Bridging XML and Relational Databases: An Effective Mapping Scheme based on Persistent

Samini Subramaniam, Su-Cheng Haw, Poo Kuan Hoong
Faculty of Information Technology, Multimedia University, Malaysia

Article Info

Article history:

Received Dec 14th, 2011
Revised Mar 1st, 2012
Accepted Mar26th, 2012

Keyword:

Labeling schemes
Mapping scheme
Persistent labeling
Query processing
Relational databases

ABSTRACT

XML has emerged as the leading medium for data transfer over the World Wide Web. At the present days, relational database is still widely used as the back-end database in most organizations. Since there is mismatch in these two structures, an effective mapping scheme is definitely essential that provides seamless integration with relational databases. On the other hand, an immutable labeling scheme is certainly significant to dentify the XML nodes uniquely as well as supports dynamic update without having the existing labels to be re-labeled when there is an occurance of dynamic update. As such, in this paper, we propose s-XML by adopting the Persistent Labeling scheme as the annotation scheme to ensure seamless integration with relational database and able to support updates without the need to re-construct the existing labels. We conduct experiments to show that s-XML performs better in terms of mapping the XML nodes to relational databases, query retrieval and dynamic update compared to the existing approaches.

Corresponding Author:

Samini Subramaniam,
Faculty of Information Technology,
Multimedia University,
63100 Cyberjaya, Malaysia
Email: ts42003@yahoo.com

1. INTRODUCTION

XML has emerged as a generic markup language for documents as well as the *de facto* standard for data exchange over the World Wide Web. There are many different types of XML data found in today's document repositories, digital libraries and on the web, which range from simple flat text with little meaningful structure to be queried to over truly semistructured data with a rich and often irregular structure. For example, a business can easily model complex structures such as invoice, orders and inventory system in XML format. In addition, there is hundreds of XML schema defined to encode data into XML format for specific application domains.

On the other hand, relational database drive most businesses of any size today. Nevertheless, relational database cannot meet all the demands of electronic business because it process data independently of the context. In other words, relational database is simply not a good match for semi-structured content represented in XML. However, since enterprises have invested trillions of dollars in relational database, they would be much reluctant to simply replace relational database with a pure XML store.

Due to the demand for storing and querying XML data, especially for data exchange, a mapper to store and retrieve XML (a tree structure) via relational database (tables with rows and columns) and vice-versa is definitely essential. Since there are mismatches between the XML-structured data and relational data, mapping plays an important role in providing seamless integration between these database infrastructures.

There are four basic relationships that a good mapping approach needs to cater for; the ancestor-descendant, parent-child, sibling and level relationships. These information are known as structural relationship need to be stored in the relational tables to identify the connection between the nodes in the XML document. This enables the user's queries to be processed competently.

The dilemma that has been enduring for sometime is to come up with a mapping scheme that can preserve basic relationships among the nodes for proficient XML processing. Basically, there are two types of user queries, which are full-text query and structural query. Many existing approaches supports full-text query but be oblivious to the structural one which results in inconsistencies in query retrieval and incapable to furnish any query with the combinations of multiple criteria.

Apart from the support for both types of queries, a good labeling scheme must be able to support dynamic updates. Dynamic update refers to the updating process (insertion of new node(s) and deletion of existing node(s)) to the original XML data source. A good labeling method should generate immutable labels that does not require modification during the occurance of dynamic update.

The rest of the paper is organized as follows. Section 2 illustrates some review on the existing mapping schemes. Section 3 describes the new mapping scheme, s-XML. Section 4 explains the experimental design, experimental results and discussions. Finally, Section 5 concludes the paper.

2. RELATED WORK

There are many mapping techniques such as the Relational DTD approach [13], the Edge approach [3] and the Attribute approach [8]. The Relational DTD approach [13] maps XML data based on the frequency of the element occurrence in an XML document. The elimination of less important elements and grouping of elements based on incidence allows lesser space consumed, straightforward table schema and efficient mapping to tables. However, this approach can only be used if the Data Type Definition (DTD) or XML schema of an XML document is available. In Edge approach [3], XML document is shred into a single relational table. As such, this approach may suffers from excessive table size error and multiple self-joins may be require for query retrieval. Attribute approach [8] creates as much table as distinct element name that appear in the document into different relational tables. This is one of the drawbacks of the Attribute approach where the number of tables depends on the distinct element names in XML document.

