeling

Since the X-Path query is expressed in text, the first crucial work is query modeling which means to model the X-Path query in relational table. After this step, each element in the X-Path query is corresponded with an unique key QID (query ID) which is computed by hash function H(m) same as the bucket number. And then the QID is used in query table as the searching key in the proposed index structure to search the result.

(Figure 7) indicates the modeled query tables for single-path query and branching query separately. Taking the S-

XML data in (Figure 1) for example, Q1: "*find all the LABEL elements which is under the path of /SchemaMemeber/River /Name*" is represented as */SchemaMemeber/River/Name /LABEL*, which is modeled to *b0/b1/b4/b6* for searching. Q3:"*find all the LABEL elements without considering the path*" can be expressed in *//LABEL*, also is modeled to *//b6*. The branching query Q2: */SchemaMember/River[Name/LABEL and Line-String]* is modeled to */b0/b1[b4/b6 and b2]*.

QID	Element Name
b0	SchemaMember
b1	River
b4	Name
b6	LABEL

/SchemaMemeber/River/Name/LABEL
(a)

QID	Element Name
b0	SchemaMember
b1	River
b4	Name
b6	LABEL
b2	gml:LineString

/SchemaMemeber/River/[/Name/LABEL and gml:LineString]
(b)

(Figure 7) X-Path query tables of (a) single-path query and (b) branching query

3.3.2 X-Path Processing using Hashing S-XML data

After the X-Path is transformed into a query table, we evaluate it using the proposed index structure in three steps. As the procedures of single-path query and branching query are almost the same, the single-path query Q1 is taken for example.

First, find the location on the hashing bucket. From the first element in query table, search the matching bucket number that equals to the QID. For Q1, the element *SchemaMember* is considered at first, the bucket which is assigned in *b0* is returned because it equals to the QID (b0).

Second, visit each node in the bucket *b0*. And check the path summary array blocks in each node to determine whether this node includes the path which is satisfied to the query result. In the first node in bucket *b0*, the path summary includes the list (b1, b2, b3, b4, b5, b6, b7), which includes all the QID in Q1's query table. It is sure that this node is the ancestor node element which includes the path of query result. Meanwhile, remember this ancestor element node's bucket number. Continue to find the next bucket corresponding to the next QID. In the next node, the parent pointer should be checked and the node whose parent pointer pointed to the ancestor element node also includes the path of query result. Based on this step, the node *b0-1* is followed and *b0-2* is discarded visiting because its path summary array has no bucket number *b6*. And in bucket *b1*, *b1-1* whose parent node is *b0-1* is returned.

Third, continue the steps 1 and 2 until the last element in query table is found and returns the query result.

The proposed Hashing S-XML data based query processing algorithm has the following advantages:

a. It combines both node indexing and path summary. For node indexing, it creates hash index on each node by its positional information. In that case, different tag elements can be located in their own bucket independent with its parent node. That is more flexible than tree-based index structure's top-down query fashion. And, it provides parent pointer and array blocks in each node for the path summary. Because of that, it supports both of single-path query and branching query.

b. Storing different elements in different buckets, it is easy to differentiate heterogeneous S-XML data value. That means each bucket has the same data value type. When user wants to find particular type of data, it locates to the corresponding bucket directly. This is the characteristic that tree-based index structures never support.

4. Conclusion and Future Work

This paper proposed a hashing based index structure for S-XML data to process the X-Path query. The algorithm of building hashing S-XML data is proposed. Then the processing method of X-Path query based on hashing S-XML data is described. The proposed index structure supports both of single-path query and branching query and improves the X-Path query performance.

Also, the dataset and various query types need to be made for performance evaluation to validate the efficiency of our proposed index structure compared with the previous research.

References

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, François Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)," W3C Recommendation, http://www.w3.org/TR/REC-xml/, 2004.

[2] Simon Cox, Paul Daisey, Ron Lake, Clemens Portele, Arliss Whiteside, "OpenGIS Geography Markup Language (GML) Implementation Specification," OGC, Document Number 02-023r4, 2003.

[3] Young-Su An, "S-XQuery: The Satial Query Language of S-XML Management System," Master thesis, Dept. of Computer Science & Information Engineering, Inha University, 2004.

[4] James Clark, Steve DeRose, "XML Path Language (XPath) Version 1.0," W3C Recommendation, http://www.w3.org/TR/xpath,1999.

[5] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, "XQuery 1.0: An XML Query Language,"W3C, http://www.w3.org/TR/2005/WD-xquery-20050211/, Working Draft, 2005.

[6] Quanzhong Li, Bongki Moon, "Indexing and Querying XML Data for Regular Path Expressions," VLDB, 2001.

[7] R.Goldman and J.Widom. Dataguides "Enabling Query Formulation and Optimization in Semistructured Databases,"VLDB, 1997.

[8] Qinghua Zou, Shaorong Liu, Wesley W. Chu, "Ctree: A Compact Tree for Indexing XML Data,"WIDM, 2004.

[9] Stephen Mohr, Steven Livingstone, Darshan Singh, Danny Ayers, Michael Coming, "XML Application Development with MSXML 4.0,"Wrox Press Ltd, 2002.