Storing XML documents in Relational Database using Multi Dimension Links

Ibrahim Mohammad Dweib
Department of Computer Science
Sultan Qaboos University
Sultanate of Oman
dweib@squ.edu.om

Abstract

Extensible Markup Language (XML) nowadays is one of the most important standard media used for exchanging and representing data through the Internet. Storing, updating and retrieving the huge amount of web services data such as XML is an attractive area of research for researchers and database vendors. In this paper, we propose and develop a new mapping model, called Multi Dimension Links (MDL), for storing, rebuilding, updating and querying XML documents using a Relational Database without making use of any XML schemas in the mapping process. This model is used to maintain XML document structure, manage the process of updating document contents and retrieve document contents efficiently. Experiments are done to evaluate MDL model. MDL will be compared with other well-known models available in the literature using total expected value of rebuilding XML document execution time and insertion of token execution time.

Keywords: XML, Mapping, Schemaless, Relational Database

I. INTRODUCTION

The World Wide Web (WWW) nowadays is an important medium used by many people for many activities in their daily life (i.e.; e-management, e-learning, e-mail, e-library and e-business). Many enterprises are working together using XML technologies for exchanging their web services data. Exchanging, sorting, updating and retrieving these huge data has become a source of concern for researchers and database vendors.

At present, storing and retrieving of XML documents can be done using mainly three approaches, i.e., native XML database [1], Object Oriented Database [2] and Relational Database (RDB) [3], [4], [5], [6], [7], [8].

The most important factor in choosing the target database is the type of XML documents to be stored, data-centric (e.g., bank transaction, airlines transactions) or document-centric (e.g., emails, books, manual).

Using a hybrid approach of RDB to store and retrieve data and XML to exchange and represent it. This will solve most of the data issues of integrity, multi-user access, retrieving, exchanging, concurrency control, crash recovery, indexing, security, storing semi-structure data, and reliability. But some drawbacks in this approach could raise: Loss of information, difficulties in updating its contents and difficulties in rebuilding of original document. The mapping techniques of this approach can generally be classified into two tracks: Schemaless-centric technique and schema–centric. Schemaless-centric technique is used to make use of XML document structure to manage mapping process [9], [10], [11], [12]. In schema–centric, XML schema information is used to develop a relational storage for XML documents [5], [3]. Each approach introduced some solutions for the mapping process but failed to solve others.

In this paper we will concentrate on a new approach for mapping XML documents into RDB which is called MDL (i.e. Multi Dimension Links). The model does not make use of any XML schemas to manage mapping process. In this model, the document structure and document contents are stored in RDB tables. It uses multi-links to reserve document structure and elements relations within the document as parent-child, ancestor-descendant, left-sibling and right-sibling. The use of multi-links will make the insertion process cost for new elements and attributes anywhere in the document close to constant value, since there is no need to relabel the elements and the attributes following the inserted element or attribute. Other models [11], [13] which consider the element or attribute label as an identifier to reserve document structure, the cost of insertion in this case will vary depending on the position of insertion, since relabelling is needed after each insertion to maintain the document order.

II. STATE OF THE ART TECHNOLOGY

A number of different techniques for storing XML documents in a RDB have been established. These techniques can be divided into two tracks: the schemaless-centric technique and the schema–centric technique. The first one makes use of XML document structure to manage the mapping process [11], [12], [9], [10]. The second one depends on schema information to develop a relational schema for XML documents [3], [5].

Some studies work on optimizing query time [13], [12], but they fail to update XML document stored in RDB. That is because each insertion requires a lot of nodes to be relabelled after insertion of new node or subtree. Others [2],[4] solve partially the updating problem by creating a gap within the label, but there is still a need for relabelling after consuming the reserved space. Other studies [3], [5] work on storage optimization and create a relational schema depending on XML schema. Redundant data are removed by creating new relation for each recursive child (or inlining some child in parent

relation to reduce the number of created relation). Sometimes a large number of relations are needed to be created for some complex document. Consequently, large numbers of joins are needed to retrieve document information from a RDB. Also sometimes XML schema is not available for some documents which require reconstructing XML schema first from document structure, and creating relational schema based on it. XML reconstruction is considered as a time overhead in this case. In some studies like [9], they do a map for some parts of the XML document. They used the query to optimize the mapping time from XML document to RDB. They did not store the entire content of a document in a RDB. This method requires a mapping for each query, and cannot make use of other data stored in RDB.

The studies that address the problem of mapping XML document into RDB take care of the above issues, and attempt to translate users' XML queries, either XPath expression [14] or W3C's recommendation XQuery expression [15] into SQL queries [16]. XQuery gives power to the translation method since XQuery comprises XPath, and it is recommended by W3C, while XPath is not. The translation method should also consider its ability to rebuild, the stored XML document without losing information, and retrieve it in an acceptable time. Many studies have tried to address translation and restore constructing labelling methods. Labelling methods aim to reserve nodes order, parent-child and ancestor-descendant relationships, and document structure [11], [2], [12], [4], [17].

