Indonesian Journal of Electrical Engineering and Computer Science
Vol. 18, No. 3, June 2020, pp. 1596~1606
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v18.i3.pp1596-1606

ORD-GAP: a hybrid-based labeling schemes to support XML dynamic updates

Aisyah Amin, Su-Cheng Haw, Samini Subramaniam
Faculty of Computing and Informatics, Multimedia University, Malaysia

Article Info

Article history:

Received Sep 13, 2019
Revised Dec 14, 2019
Accepted Dec 30, 2019

Keywords:

Dynamic updates
Labeling scheme
Mapping approach
Structural relationship
XML database

ABSTRACT

eXtensible Markup Language (XML) has been widely used as the standard for data exchange standard over the Internet. With the fast growing rate of data, especially with high updates, it is crucial to ensure that the XML is able to cope with frequent changes with very least effect on the existing structure. Therefore, in this paper, we investigate on the existing labeling schemes and mapping approaches to gauge a better understanding in terms of the robustness of the labeling schemes and the importance of the mapping schemes. Next, we propose ORD-GAP labeling schemes to identify the structural relationship among XML nodes and yet, it is persistent to re-labeling when new nodes are inserted. Subsequently, a mapping scheme is proposed to transform XML into Relational Database (RDB). Preliminary experimental evaluation demonstrated that the proposed approach achieve 66% better as compared to ORDPATH, and 56% better as compared to ME labeling in terms of data loading time.

Corresponding Author:

Su-Cheng Haw,
Faculty of Computing and Informatics,
Multimedia University, Malaysia.
Email: sucheng@mmu.edu.my

1. INTRODUCTION

Extensible Markup Language (XML) has arisen as the standard for data exchange over the Internet due to the fact that it is easily readable by human and machine, as it uses natural language for markup expression. Alternatively, Relational Database (RDB) is commonly being used as back-end in various industries. Nevertheless, due to the data is process independently of its context, RDB could not fulfill the market demand. As such, it is essential to have a good mapper for storing and retrieving XML structure (hierarchical model) into RDB (tables with rows and columns) [1]. The key criterion for a good mapper is to ensure that the four main structural relationships, i.e., Ancestor-Descendant (A-D), Parent-Child (P-C), sibling and order are preserved [2, 3]. In order to do so, a good and effective labeling scheme employed on the node (also known as node indexing) is essential [4, 5].

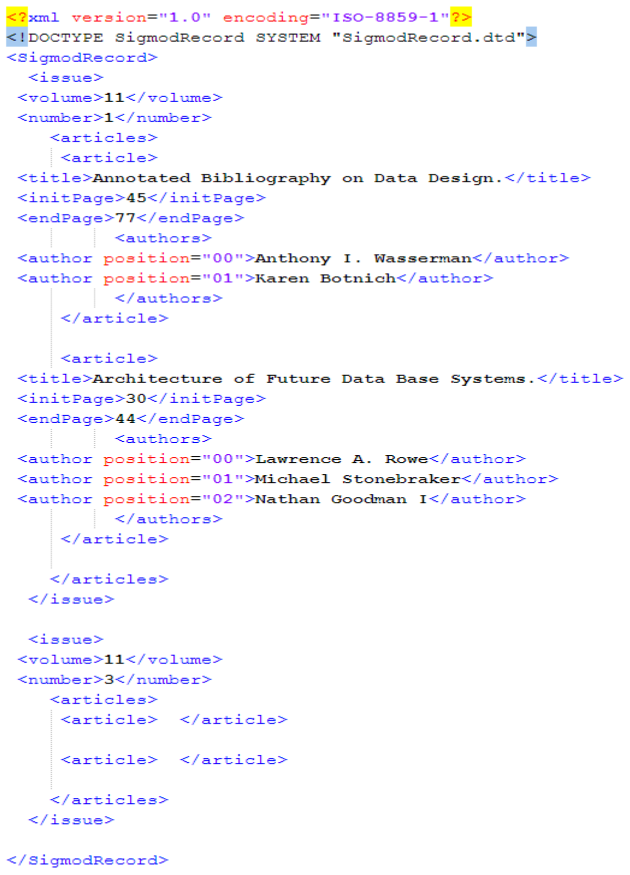
There are numbers of researches done on labeling scheme [6-9]. In this paper, we analyze some recent labeling schemes, with focus on the support during dynamic updates (insertion operation). The main types of insertion happen: (i) left-most insertion, (ii) right-most insertion, and (iii) in-between insertion [7, 8, 10]. Nevertheless, most of the current approaches still exhibit the needs of re-labeling when the reserved label has been used up. On the other hand, mapping plays a role to determine how effective a XML database has been transformed into RDB. Excessive tables after the mapping process will resulted in excessive joins; while too many information stored in a single table may not be the good solution as well [11]. Thus, it is essential to propose a new labeling and mapping scheme to ensure (i) no loss of information when transforming from XML to RDB storage, (ii) avoid re-labeling if dynamic updates happen, (iii) fast query retrieval.