An automatic mapping technique was proposed [7] from an XML document to relational databases especially the nested structure of the XML documents is preserved. Association inlining was proposed [12], a new inlining method, for mapping DTD to relational tables by improving their earlier versions of inlining methods, i.e., Shared inlining and Hybrid inlining to reduce fragments and excessive joins. A lossless schema mapping algorithm was proposed [1] to generate a database schema from a DTD, which makes several improvements over existing algorithms. In addition, they also proposed two linear data mapping algorithms based on Document Object Model (DOM) and Simple API for XML (SAX), respectively, to map ordered XML data into relational data. Nevertheless, these mapping techniques are unable to support dynamic update, an important feature to support ever-changing environment because of the limitation of the labeling scheme in terms of persistency.

A good labeling scheme is certainly needed to ensure that the labels generated to uniquely identify XML nodes are immutable at any point of time; to be exact during dynamic update. Dynamic update comprises of updating processes (insertion of new node(s), deletion of existing node(s) or any kind of updating processes) which happen at any point of time and require the existing labels to be maintained while generating new labels for the new nodes. Several labeling scheme [11] [14] [5] [9] have been proposed which can be broadly classified into four main categories; namely, Subtree, Prefix-based, Multiplicative and Hybrid [6]. Nevertheless, not many existing labeling schemes support dynamic update especially in situation where a massive updates are required. Yet, we observed that under heavy update, prefix-based scheme may not need to be re-generated. As such, we adopt the Persistent Labeling (one of the prefix-based scheme) as the labeling scheme in our propose mapping technique. In order to show the feasibility of our mapping approach, we evaluate the (1) query response time needed to retrieve a set of queries, (2) time required for insertion, and (3) time required for deletion against the existing techniques.

3. s-XML: OUR PROPOSED APPROACH

4.1. Background of Persistent Labeling Scheme

In XML, there are four main hierarchical relationships namely parent-child, ancestor-descendant, sibling and level. A compact and robust labeling scheme is essential to allow quick determination of these relationships between pair of nodes. In Persistent Labeling [4], each node is labeled as (l,[n_p,d_p],[n,d]), where l is the level of the node in the tree, [n,d] is the local label of the node, [np,dp] is the local label of the parent node. Parent label of a node is the self label of the parent node.

Figure 1 explains how the [*n*,*d*] and [n_p,d_p] is assigned.

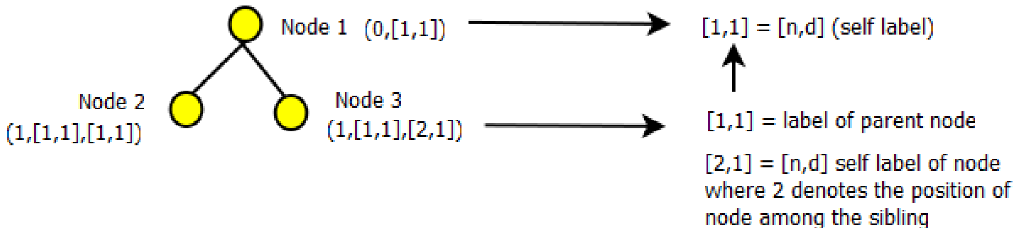

Node 1 (0,[1,1])
[1,1] = [n,d] (self label)
Node 2 (1,[1,1],[1,1])
Node 3 (1,[1,1],[2,1])
[1,1] = label of parent node
[2,1] = [n,d] self label of node where 2 denotes the position of node among the sibling

Figure 1. Explanation for [n,d] and [n_p,d_p]