Storing XML documents into relational databases makes use of RDBMS, (i.e. multi-user access, data integrity, security, crash recovery) and makes use of its high potential query language SQL. Using original XML document after the mapping process will be out of use if any updating is done on the document. This makes rebuilding of XML documents from relational databases is equally important as a big deal. The rebuilding process raises a lot of issues to be considered, such as: 1) Reserving the structure of the original document, including nodes order and relationships when efficient labelling methods are used for rebuilding (i.e., parent-child, ancestor-descendant and preceding-following relationships), 2) Making sure that all document contents are stored (elements, attributes, comments … etc.), 3) The rebuilding process should be efficient for the entire document or some parts of it.

These rebuilding solutions depend on the method used for mapping, and the way of labelling the contents of XML document in RDB.

Schema-less centric techniques do not require an XML DTD or XML Schema. Present proposals depend on XML document's structure to manage the mapping process. In such approaches, XML document is entirely stored as a large solid object data type (CLOBs, BLOBs for example), which are provided by most RDB vendors (e.g., Oracle interMedia Text, DB2 Text Extender). Another way is to map the tree or graph of the XML document generically onto predefined relations. These approaches depend on using a long-character-string data type, such as CLOB in SQL, to store XML documents or fragments as texts in columns of tables. The advantages of these approaches are: (1) They could provide textual fidelity since they preserve the original XML at the character string level, and (2) there is no need for an XML schema in the storing process. The drawbacks of these methods are: (1) They cannot make use of the XML Markup structural information, (2) they don't take into account the query workload while constructing the relational schema, and (3) the XML document structure is not preserved.

Schema centric techniques need XML schema to develop the relational schema. Such techniques sometimes need to create a relational schema to store the XML schema. The created schema is used during and after shredding the XML documents. The data that is captured from the XML document is stored in the created relational tables. The advantages of these techniques are: (1) they restrict XML structure to the defined schema, (2) they enforce referential constraints, primary and foreign key relationships, and (3) they simplify the mapping process because users are not involved in addressing a new mapping language. But, the techniques reviewed above are (1) do not consider multiple possible relational mappings so as to choose the optimal one; (2) XML schemas are sometimes not available, so there is a need to construct the schema first and then do the mapping. (3) A reconstruction of database schema is needed as any change in the XML schema happens, which makes it very expensive in this case. (4) Sometimes, a large number of relations need to be created depending on the XML schema; consequently, a lot of joins are needed to retrieve XML document information.

III. MDL MODEL

MDL Theory

Storing XML document into RDB means storing ordered, hierarchical and structured information into an unordered tables. XML manipulation is still facing some problems such as retrieving information, updating data contents, concurrency control and multi-user access. These problems can be overcome by using RDB to store, update and retrieve XML documents contents. Labelling techniques are used in order to preserve XML document structure, and the relations among its contents. MDL adopts the Global Labelling method with some modifications [11]. Global Labelling is modified to make the cost of the execution time of XML document updating constant, and to preserve parent-child and ancestor-descendant relationships. The new model uses document structure information to guide the mapping process, Consequently DTD or XML Schema information availability is not required.

Theory Background

The hierarchy of XML document could be represented as a tree structure. XML tree can clearly represent the relationships between nodes of document content.

Definition 1: An XML tree is a collection of many nested subtrees of depth two. It can be denoted as follows:

$$T = \sum_{i=1}^{n} \sum_{j=1}^{m} S_{ij} \qquad (1)$$

Where:

$J = 1, 2, 3 \ldots m$ represent the order of subtree number within i^{th} level;

$I = 1, 2 \ldots$ n represents tree level number and 1 also represents the tree root; and

S_{ij} represents a subtree structure and is denoted as

$$S_{ij} = E_{ij}(\sum_{r=1}^{l_1} X_{jr}, \sum_{k=1}^{l_2} A_{jk}, \sum_{w=1}^{l_3} E_{jw}, \sum_{z=1}^{l_2+l_3} G_{iz}) \quad (2)$$

Where:

E_{ij} represents the root of the subtree S_{ij}

l_1 represents number of text (X) in subree S_{ij}

l_2 represents number of attributes (A) in subree S_{ij}

l_3 represents number of elements (E) in subree S_{ij}

$G_{i,z}$ is a finite set of edges between E_{ij} and its childs representing parent-child relationship (l_2+l_3).

Definition 1 moves the organization of XML document, from being a tree of multi-dimensional way with arbitrary depth and width, to a tree structure of depth two. The resultant tree is unbounded fan-out at first level and fixed fan-out at the second level. The first level can be represented in RDB as tuples (i.e. rows) and the second level can be represented by fields (i.e. columns).

Mapping Framework

Our approach considers well-formed XML documents, which are shredded and decomposed into elements and attributes, and then these elements and attributes are inserted into the RDB tables. It does not consider the XML schema for the following reasons:

- Many applications need highly flexible XML documents whose structure is not easy to define by DTD or fixed schema. Therefore, schema-less approach is better to deal with such XML documents.
- It is not practical to design many candidate relational schemas for all potential XML data which may have different XML schema.