The subsequent sections are organized as follows. Section 2 reviews the labeling schemes and mapping schemes. In addition, this paper also summarizes and discusses on the advantages and disadvantages of these schemes. Throughout the review, SigmodRecord dataset is used as an example. The partial view of SigmodRecord Dataset is depicted in Figure 1. Section 3 proposes our proposed approach namely as ORD-GAP labeling schemes which support dynamic updates on XML databases.

2. LITERATURE REVIEW

2.1. Review on Existing Labeling Schemes

Labeling schemes can be broadly grouped into region encoding, multiplicative, prefix-based, and hybrid-based [3]. A region-based labeling scheme utilize the tree traversal navigation to assign label on the nodes to preserve the ordering while ensuring the structural relationships are preserved. A prefix-based labeling schemes [3, 12, 13] is usually the most simple scheme as it directly encode a node's parent label as the prefix of its label. On the other hand, multiplicative labeling scheme usually assign label based on some arithmetic computation to identify the structural relationships among nodes. A hybrid labeling scheme, however, is composed of some combinations of existing scheme grouping to balance between one weakness with the strength of the other group [14].



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE SigmodRecord SYSTEM "SigmodRecord.dtd">
<SigmodRecord>
  <issue>
 <volume>11</volume>
 <number>1</number>
    <articles>
     <article>
 <title>Annotated Bibliography on Data Design.</title>
 <initPage>45</initPage>
 <endPage>77</endPage>
          <authors>
 <author position="00">Anthony I. Wasserman</author>
 <author position="01">Karen Botnich</author>
          </authors>
     </article>

     <article>
 <title>Architecture of Future Data Base Systems.</title>
 <initPage>30</initPage>
 <endPage>44</endPage>
          <authors>
 <author position="00">Lawrence A. Rowe</author>
 <author position="01">Michael Stonebraker</author>
 <author position="02">Nathan Goodman I</author>
          </authors>
     </article>

    </articles>
  </issue>

  <issue>
 <volume>11</volume>
 <number>3</number>
    <articles>
     <article>  </article>