Figure 2 shows an example of labeling scheme based on Persistent Labeling [4]. The root element will be labeled as (0,[1,1]) where 0 represents the level and [1,1] represents the local label of the node. This element does not have a parent label since the node is the origin of the XML tree. Next, the label for 'book' is (1,[1,1],[1,1]). i.e., the 'book' reside in level 1, with the parent node labeled as [1,1] and 'book' is the first child [1,1] among its sibling. Let us take another example. The 'publisher' is annotated with the label of (2, [1,1],[5,1]), where 2 indicates that 'publisher' is in the level 2, [1,1] denotes the parent's label of 'publisher' (which in this case the local label for 'book'), [5,1] is the ordinal occurrence of 'publisher' among its sibling ('publisher' is the 5th child of 'book').

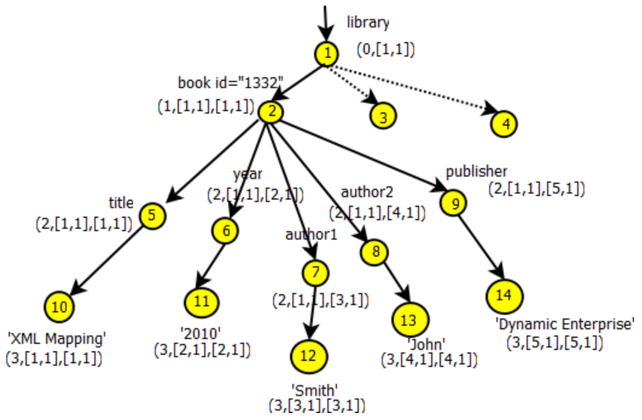


Figure 2. The labeling scheme adopted from Persistent Labeling

In terms of support for the new inserted node, new labels will be generated based on the following rules [4]. Let C be a node to be inserted, while Node A and Node B are the sibling of Node C.

a) Node C (ci,cj) is inserted before Node A provided that no nodes before Node A. Label C = (ai -1,aj) (see Figure 3(a)).
b) Node C is inserted after Node B provided that no nodes after Node B. Label C = (ci,cj) = (bi + 1,bj) (see Figure 3(b)).
c) Node C is inserted between Node A and Node B. Label C = (ci,cj) can be computed as follows: ci = bi. aj + ai. bj / d; cj = 2. aj. bj / d; where d is Highest Common Factor for (bi.aj + ai.bj) and (2.aj.bj) (see Figure 3(c)).

Based on the beautiful features of persistent labeling such as supports for the four hierarchical relationships and the support for dynamic update, we adopt the labeling scheme in our approach.

4.2. s-XML Table Schema

In s-XML, there are two tables namely the ParentTable and the ChildTable. All nodes in the XML will be shred into the two tables. The ParentTable stores all the internal nodes (annotated based on Persistent Labeling elaborated earlier in Section 3.1), while the ChildTable maintains leaf nodes information. The schemas of the tables are ellaborated as below.

ParentTable *(IdNode, pName, cName, Level, LParent, SelfLabel)* where:

a) **IdNode** - uniquely identify the nodes stored in the ParentTable (assigned based on breath-first traversal).

b) **pName**-stores parent node name.
c) **cName**-maintains child name.
d) **Level**-maintains level information
e) **LParent** – maintains the parent label of the node which stores the reference of the parent label (IdNode).
f) **SelfLabel** - maintains the self-label or local label of the node which is [n,d] in Persistent Labeling.