Labelling Method

Multi Dimension Links (MDL) are used to maintain the XML document contents. MDL uses a global labelling approach that gives labels for XML elements and attributes. A unique label is given for each element and attribute. The sequence of label is not essential as [11], [12]. Point out, an initial pre-order traversing for the XML document is performed to assign a label for each element or attribute. No re-labelling is needed for XML document elements and attributes (tokens) in case of adding new element or attribute. In contrast [11], [12] and [13]; proved the reverse, all tokens that follow the new inserted token should be relabelled. In pre-order, post-order two labels are to be updated. In order to achieve this objective, MDL uses the following format to identify a token:

- Token (tokenID, leftID, parentID, rightID, prevID)
- tokenID is a unique label given to identify each token.
- leftID (Left-sibling) is the tokenID of the preceding sibling token.
- parentID (Parent) is the tokenID of the current token parent.
- rightID (Right-sibling) is the tokenID of the following sibling token.
- prevID is the tokenID of the previous token of the current token in the document structure.

Fig. 1 gives an overview on the multi dimension links (MDL) and the relations between its nodes.

The tokenID and parentID are used to maintain the parent-child and ancestor-descendant relationships, while leftID and rightID together with tokenID are used to maintain elements and attributes order as siblings and brothers relationships within the documents structure.

A fixed relational schema consisting of three tables is used to store XML documents' contents and their structure. The first table is called "documents table"; it preserves XML documents information. The second table is called "tokens table"; it preserves XML documents contents and structure. The third table is called "XpathQuery table"; it is a temporary table used to preserve token paths for a desired XPath expression from a document's root down to the desired token.

Relational Schema

1. Document master table: It is called "documents" table. This table keeps information about documents themselves; its minimal structure is:

   Documents(documentID, documentName, Header, docElement, schemaInfo, maxTokenId, XpathCount)

2. *"Tokens"* table: A table to store the actual content and structure for all documents. Documents will be shredded into pieces of data called tokens. Each document element, or element attribute will be considered as a token. The "tokens" table will have the following structure:

   Tokens(documentID, tokenID, leftID, parentID, rightID, treeLevel, prevID, tokenName, tokenValue, tokenType)

3. *"XpathQuery"* table: A dummy table that is used to store all tokens involved in desired XPath expression. This table will have the following structure:

   XpathQuery(documentID, XpathID,tokenID, TreeLevel, ParentID, tokenName, TokenValue, TokenType)

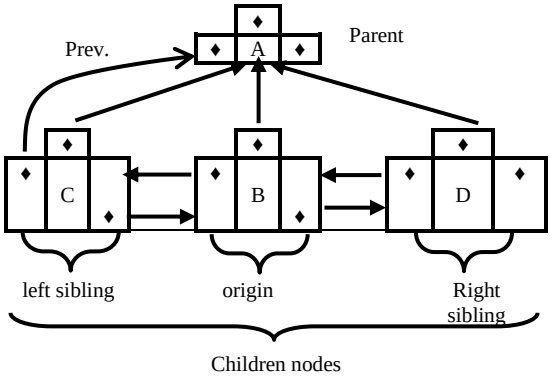

Fig. 1: Multiple linked list over view

Updating XML Document Contents

Insertion of New Token

This section gives more evidence that the method used in this model makes insertion time cost of new token, anywhere in the document, constant; this, one of the main objectives of the new model is achieved. The insertion process can be clarified by the following rules:

a. Insertion of a new token to the left of a subtree, left to S1:

1) The new token T gets a label tokenID following the maxTokenID in the document. *TokenID(T) = maxTokenID + 1*.
2) *RightID(T) = RightID(S1)*
3) *LeftID(T) = 0*
4) *LeftID(S1) = tokenID(T)*
5) *ParentID(T) = ParentID(S1)*
6) *prevID(T) = prevID(S1)*
7) *PrevID(S1) = tokenID(T)*

b. Insertion of a new token to the right of a subtree (right to S1), insertion of a new token T as a leaf and child of S1 and insertion of a new token T as a parent of S1 will be the same as for *a* but the updating will be for different links and it is removed due to space limit.

Fig. 2 gives an overview of inserting new token (i.e. element or attribute) in the XML document. In the figure, the new element "subject" is inserted between "author" (labelled (4, N, 2, 6)) and "title" of label (6, 4, 2, N). The new element is given tokenID equals to maxTokenID + 1, which is 11. And the token links are updated as follows:

1) *rightID(subject) = rightID(author)*
2) *leftID(subject) =leftID(title)*
3) *rightID(author) = tokenID(subject)*
4) *leftID(title) = tokenID(subject)*
5) *prevID(subject) = prevID(title)*
6) *prevID(title)= tokenID(subject)*
7) *ParentID(subject) = ParentID(title)*

As seen from the previous discussion, there is no need for relabelling the tokens that follow the inserted token "subject". All tokens' labels in the document remain as they were before the insertion process. While in [11], [12], [13], all the following nodes of new inserted element "subject" must be relabelled, and the cost of relabelling depends on the location of the new inserted element. The highest cost is gained when the insertion happens at the beginning of the document, and the lowest cost is gained when the insertion takes place at the end of the document.