     <article>  </article>

    </articles>
  </issue>

</SigmodRecord>
```

Figure 1. A Sample of SigmodRecord XML document

V-Containment [7] is a region encoding labeling scheme, which is based on containment label [15]. Primarily, the labeling structure contains (startV, endV, level), where by startV, endV are two vectors representing the one-time assignment of initial labeling pre/post labeling scheme, and level denotes the depth from the root node to current node. The initial labeling schemes are assigned based on depth-first traversal. Subsequently, to support dynamic updates, they proposed some algorithms to accommodate the insertion in the left-most, right-most and in-between nodes. The insertion is possible if the start and end are between the range of the previous and following siblings.

Multiplicative labeling (ME) uses multiplication operation on odd numbers to label the XML tree. It consists of (level, [selflabel, ordinal]), where by "level represent the node depth of the tree, selflabel is computed as parent *ordinal (parent is the selfLabel of parent node, and ordinal is the position of the current node within its sibling)" [8]. The root node starts as 1. The formula 2n+1 was applied to generate odd numbers, where n represent the position of a node. In addition, they also outlined some formulas to calculate new label for left-most, right-most and in-between insertion without any relabeling incurred.

Level-based labeling scheme (LLS) is a hybrid labeling scheme based on interval and prefix-based labeling scheme [16]. LLS labeling structure are assigned as <d.p.s> whereby d denote the depth of level, p (indicate as PerL) is the number of node across d level, and s is the instance serial number that recognize nodes between the same node from the same class. If an insertion happen, the re-labeling of nodes only effect on the node label of the subsequent inserted nodes as shown in Figure 2.

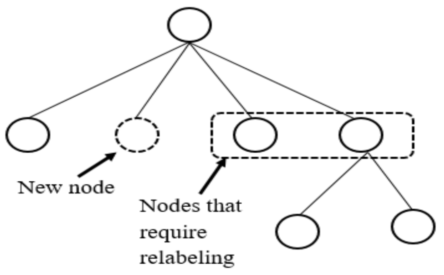


Figure 2. LLS Labeling Scheme of dynamic updates

Dynamic Prefix-based Labeling Scheme (DPLS) [13] is an example of prefix-based labeling by extending on Dewey encoding scheme [17]. This approach has two stages, whereby the first stage includes constructing the initial DPLS labeling based on Dewey scheme, followed by the next stage is to handle any updates. Similarly, this approach supports three types of insertion: left-most, in-between and right-most.