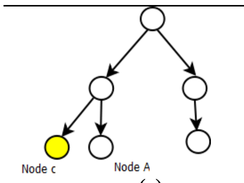
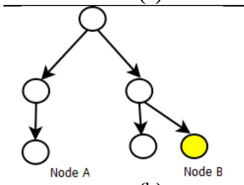
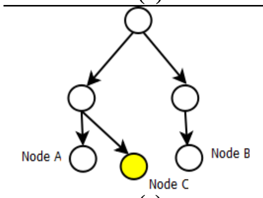
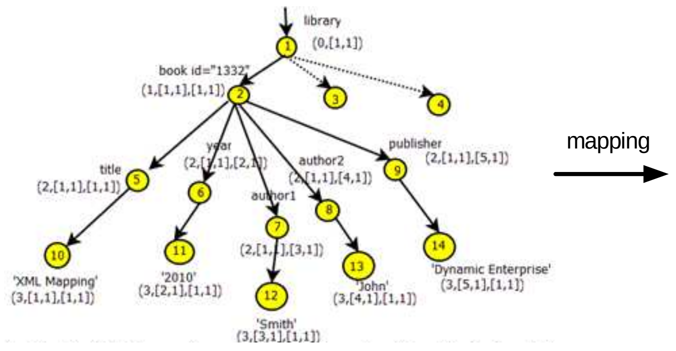
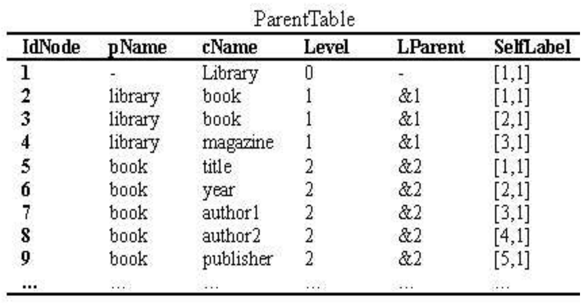
Scenario	Technique to Generate Unique Label
 (a)	Node C is inserted before Node A provided that no nodes before Node A; Label C = (c_i,c_j) :- $(c_i,c_j) = (a_i -1,a_j)$
 (b)	Node C is inserted after Node B provided that no nodes after Node B after Node B; Label C = (c_i,c_j) :- $(c_i,c_j) = (b_i + 1,b_j)$
 (c)	Node C is inserted between Node A and Node B Label C = (c_i,c_j) $\mathbf{c_i} = b_i.\ a_j + a_i.\ b_j / d$ $\mathbf{c_j} = 2.\ a_j.\ b_j / d$: where d is Highest Common Factor for $(b_i.a_j + a_i.b_j)$ and $(2.a_j.b_j)$

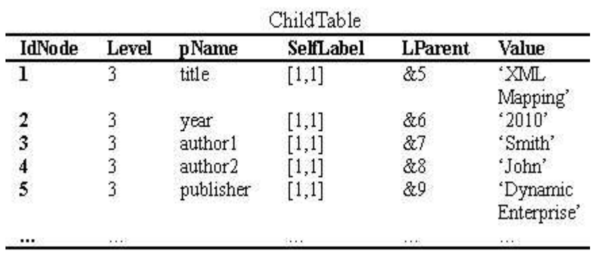
Figure 3. New labels generated due to insertion in Persistent Labeling





ParentTable

IdNode	pName	cName	Level	LParent	SelfLabel
1	-	Library	0	-	[1,1]
2	library	book	1	&1	[1,1]
3	library	book	1	&1	[2,1]
4	library	magazine	1	&1	[3,1]
5	book	title	2	&2	[1,1]
6	book	year	2	&2	[2,1]
7	book	author1	2	&2	[3,1]
8	book	author2	2	&2	[4,1]
9	book	publisher	2	&2	[5,1]
...	...	...	...	...	...



ChildTable

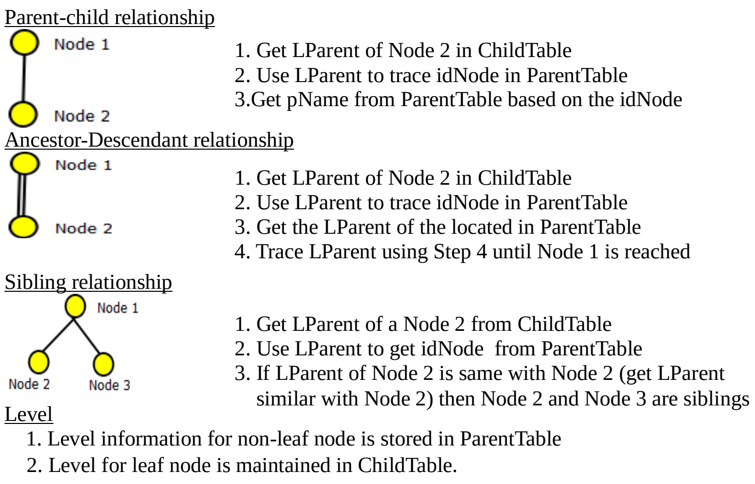
IdNode	Level	pName	SelfLabel	LParent	Value
1	3	title	[1,1]	&5	'XML Mapping'
2	3	year	[1,1]	&6	'2010'
3	3	author1	[1,1]	&7	'Smith'
4	3	author2	[1,1]	&8	'John'
5	3	publisher	[1,1]	&9	'Dynamic Enterprise'
...	...		....	....	...