IV. Experiments and Their Assessment

Experiment Setup

XML Data Sets Used for Testing the Model

Three XML benchmarks are used to assess the usability and efficiency of MDL: XML benchmark from Washington University [18], XMark benchmark [19] and Michigan XML benchmark [20]. XML document generator XMLgen from XMark is used to create documents of different sizes using factors of the original one.

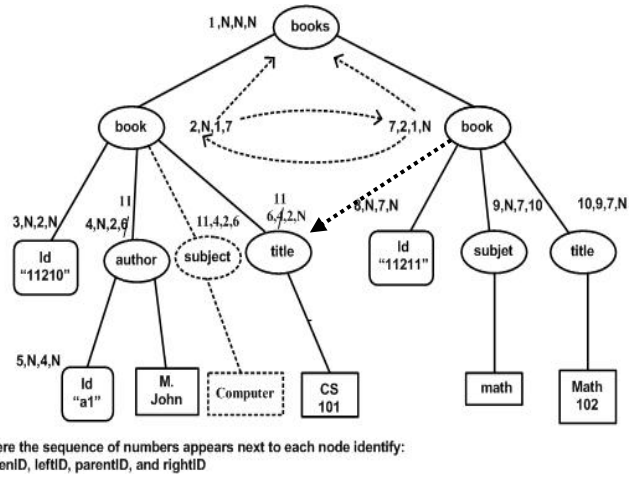

Fig. 2: A tree representation for XML document

"Tree-bank" document is taken from Washington benchmark, "Auction documents" from XMark, and "Xbench-TCSD-small" and "Xbench-TCSD-normal" from Michigan benchmark. These documents characteristics are shown in Table 2.

Michigan XML benchmark data sets are used for evaluating the performance of the model against the complicated characteristics of XML documents such as depth, fan-out in "tree-bank" document. Scaling a benchmark data set in the relational model is done by increasing the number of records. Scaling in XML, however, can be done by increasing depth, number of nodes, or fan-outs. The data sets in Table 2 and Table 1 are used to evaluate the model performance and usability in both directions, for mapping the documents into RDB and for rebuilding the mapped documents from RDB.

For evaluating the update performance of our model, we used the set of documents in Table 4. The documents are created from auction document using XMLgen. We choose small factor between 0.001 and 0.006 to get small size document that can be managed by our editor. Many experiments can be performed to insert new tokens in different places: In the beginning, in the middle and at the end. They can also have different relationship with the candidate element such as parent, child, left-sibling and right-sibling.

The experiments will be done to check the scalability and effectiveness of our model. Then we will compare our model with the Global Encoding model [11] and the Accelerating XPath model [13]. The comparisons consist of three stages: mapping, building, and updating.

Performance Measurement

- Mapping XML document into RDB execution time.
- Rebuilding of XML document from RDB execution time.
- Inserting nodes processing time (number of nodes to be relabelled).

The execution time is used as an evaluation scale in this research rather than storage space since the former is crucial nowadays for the users, while storage space is available in a very huge size with reasonable prices.

Testing Strategies

Mapping XML Document into RDB Performance

The experiment is performed as follows:

Face 1, scalability test: An XML document generator from XMark [19] is used to create documents of different sizes with factors of 0.1, 0.2, 0.3, 0.4 and 0.5. The documents characteristics are shown in Table 1. In this experiment, our model shows performance in a linear and scalable manner as document size is increasing. The mapping result over different sizes of the same document is shown in Fig. 3.

Face 2, effectiveness test: Three groups of documents of different sizes 11MB, 82MB and 107MB but with different structure and different numbers of token are included in this experiment. Table 2 shows documents properties and their mapping and rebuilding time. Fig. 4 shows the time required for mapping XML documents into RDB which consistently increases as the number of tokens increases in the document.

Considering the results shown in Fig. 3 for homogenous documents and those shown in Table 2 and Fig. 4 for heterogeneous documents coupled with calculating the correlation coefficient between document size and mapping time in the two cases r1=0.99988 and r2=0.8751 on the one hand, and the number of tokens and mapping time in the two cases r3= 0.99991 and r4=0.9991 on the other hand, we can conclude that the time required for mapping the document largely depends on the number of tokens (i.e., elements and attributes) in the document, the document size and document depth (r=0.1752) respectively.

Now let us compare MDL model with Global Encoding for [11] and Accelerating XPath for [13], since the three models are using the same general number encoding to identify the XML document of elements and attributes (tokens).

The three models use one scan to shred the document contents, assign an identifier for each token, reserve node information, (i.e. token name and token value) to store them in one tuple in RDB. Global Encoding adds another table for tokens path from the document element passing through until the candidate token. MDL and Accelerating XPath are similar in using just one table to store documents contents. Both also use a stack collection to manage post-order label in Acceleration XPath and RightID link in MDL.

Based on previous experiment, one finds that mapping time mainly depends on the number of tokens in the document. Based on that, we may consider the following assumptions:

$$T = \begin{cases} t, & for\ both\ MDL\ and\ Accelerating\ XPath \\ t + tp, & for\ Global\ enconding \end{cases} \quad (3)$$

Where T is the mapping time and tp is the time required to process the tokens path.

$$tp = (t/n) * m \quad (4)$$

Where n is the number of tokens in the document and m is the number of distinct paths in the document.