Fu & Meng [18] proposed Triple-code, which consists of <start, end, parent-id>. Their approach adopted the interval-based labeling scheme by replacing the 'level' tag with node's 'parent-id', making it simple to obtain P-C and sibling relationships. He [19] proposed prefix-based scheme using fractions, which he named it as DPESF Encoding. This labeling scheme is in Number-Character format based on the mapping rules as follows: "to map each digit $n \in N=$ {0,1,2,3,4,5,6,7,8,9} in the numerator to a matching character $c \in C=\{A,B,C,D,E,F,G,H,I,J\}$". As such, label with (12514) is expressed as BCF14.

More recently, Gopinathan & Asawa [20] proposed an extended Dewey labeling scheme, which consists of [prefix.ordinal] label to support Content and Structure Query (CAS) effectively. In addition, they also proposed new path based indexing namely, path index (p_index) and path combined index (pc_index). These indexes were constructed using B+Tree and HashMap respectively.

Ahn et al. [21] proposed to implement repetitive prime number [22] labeling in a Map Reduce-based algorithm to overcome the problem of memory insufficient should a massive XML data is loaded in a single machine. Being in parallel environment, this allows multiple machines to compute labels independently. In another research work, Mohammed et al. [23] indicated that selectivity estimation is crucial to support several types of XML query, especially those that contains wildcards or logical operators. They proposed combining prime number labeling and synopsis modeling for answering these queries.

2.2. Review on Existing Mapping Schemes

Mapping schemes are essential to transform XML into relational storage. These schemes can be grouped into path-based, edge-based and node-based technique. Zhu et al. [11] proposed Mini-XML, which is a path-based mapping scheme. Structural relations of Mini-XML consist of "(Level,[P-pathId, S-order]), where Level indicates the depth of the current node, P-pathId is the path id of direct parent node, and S-order is the position of current leaf node in direct node". Root label is (0,[0,1]). After the mapping process, there are two mapping tables namely PathTable and LeafTable. The PathTable as shown in Table 1, is used to keep all path information of inner nodes, while the LeafTable as shown in Table 2, is used to keep leaf nodes information. The authors compared the performance of Mini-XML against s-XML (citation), which indicated that Mini-XML has better performance and better storage space as well as more scalable to support huge XML data.

Table 1. Partial View of PathTable

pathID	Path	Pos
1	./SigmodRecord	(1,[0,1])
2	./SigmodRecord/issue	(2,[1,1])
3	./SigmodRecord/issue	(2,[1,2])
4	./SigmodRecord/issue/articles	(3,[2,3])
5	./SigmodRecord/issue/articles	(3,[3,3])
6	./SigmodRecord/issue/articles/article	(4,[4,1])
7	./SigmodRecord/issue/articles/article	(4,[4,2])
8	./SigmodRecord/issue/articles/article/authors	(5,[6,4])
9	./SigmodRecord/issue/articles/article/authors	(5,[7,4])