Figure 4. The structure and sample data of s-XML

ChildTable ***(IdNode, Level, pName, SelfLabel, LParent,Value)*** where:
a) **IdNode** - uniquely identifies the nodes stored in the ChildTable (assigned based on breath-first traversal).
b) **Level** - stores the level information of the node in the XML document.
c) **pName** - stores the element name of the parent node

d) **SelfLabel** - maintains the self-label or local label of the node node which is [n,d] in Persistent Labeling.
e) **LParent**- maintains the parent label of the node which stores the reference of the parent label (IdNode) from the ParentTable.
f) **Value**- stores the value of the node

Figure 4 illustrates some sample data after the annotation and mapping processes. From Figure 4, the initial triplets of Persistent Labeling (level, [parent label], [local label]) is shredded into three columns namely, Level, LParent and SelfLabel. Figure 5 illustrates how the hierarchical relationships could be determined from s-XML.



Parent-child relationship

1. Get LParent of Node 2 in ChildTable
2. Use LParent to trace idNode in ParentTable
3.Get pName from ParentTable based on the idNode

Ancestor-Descendant relationship

1. Get LParent of Node 2 in ChildTable
2. Use LParent to trace idNode in ParentTable
3. Get the LParent of the located in ParentTable
4. Trace LParent using Step 4 until Node 1 is reached

Sibling relationship

1. Get LParent of a Node 2 from ChildTable
2. Use LParent to get idNode from ParentTable
3. If LParent of Node 2 is same with Node 2 (get LParent similar with Node 2) then Node 2 and Node 3 are siblings

Level

1. Level information for non-leaf node is stored in ParentTable
2. Level for leaf node is maintained in ChildTable.

Figure 5. Relationship supported by s-XML

4. EXPERIMENTAL DESIGN

4.1. Experimental Setup

We have implemented s-XML using IntelliJ IDEA Community Edition 9.0.1 using JDK 1.5.0 and MySQL as the database. Experiments have been carried out on the lineitem dataset obtained from the University of Washington XML repository [15]. All our experiments are performed on Acer Intel Pentium dual-core processor T2390 with 160 GB HDD and 1GB DDR2. All numbers presented here are produced by running the experiments five times and averaging the execution times of several consecutive runs.

4.2. Performance Results

Mapping to Relational Database

The first experiment was conducted to evaluate the efficiency the mapping scheme to map the XML data to relational database. s-XML was compared against the existing mapping approaches such as the Edge, Attribute and DTD schemes. The results of the experiments are shown in Figure 6.

The experimental results show that Edge approach took the longest time to map the XML data to relational database, followed by Attribute and DTD mapping schemes. This is due to the fact that Edge approach is only practical when smaller dataset is concern because the entire document is loaded into single Edge table. This consequence will be an inverse when larger dataset is concern bcause it complicates the mapping process and data management becomes inefficient. The delay in Attribute and DTD mapping schemes are caused by the property of these schemes that is to create tables based on dictinct element names that appear in an XML document and table creations depends on the cardinality of the elements in the DTD document respectively. The s-XML mapping scheme performed the best due to its simple mapping techniques and the data is well distributed among adequate number of tables whereby the number of the tables and format of the tables are fixed regarless of the complexity of the XML document.