TABLE 1: XML DATASETS OF EQUAL DEPTHS* AND DIFFERENT SIZES

Document Name	Factor used	Doc. Size (MB)	# of nodes	Mapping Time (Sec)	Building Time (Sec)
Auction_1	0.1	11.3	206130	26.84	14.14
Auction_2	0.2	22.8	413111	54.31	31.09
Auction_3	0.3	34.0	616229	80.45	48.88
Auction_4	0.4	45.3	820438	108.43	69.00
Auction_5	0.5	56.2	1024073	136.52	88.00

*. Max depth for all documents = 12

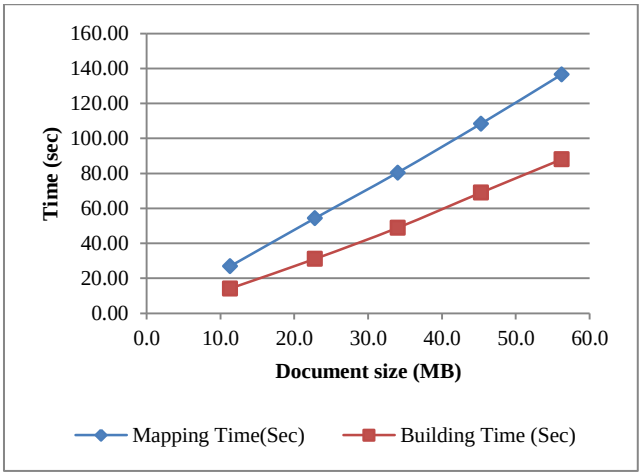

Fig. 3: Mapping time Building time for dataset in Table 1.

TABLE 2: XML DATASET OF DIFFERENT STRUCTURES

Document	Doc Size (MB)	# of Token	# of XPath	Max depth	Mapping (Sec)	Building (Sec)
Auction11	11	200358	502	12	25.50	13.41
Xbench-TCSD-small	11	283312	26	8	36.75881	17.3946
Auction82	82	1485699	502	12	186.7157	141
Tree-bank	82	2437667	168123	36	325.2331	150
Auction107	107	1946203	502	12	260.3572	200
Xbench-TCSD-Normal	107	2757084	26	8	376.7195	181

Now we can use the results of experiments 1 and 2 for mapping XML documents into RDB and compare our model with the other two models.

From Table 3 we can see that MDL and Accelerating XPath are identical while Global Encoding is closed to the other two models in homogeneous documents where the number of paths is small and the gap becomes larger for heterogeneous documents where the number of paths becomes very large as in tree_bank document.

Rebuilding XML Document from RDB Performance

The experiment is done at different stages as follows:

Face 1, scalability test: the auction documents in Table 1 mapped before will be built in this experiment to see the scalability of MDL in rebuilding XML documents from RDB.

From the results shown in Fig. 3, we find that our model performs well for rebuilding the XML document. The time for rebuilding a document of 11.3MB size is 14.14 seconds and for

56.2MB size is 88.00 seconds. This shows that the relation between rebuilding time and document size is approximately linear as it passes through the origin and is given as follows:

$$t = 1.644989\, s \qquad (5)$$

Where t is the time in seconds for rebuilding the document and s is the size of the document in MB.

Face 2, effectiveness test: The same sets of documents from Table 2 are also used to check the ability of MDL in dealing with different XML document types. The documents are grouped by size and every two have the same size.

From the experiments done and results shown in Table 2, it can be concluded that the time of rebuilding the document is influenced by the number of tokens formulating the document because two documents of the same size need different amounts of time for rebuilding.

From the results shown in Fig. 3 for homogenous documents and results shown in Table 2 and Fig. 4 for heterogeneous document, and after calculating the correlation coefficient between document size and rebuilding time (r_1= 0.998795203, r_2= 0.926455747), and number of tokens and rebuilding time (r_3= 0.999311324, r_4= 0.308485455), we can conclude that the time required for rebuilding the document mainly depends on the document size, the number of tokens (elements and attributes) that exist in the document and the document depth (r= 0.214860654) respectively.

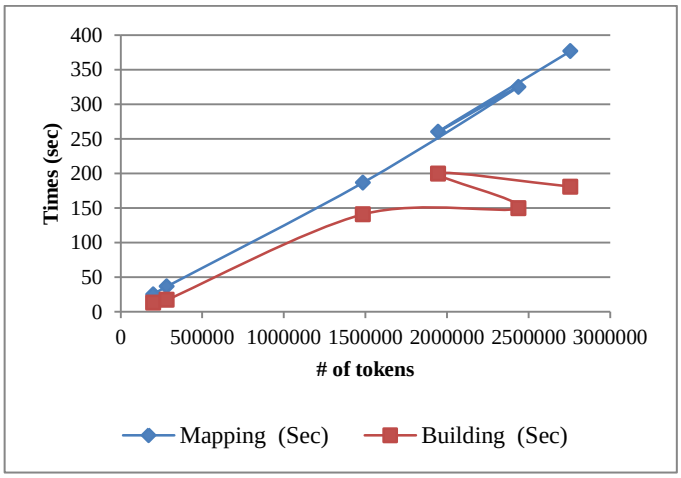

Fig. 4: Mapping time and Building time for documents in Table 2.