Table 2. Partial View of LeafTable

LeafID	name	value	pos
10	volume	11	(3,[2,1])
11	number	1	(3,[2,2])
12	volume	11	(3,[3,1])
13	number	1	(3,[3,2])
14	article	-	(4,[5,1])
15	article	-	(4,[5,2])
16	title	Architecture of Future Data Base Systems.	(5,[6,1])
17	initPage	30	(5,[6,2])
18	endPage	44	(5,[6,3])
19	title	Errors in 'Process Synchronization in Database Systems'.	(5,[7,1])
20	initPage	9	(5,[7,2])
21	endPage	29	(5,[7,3])
22	author	Lawrence A. Rowe	(6,[8,1])
23	author	Michael Stonebraker	(6,[8,2])
24	author	Philip A. Bernstein	(6,[9,1])
25	author	Marco A. Casanova	(6,[9,2])
26	author	Nathan Goodman	(6,[9,3])

In another approach, Qtaish and Ahmad [24] proposed XAncestor, which is also a path-based mapping technique. XAncestor contains of XtoDB, which is a mapping algorithm, and XtoSQL, which is a query retrieval algorithm and Fixed RDB, which is an integrated solution. The beauty of XAncestor approach is mapping of XML node into RDB without having the needs to map the entire document. This means that only ancestor path will be map into RDB whereas, the whole documents that consist of leaf-node and inner-nodes will be ignored. Thus, it reduces the storage space. The algorithm of XAncestor helps to keep the P-C and A-D relationships in translating SQL queries. Figure 3 illustrates in the architecture of XAncestor. Ancestor_Path of XAncestor Scheme and Leaf_Node of XAncestor Scheme as shown in Table 3 and 4.

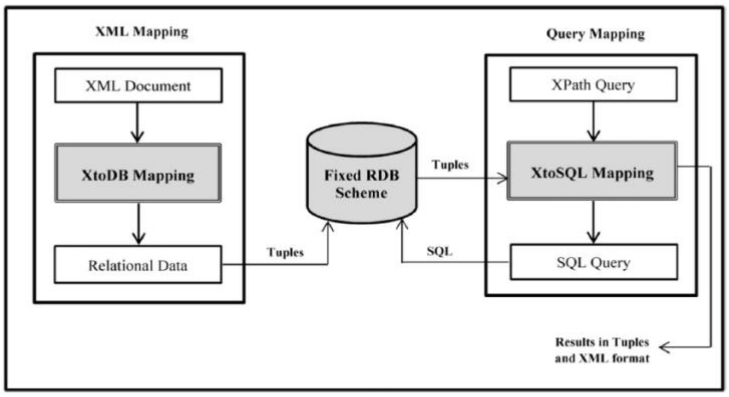


Figure 3. XAncestor mapping approach [20]

Table 3. Ancestor_Path of XAncestor Scheme

Ances_PathID	Ances_PathExp
1	/SigmodRecord
2	/SigmodRecord/issue/articles/article
3	/SigmodRecord/issue/articles/article/authors
4	/SigmodRecord/issue/articles

Table 4. Leaf_Node of XAncestor Scheme

Node Name	Ances PathID	Ances Pos	Node Value
volume	1	1.1	11
number	1	1.1	1
volume	1	1.2	11
number	1	1.2	1
article	4	1.2.3	-
article	4	1.2.3	-
title	2	1.1.3	Architecture of Future Data Base Systems.
initPage	2	1.1.3	30
endPage	2	1.1.3	44
title	2	1.1.3	Errors in 'Process Synchronization in Database Systems'
initPage	2	1.1.3	9
endPage	2	1.1.3	29
Author	3	1.1.3.1.4	Lawrence A. Rowe
Author	3	1.1.3.1.4	Michael Stonebraker
Author	3	1.1.3.1.4	Philip A. Bernstein
Author	3	1.1.3.1.4	Marco A. Casanova
Author	3	1.1.3.1.4	Nathan Goodman

3. RESEARCH METHODOLOGY

3.1. The Architecture Diagram

There framework consists of three main components: XML Parser, XML Encoder, and XML Mapper as shown in Figure 4. Firstly, the uploaded XML document will undergo XML Parser for well-formedness checking, and get validated before data exchange take place. We employed the Simple API for XML (SAX) parser for this. In XML Encoder, XML documents are encoded as a set of streams labeled with our proposed labeling scheme (describe in the next section). In XML Mapper, the XML data is mapped into RDB.