Query Processing

Table 1 shows the description on the query performed on the lineitem dataset stored in relational database. Using relational database as the underlying storage, the query is written based on Structured Query Language (SQL) command. The time taken to retrieve the queries is depicted in Table 2 while Figure 7 shows the performance comparison.

Table 1. Query description for liteitem dataset

Query No.	Query Description
Query1	Retrieve the label name for the value like ‘careful packages wake’
Query2	Calculate total quantity of orders in the XML document
Query3	Retrieve the ship instruction for the items with the comment like ‘even accounts cajole slyly’

Table 2. The SQL command and query retrieval time for Edge, Attribute, relational DTD and s-XML approaches.

Approach	Query1	Time (ms)
Edge	select * from edgetable where data like ‘careful packages wake%’	1033
Attribute	select * from t where targetID = (select sourceID from L_COMMENT where data = ‘careful packages wake%’)	315
DTD	select * from L_COMMENT where data like ‘careful packages wake%’	252
s-XML	select parentName from childtable where value = ‘careful packages wake’.	218
Approach	**Query2**	**Time (ms)**
Edge	select sum(data) from edgetable where tag ='L_QUANTITY'	1646
Attribute	select sum(data) from l_quantity	495
DTD	select sum(data) from l_quantity	279
s-XML	select sum(data) from childtable where parentName=’L_QUANTITY’	310
Approach	**Query3**	**Time (ms)**
Edge	select ship.data from Edgetable ship, Edgetable commentT, Edgetable t, Edgetable table1 where ship.tag='L_SHIPINSTRUCT' and commentT.tag='L_COMMENT' and t.tag='T' and table1.tag='table' and table1.targetID=t.sourceID and t.targetID = ship.sourceID and ship.sourceID = commentT.sourceIDand commentT.data like 'even accounts cajole slyly%'	5346
Attribute	select ship.data from l_shipinstruct3 ship, l_comment3 comm, t3 t, table3 tb where tb.targetID = t.sourceID and t.targetID = comm.sourceID and comm.sourceID = ship.sourceID and comm.data like 'even accounts cajole slyly%'	921
DTD	select t.L_SHIPINSTRUCT from t1 t, l_comment1 cm, table tb where tb.id = t.parentID and t.parentID = cm.parentID and cm.text = 'even accounts cajole slyly'	537
s-XML	select value from childtable where parentLabel = (select selfLabel from parenttable where parentName = 'L_SHIPINSTRUCT' and parentLabel = (select parentLabel from parenttable where selfLabel = (select parentLabel from childtable where value = 'even accounts cajole slyly')))	591

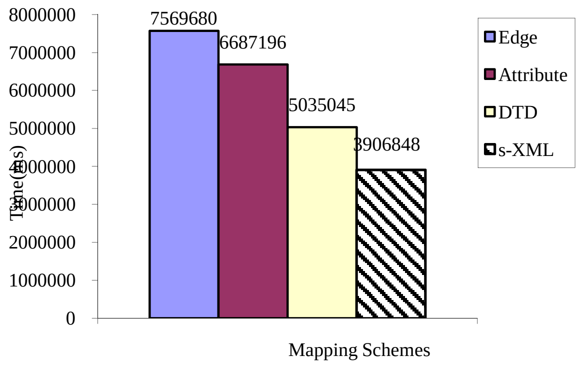


Figure 6. Mapping XML nodes into the relational databases

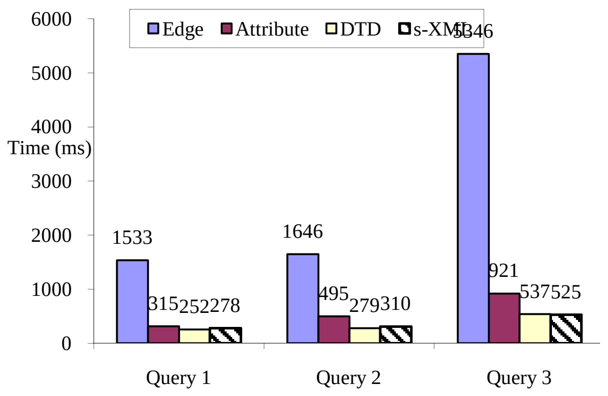


Figure 7. Performance Evaluation Results

From the results obtained, we observed the following:

a) For simple query, Query1, the performance of the Relational DTD, Attribute and s-XML approaches are comparable while the Edge approach performs the worst. All approaches perform only simple table scan. In the Edge approach, all data are shredded into a single table. As such, the table scan operation on the Edge approach is rather slow due to its huge number of rows.