TABLE 3: MAPPING TIME FOR MDL, ACCELERATING XPATH AND GLOBAL ENCODING IN SECONDS

Doc. Size (MB)	# of Token	# Different path	M-MDL	M-Accel	M-Global
11	200358	502	25.4965	25.4965	25.56038
11	283312	26	36.7588	36.7588	36.76218
82	1485699	502	186.7157	186.7157	186.77879
107	1946203	502	260.3572	260.3572	260.42436
82	2437667	168123	325.2331	325.2331	347.66404
107	2757084	26	376.7195	376.7195	376.72305

Face 3, Building XML document after the insertion of elements in three locations:

1. At the beginning of the document.
2. In the middle of the document
3. At the end of the document.

Table 4 and Table 5 show results of rebuilding auction document of several values of n, where n is the number of tokens in the document. In Table 4, column 3 shows the time required for rebuilding the documents before any update, column 4 after inserting a token at the beginning of the documents, column 5 after inserting at the middle and column 6 after inserting at the end of the documents. Table 5 shows the difference between the required time for rebuilding the document after inserting in the defined location and the rebuilding time of the original document and the percentages of that difference.

The averages of percentages are different. The cost of rebuilding the document depends mainly on the location of inserting the new tokens. The cost decreases from *1.24*t* at location L1 to t at location L3, where L1 denotes token number *2* and L3 denotes token number *n + 1*, and t represents the time required for rebuilding the original document before any insertion.

Next, we will compare our model with the models of [11] and [13]. The comparison will be based on the rebuilding document cost in time (BDCT) and the cost in time of inserting a new token (element or attribute) (ITCT). The comparison will make use of the discussion above. In the following results, we will give the expected value of the BDCT and ITCT for the models under study.

Theorem-1:

(a) Under the following assumptions:

1- We will assume that the locations of insertion have the same probability,

$$P[X = x] = 1/n, \qquad x = 2, 3, \ldots n+1 \qquad (6)$$

where X denote the location of insertion.

2- We will assume that the time decreases from 1.24*t at location 2 to t at location $n+1$ uniformly, i.e.

$$P[Y = 1.24 - [0.24*(y-2)/(n-1)]*t] = 1/n,\ y = 2,3 \ldots n+1 \qquad (7)$$

Where Y denotes the time required to build the document after inserting a new token at position y, we have:

$$E_{11} = E_{MDL}[BCDT] = 1.24*t - 0.12*t\,(n-1)/n \qquad (8)$$

(b) $$E_{12} = E_{Blobal}[BDCT] = t \qquad (9)$$

(c) $$E_{13} = E_{Acc}[BDCT] = t \qquad (10)$$

where E_{model} denotes the expected value of BDCT under the model. The Proof of Theorem 1 removed due to space limit.

For E_{12} and E_{13}, in both cases the tokens there are sorted in sequential order and the time needed for building the document is equal to t.

Remark: the motivation of the assumptions 1 and 2 in the theorem are based on the experiment results in Table 4 and Table 5.

Updating Performance

To evaluate our model updating performance, the experiment is performed as follows:

a. Inserting a child node in different location in the document and at different levels.
b. Inserting a preceding-sibling (i.e. before) node in different locations in the document and at different levels.
c. Inserting a following-sibling (i.e. after) node in different locations in the document and at different levels.
d. Inserting a parent node in different location in the document and for different levels.

Table 6 shows the time in seconds needed to process the inserting nodes. The figures in the table show that the number of tokens has an influence on the processing time wherever the insert on process occurs, in the beginning of the document, in the middle or at the end. For cases of inserting a token as a child or before (i.e. left-sibling), the time cost is constant, but for the other two cases, parent and after (i.e. right-sibling), the cost is variable. For the parent node since we have an identifier for token level in the document, all descendant nodes tree level should be updated (i.e. incremented by 1). While for after nodes (right-sibling) we should look at descendant nodes for the proper PrevID link for the new node. That means, there is an increase in the cost of insertion time depending on the size of the candidate node (i.e. number of descendant nodes) for the two cases.

Theorem-2:

(a) Under the following assumptions:
1- We will assume that the locations of insertion have the same probability,

$$P[X = x] = 1/n,\ x = 2, 3 \ldots n+1 \qquad (11)$$

where X denotes the location of insertion.
2- We will assume that the time decreases from $n*t_0$ at location 2 to t_0 at location $n+1$ uniformly, i.e.

$$P[Z= t_0 [n - z + 2] = 1/n,\ y= 2,3 \ldots n+1, \qquad (12)$$

Where Z denotes the time required to insert the new node at position z, we have:

$$E_{22} = E_{Global}[ITCT] = t_1 n/2 + t_0/(n+1) \qquad (13)$$

(b) $E_{21}=E_{MDL}[ITCT] = t_0 \qquad (14)$

(c) $E_{23} = E_{Acc}[ITCT] = t_1 n + t_0/(n+1) \qquad (15)$

where E_{model} denotes the expected value of ITCT under the model.