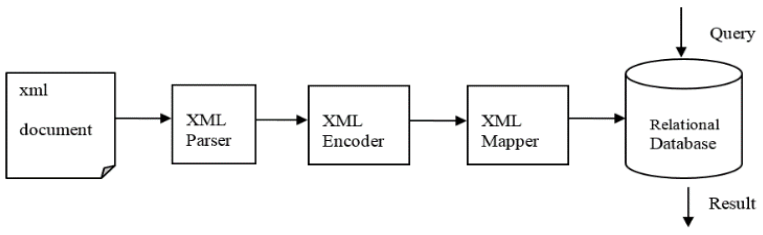


Figure 4. The Proposed Architecture

3.2. Proposed Labeling Scheme in XML Enconder

The labeling scheme is named as ORD-GAP, which comes after the approach ORDPATH by reserving gap for future insertion. The initial labeling of ORD-GAP is computed based on depth-first traversal in a form of a (s-e)l, where s and e are the start and end of the range, and l represents the level of the node. The s and e are computed based on the gap, which is result of Σ(maxfan-out + maxdepth). Figure 5 shows the partial of SigmodRecord dataset annotated with ORD-GAP. Firstly, the gap must be calculated. In this sample, the tree has the largest fan-out (maxfan-out) of 4 and deepest level (maxdepth) of 6. As such, the gap is 10.

The root node begins with 1 (for the s). Subsequently, the tree will be annotated based on depth-first traversal. As such, the s for the succeeding node, 'issue', will be assigned with the previous node's s added with the gap (in this case, it is 1+10). This is followed by 'volume', which will be assigned as 21 (11+10) for the s label. Once a leaf has been reached, the e will then be generated by adding the previous running number of s with the gap. For example, take a look into the first leaf node reached. This node has the s label of 31, thus, the e label computed will be 41. Conversely, if the node is not a leaf node, it's e label will be computed by adding the e label of the rightmost child with the gap based on the depth-first traversal.

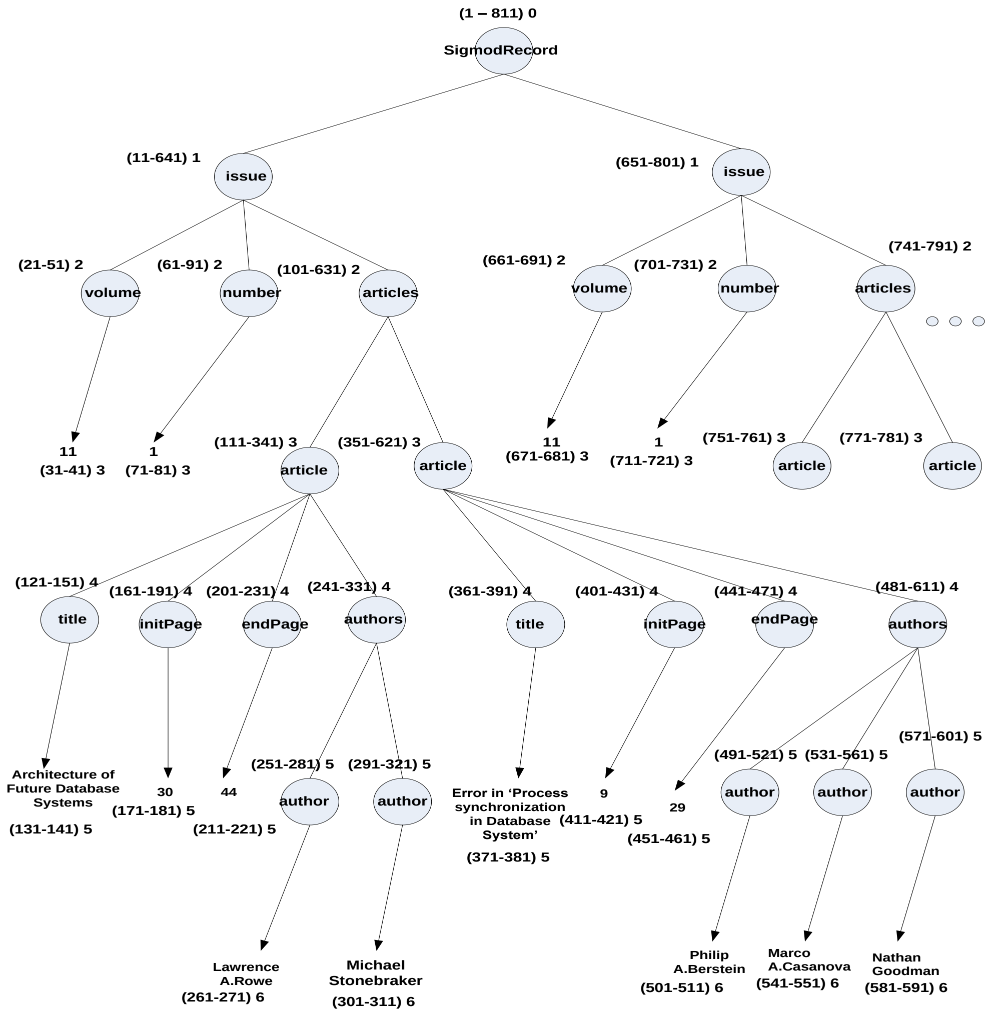


Figure 5. The initial labeling of ORD-GAP

Figure 6 shows the pseudocodes for ORD-GAP. Figure 6(a) outlines the GetGap function. It shows how the "gap" is calculated. The algorithm takes a node and the next level of the current node as input. By traversing the entire child under the particular node, it will be able to count the maximum number of child, which will then be computed as the maximum fan-out. Similarly, the maximum level will be computed as the maximum depth of the tree.

Figure 6(b) shows the AssignLabel function. Based on the "gap" calculated from GetGap function described earlier, the algorithm has the parent node, the current range, and the current level as the input. Then, it will assign the start of the label as the current range, as well as the current level. Based on depth-first traversal, it will first traverse the child nodes, and assign each node with the start and level (see Line 10). When the leaf node is reached, it will then assign the end (which is the sum of start and “gap”). This process repeats until each node is labelled.