b) For aggregated query, Query2, the Relational DTD and the s-XML approaches performed the best.

c) Relational DTD performance degrades for queries involving complex/assorted combinations (especially on Query3). Since Relational DTD solely depending on the occurrences of elements in the dataset, it performed slower for complex queries due to multiple joins required. Unlike Relational DTD, the number of tables generated in s-XML approach is fixed regardless of the frequency occurrence of the element. As for the Edge approach, the performance is the worst as it involves several self-joins within the huge table itself. Since joins are the most expensive evaluations in relational database, the query processing on the database stored with the Edge approach was the worst.

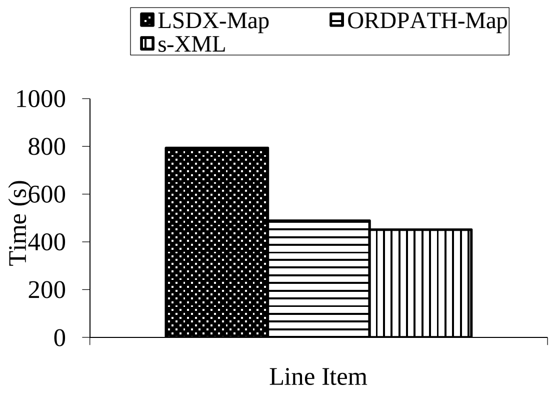


Figure 8. Insertion of new nodes

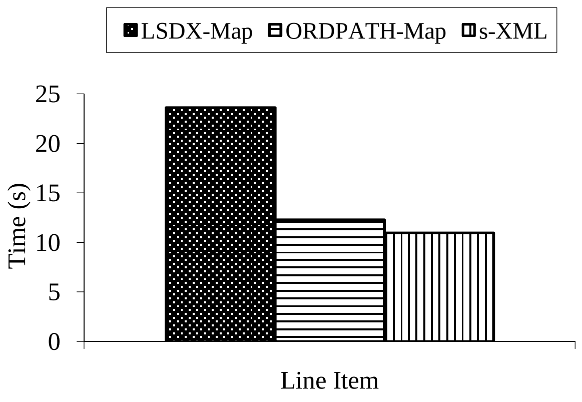


Figure 9. Deletion of nodes

Dynamic Update

The next experiment was conducted to evaluate the efficiency of the labeling schemes in terms of dynamic update, to be exact, measuring the time taken to insert and delete bulk of nodes from the lineitem dataset. Since the labeling scheme in the Relational DTD, Edge and Attribute approach do not support dynamic update, we employ ORDPATH [10] and LSDX [2] as the labeling scheme for comparison. Henceforth, ORDPATH, LSDX and Persistent labeling are known as ORDPATH-map, LSDX-map and s-XML respectively. Figure 8 shows the experimental results for new insertion of nodes into lineitem dataset.

LSDX-map took the longest time to generate new labels for newly inserted nodes. This is for the reason that LSDX causes collision during new label generation and also complexity in mapping process. Furthermore, the size of the labels reduces the efficiency of this labeling scheme as compared to s-XML. On the other hand, the performance of ORDPATH-map is comparable to s-XML due to simple calculation for new label generation and faster mapping to the relations. s-XML performed the best due to controlled labeling size regardless of the complexity of the XML document and dynamic update which is an added advantage compared to other approaches.