The Proof of Theorem-2 removed due to space limit.

b) For E_{21}, since there is no relabelling needed after insertion of a new token, then, the cost of inserting a new node is equal to t_0.

c) For E_{23}, since there is a need to update the pre-order and post-order label, the cost of update will be double the cost of update one of label after insertion of a new token.

Remark: the motivation of the assumptions 1 and 2 in the theorem are based on the experiment results in Table 4 and Table 5.

Model Analysis and Comparison

We will compare the models, MDL, Global encoding and Accelerating XPath using the total expectations of the cost of building the document (BDCT) and the cost in time of insertion of a new token (ITCT) (whose expression are given in Theorems-1 and Theorem-2) as follows:

$$E_1 = E_{11} + E_{21} = 1.24\ t - 0.12\ t\ (n-1)/n + t_0 \qquad (16)$$

$$E_2 = E_{12} + E_{22} = t + (t_1 n/2 + t_0/(n+1)) \qquad (17)$$

$$E_3 = E_{13} + E_{23} = t + (t_1 n + t_0/(n+1)) \qquad (18)$$

Where t denotes the time in seconds required for building the document, t_0 denotes the time in seconds required for inserting the new token, t_1 denotes the time required to update the label.

In Table 7, we calculated the total expectation time for building XML documents from RDB and for inserting new tokens in different positions in the document with probability $1/n$, where n the number of tokens in the document. E_1, E_2 and E_3 which is the total expectation time for MDL, Global Encoding and Accelerating XPath respectively.

From Table 7 and Fig. 7 we can see that our model MDL outperform the two models for the total expectation time. And the difference becomes large for a large number of tokens n.

TABLE 4: BUILDING TIME AFTER UPDATE FOR ACUTION DOCUMENT*

Doc. Size (KB)	# of Tokens	Before insertion	Insertion Location		
			Begin	*Middle*	*End*
115	2086	0.1256	0.1598	0.1384	0.12623
210	3684	0.2264	0.2759	0.2474	0.22581
318	6284	0.3854	0.4799	0.4341	0.37913
457	7957	0.4963	0.6134	0.5671	0.49238
567	10492	0.6419	0.8116	0.7307	0.64538
682	11911	0.7295	0.8924	0.8245	0.73666

*. DOC. SIZE OF AUCTION DOC. FACTORS FROM 0.001, 0.002, … 0.006 USING XMLGEN

TABLE 5: DIFFERENCES IN BUILDING TIME

Doc. Size (KB)	Differences			Percent		
	Begin L1	*Middle L2*	*End L3*	*Begin*	*Middle*	*End*
115	0.03425	0.01281	0.00067	27%	10%	1%
210	0.04950	0.02100	-0.00056	22%	9%	0%
318	0.09450	0.04869	-0.00631	25%	13%	-2%
457	0.11719	0.07088	-0.00388	24%	14%	-1%
567	0.16969	0.08875	0.00344	26%	14%	1%
682	0.16291	0.09497	0.00716	22%	13%	1%

TABLE 6: TIME COST OF INSERTION OF A TOKEN IN DIFFERENT LOCATION

Location in Doc.	Insert Location (time in Sec)				# of token in Doc.
	Parent	*Child*	*Before*	*After*	
In-Beginning	0.046875	0.015625	0.015625	0.015625	2086
At-Middle	0.015625	0.015625	0.015625	0.015625	
At-End	0.015625	0.015625	0.015625	0.078125	
In-Beginning	0.0625	0.015625	0.015625	0.03125	3684
At-Middle	0.015625	0.015625	0.015625	0.015625	
At-End	0.015625	0.015625	0.015625	0.015625	
In-Beginning	0.046875	0.015625	0.015625	0.03125	6284
At-Middle	0.0625	0.015625	0.015625	0.015625	
At-End	0.015625	0.015625	0.015625	0.015625	

TABLE 7: TOTAL EXPECTATION TIME FOR BUILDING AND INSERTING TOKENS FOR THE THREE MODELS (IN SEC)

n	t_0	*t*	t_1	E_1	E_2	E_3
2086	0.015625	0.1256	0.00488	0.15882	5.21734	10.30907
3684	0.015625	0.2264	0.00488	0.27373	9.21870	18.21099
6284	0.015625	0.3854	0.00488	0.45499	15.72405	31.06270
7957	0.015625	0.4963	0.00488	0.58142	19.91858	39.34086
10492	0.015625	0.6419	0.00488	0.74740	26.25188	51.86185
11911	0.015625	0.7295	0.00488	0.84726	29.80312	58.87674