```
    Function GetGap
    input : k - the kth node
    input : nextlevel - next level of the tree
    output: Gap used for labeling
1   MaxFanOut = 0
2   MaxDepth - 0
3   level = nextlevel + 1
4   while( k hasChild, j ) do
5     order = k's number of childs
6     if order > MaxFanOut then
7       MaxFanOut = order
8     GetGap( j, level )
9     if level > MaxDepth then
10      MaxDepth = level
11  return MaxFanOut + MaxDepth
```

(a)

```
    Function AssignLabel
    input : k - the kth node to be labelled.
    input : gap - gap used for identify the range
    input : currentRange - the current range that the algorithm
    has reached
    input : nextlevel - next level of the tree
    output: label - the label of node k.
1   if is root node then
2     start = 1
3     level = 1
4   else
5     startRange = currentRange
6     level = nextlevel
7   currentRange = start + gap
8   level = level + 1
9   While ( k has child nodes, j ) do
10    AssignLabel( j, gap, currentRange, level )
11  end
12  endRange = currentRange + gap
13  return level( startRange - endRange )
```

(b)

Figure 6. Algorithm for (a) Function AssignLabel (b) Function GetGap

3.3. Proposed Mapping Scheme in XML Mapper

While assigning the ORD-GAP labeling, the XML tree is mapped into RDB. Table 5 and Table 6 depict the mapped result. Basically, the iTable and tTable are constructed to store the internal node and text node respectively.

Table 5. iTable (Parent Table)

Start	End	Level	Pstart	Value
251	281	5	241	author
291	321	5	241	author
491	521	5	481	author
531	561	5	481	author
571	601	5	481	author
121	151	4	111	title
161	191	4	111	initPage
201	231	4	111	endPage
241	331	4	111	authors
361	391	4	351	title
401	431	4	351	initPage
441	471	4	351	endPage
481	611	4	351	authors
111	341	3	101	article
351	621	3	101	article
751	761	3	741	article
771	781	3	741	article
21	51	2	11	volume
61	91	2	11	number
101	631	2	11	articles
661	691	2	651	volume
701	731	2	651	number
741	791	2	651	articles
11	641	1	1	issue
65	801	1	1	issue
1	811	0	-	SigmodRecord

The A-D relationship is determined with the following formula:

1. *if(A(s) < D(s) < A(e)).*

Example 1: Let node1 be volume (21-51)2 and node2 be SigmodRecord (1-811)0, (SigmodRecord (1) < volume (21) < SigmodRecord (811)). As such, node1 and node2 has A-D relationship.

Table 6. tTable (Child Table)

Start	End	Level	Pstart	Value
261	271	6	251	Lawrence A.Rowe
301	311	6	291	Michael Stonebraker
501	511	6	491	Philip A.Berstein
541	551	6	531	Marco A.Casanova
581	591	6	571	Nathan Goodman
131	141	5	121	Architecture of Future Database Systems
171	181	5	161	30
211	221	5	201	44
371	381	5	361	Error in 'Process synchronization in Database System'
411	421	5	401	9
451	461	5	441	29
31	41	3	21	11
71	81	3	61	1
671	681	3	661	11
711	721	3	701	3

The proposed approach supports all structural relationships which are P-C, A-D, and sibling. The P-C relationship is determined with the following formula:

2. *if (P(s) < C(s) < P(e)) and (C(level) – P(level) = 1)*

3. *Pstart for C == Start for P (Mapping Scheme)*

The first criteria is the range, which is similar to the criteria for A-D relationship. However, it is essential to check for the level difference between the two nodes; if the difference is 1, then, the two nodes are having P-C relationship. It can also be determined easily from the table by using PStart value.