Besides that, these labeling schemes were also evaluated in terms of their robustness during node deletion from lineitem dataset and their results were recorded in Figure 9. LSDX-map took the longest time to delete the nodes and update the new document followed by ORDPATH-map and s-XML. The performance of ORDPATH-map and s-XML is analogous since they require least time to delete the nodes and update the document. The ever-increasing label size of ORDPATH-map causes its performance to degrade as compared to s-XML which maintains the labeling format in any circumstances.

5. CONCLUSION

XML document requires robust and seamless mapping approach which allows for efficient and accurate data shredding into relational database. In this paper, we proposed a new mapping scheme named s-XML which is based on Persistent Labeling scheme to support structural queries retrieval efficiently. The experimental evaluations revealed that s-XML processed query efficiently, especially on complex queries as compared to Relational DTD, Attribute and Edge approaches. In addition, the performance of s-XML was better than ORDPATH-map and LSDX-map in terms of the support during dynamic update.

REFERENCES

[1] M. Atay., et al., “Efficient schema-based XML-to-Relational Data Mapping,” *Information Systems*, vol. 32, no. 3, pp. 458-476, 2007.

[2] M. Duong, and Y. Zhang, "LSDX: New Labeling Scheme for Dynamically Updating XML Data", *Proc. Of 16th Australian Database Conference*, pp. 185- 193, 2005.
[3] D. Florescu, and D. Kossman, "Storing and Querying XML Data Using an RDBMS", *IEEE Data Engineering Bulletin*, vol. 22, no. 3, pp. 27-34, 1999.
[4] A. Gabillon, and M. Fansi, "A Persistrent Labeling Scheme for XML and tree Database", *Proc. of ACI*, pp. 110-115, 2006
[5] T. Harder, et al. "Node Labeling Schemes for Dynamic XML Documents Reconsidered", *Data & Knowledge Engineering*, 60, pp. 126-149, 2007.
[6] S.C. Haw, and , C.S. Lee, "Node Labeling Schemes in XML Query Optimization: A Survey and Trends", IETE *Technical Review*, pp. 88-99, 2009.
[7] L. Khan, and Y. Rao, "A Performance Evaluation of Storing XML Data in Relational Database Management Systems", *Proc. of the Workshop on Web Information and Data Management*, pp 31-38, 2001
[8] I. Nekrestyanov, et al. "An Analysis of Alternative Methods for Storing Semistructured Data in Relations", *Lecture Notes In Computer Science*, 1884, pp 354-361, 2002.
[9] M.F. O'Connor, and M. Roantree, "Desirable Properties of XML Update Mechanisms", *Proc. of EDBT/ICDT Workshop*, 2010.
[10] P. O'Neil, et al. "ORDPATHS: Insert-Friendly XML Node Labels", *Proc. Of ACM SIGMOD*, pp.903- 908, 2004.
[11] V. Sans, and D. Laurent, "Prefix Based Numbering Schemes for XML: Techniques, Applications and Performances", *Proc. Of VLDB*, pp.1564-72, 2008.
[12] B.J. Shin, and M. Jin, "Association Inlining for Mapping XML DTDs to Relational Tables", *Lecture Notes In Computer Science*, 3046, pp 849-858, 2004.
[13] F.Tian, et al. "The Design and Performance Evaluation of Alternative XML Storage Strategies", *Proc. of the ACM SIGMOD*, pp 5-10, 2001.
[14] Tatarinov, et al. "Storing and Querying Ordered XML Using a Relational Database System", *Proc. of SIGMOD*, pp. 204-215, 2002.
[15] UW XML Repository, http://www.cs.washington.edu/research/ xml/datasets/www/repository.html.

BIOGRAPHY OF AUTHORS

Samini Subramaniam's research interest are in Relational Database Management System (RDBMS), XML processing and Query processing and optimization.

Su-Cheng Haw's research interests are in XML Databases and instance storage, Query processing and optimization, Data Modeling and Design, Data Management, Data Semantic, Constraints & Dependencies, Data Warehouse, E-Commerce and Web services.

Poo Kuan Hoong's research interests are in the areas of peer-to-peer networks and distributed systems. He is a member of IEICE, IEEE and ACM.