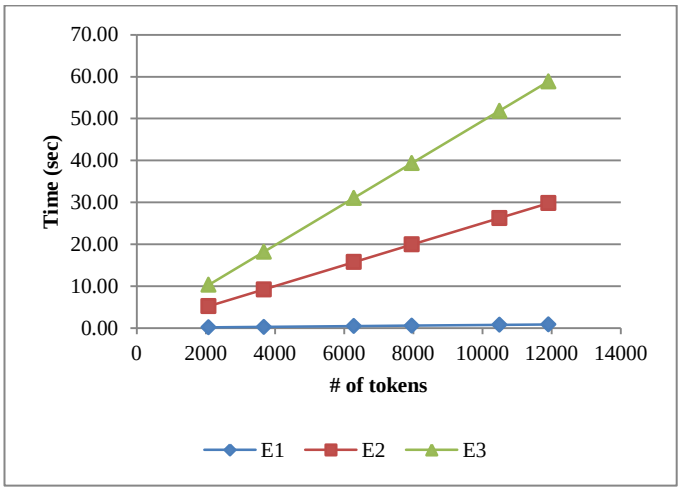

Fig. 7: Total expectation time for the three models, MDL, Global Encoding, and Accelerating XPath

V. CONCLUSIONS AND FURTHER RESEARCH

In this paper, we have introduced and implemented a new model for mapping XML documents into RDB. The model examined the problem of solving the structural hole between ordered hierarchical XML and unordered tabular RDB to enable us to use the RDBMS systems for storing, updating and querying XML data. The experiments show that the new model is high flexible for updating.

For a future research, since multi dimension links are used, an optimization of labels sizes may reduce the size of "Tokens Table" and indexes used for these links. Other performance measurement for evaluation will be considered such as querying and retrieving, storage space and mapping accuracy.

REFERENCES

[1] H. Jagadish, S. Al-khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu & C. Yu, "TIMBER: A Native XML Database", In: SIGMOD, San Diego, CA, 2003

[2] S. M. Chung & S. B. Jesurajaiah, "Schemaless XML document management in object-oriented databases", Information Technology: Coding and Computing, ITCC 2005, International Conference on, 2005, pp. 261-266 Vol. 1.

[3] K. Fujimoto, T. Yoshikawa, D. D. Kha, M. Yashikawa & T. Amagasa, "A Mapping Scheme of XML Documents into Relational Databases Using Schema-based Path Identifiers", International Workshop on Challenges in Web Information and Integration (WIRI'05), 2005, 82-90.

[4] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller & N. Westbury, "ORDPATHs: insert-friendly XML node labels", In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM, Paris, France, 2004.

[5] M. Atay, A. Chebotko, D. Liu, S. Lu & F. Fotouhi, "Efficient schema-based XML-to-Relational data mapping", Information Systems 32, 2007, 458-476.

[6] J. K. Min, C. H. Lee & C. W. Chung, "XTRON: An XML data management system using relational database", Information and Software Technology Volume 50, issue 5,2008, p. 462-479.

[7] J.-H. Yun & C.-W. Chung, "Dynamic interval-based labelling scheme for efficient XML query and update processing", Journal of Systems and Software 81, 2008, p. 56-70.

[8] P. Ahlgren & C. Colliander, "Document-document similarity approaches and science mapping: Experimental comparison of five approaches", Journal of Informetrics 3, 2009, p. 49-63.

[9] H. Zhang & F. W. Tompa, "Querying XML documents by dynamic shredding", In: Proceedings of the 2004 ACM symposium on Document engineering, ACM, Milwaukee, Wisconsin, USA, 2004.

[10] H. Jiang, H. Lu, W. Wang & J. X. Yu, "XParent: An Efficient RDBMS-Based XML Database System", ICDE, 2002, p. 335-336.

[11] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita & C. Zhang, " Storing and Querying Ordered XML using a Relational Database System", SIGMOD, 2002, p. 204-215.

[12] S. Soltan & M. Rahgozar, "A Clustering-based Scheme for Labelling XML Trees". IJCSNS International Journal of Computer Science and Network Security 6, 2006, p. 84-89.

[13] G. Torsten, M. V. Keulen & T. Jens, "Accelerating XPath Evaluation in Any RDBMS". ACM Transactions on Database Technology 29, issue 1, 2004, p. 91-131.

[14] A. Berglund, S. Boag, D. Chamberlin, M. Fernández, M. Kay, J. Robie & J. Siméon, "XML Path Language (XPath) 2.0", W3 Consortium, 2007.

[15] X. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie & J. Siméon, "XQuery 1.0: An XML Query Language". W3 Consortium, 2007.

[16] Oracle, "Oracle XML DB Developer's Guide 10g", n. a.

[17] X. Wu, M. L. Lee & W. Hsu ,"A prime number labelling scheme for dynamic ordered XML trees", Proceedings. 20th International Conference on Data Engineering, (ICDE 2004), 2004.

[18] Washington University, C. S. E. R., "XMLData Repository", 2002.

[19] R. Busse, M. Carey, D. Florescu & M. Kersten, "XMark- An XML Benchmark Project", 2002.

[20] K. Runapongsa, J. M. Patel, H. Jagadish, Y. Chen & S. Al-Khalifa, "The Michigan benchmark: towards XML query performance diagnostics", Information Systems 31, 2006, p. 73-97.