Example 2: Let node1 be volume (21-51)2 and node2 be issue (11-641)1, (issue (11) < volume (21) < issue (641) and volume (2)-article(1)=1). As such, node1 and node2 has P-C relationship.

To determine for the sibling relationship, *if the nodes have the same PStart, they are having sibling relationship.*

Example 3: Let node1 be volume (661-691)2 and node2 be number(701-731)2. From iTable, both have PStart '651'. As such, node1 is a sibling of node2.

3.4. Support for Dynamic Updates

The dynamic updates are supported in the proposed approached as there is no re-labeling required. For the updates due to insertion, we apply the ORDPATH labeling scheme. Insertion consists of inserting between the nodes, insert into the right-most and left-most nodes. For insertion involved in in-between nodes and insertion into the right-most nodes, the e value on the left sibling will be used as the prefix to the ORDPATH label. Similarly, insertion into the left-most requires the left-most sibling s value.

Figure 7 shows the example of the three types of insertion. The dotted lines and dotted circles indicate the insertion of nodes. As shown in Figure 7, the left-most insertion of a subtree on the second level, will produce a label of (21.1)2, followed by its child with label (21.1.1)3, and descendant with label (21.1.1.1)4 respectively. For the rightmost insertion, the s value is (791.1)2 as it uses the e values on the left sibling.

Meanwhile, for the insertion in-between nodes, it uses the e value of the left node (641.1)2, followed by its two children with label (641.1.1)2 and (641.1.2)2. In a way, the structure of the label remains, and the retrieval of relation between nodes still works with the insertion node.

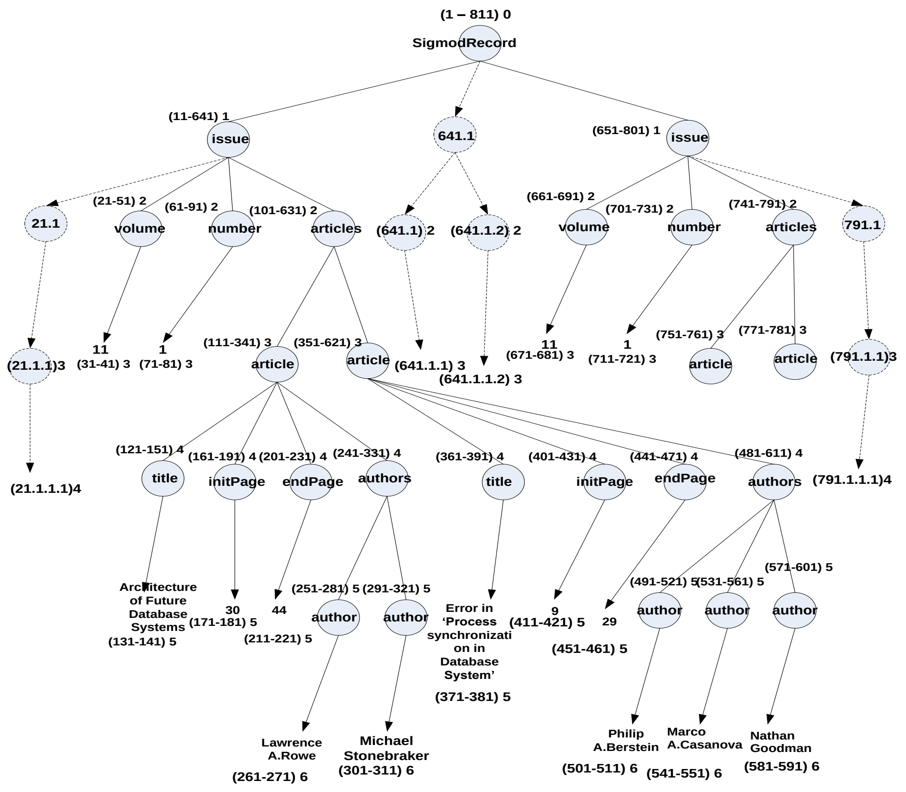


Figure 7. The proposed approach labeling scheme for insertion

4. EXPERIMENTAL EVALUATIONS AND PRELIMINARY RESULTS

In this section, the performance of the proposed approach was compared to other approaches using the dataset obtained from University of Washington repository [25] as this is the standard datasets for benchmarking. For the preliminary evaluation, we have only conducted experimental based on storing time, i.e., the data loading time. The other evaluation on the query response time will be carried out in our future work.

Table 7 shows the details of Yahoo.xml, SigmodRecord.xml and Dblp.xml datasets. The experiment was performed on a 3.40 GHz Intel (R) Core(TM) i7-3770 CPU, with 8.00 GB RAM on a Windows 7 Home Premium. In order to obtain a better consistency, the evaluations were executed for three times on each test case. The results obtained are then computed as the average of these three consecutive runs. Table 8 shows the insertion result on the three selected datasets.

From the result, it is observed that generally ORD-GAP outperformed the rest of the approaches in terms of the data loading time by 66% better as compared to ORDPATH, and 56% better as compared to ME labeling. This is due to the reason that integer computation is faster as compared to string (ORDPATH). ME labeling is also integer-based, yet, it is slower as multiplication calculation is needed to compute the label.

Table 7. Description of XML Datasets

Filename	Description	Size	Elements	Attributes	Max-Depth	Avg-depth
Yahoo .xml	Yahoo auction data	24KB	342	0	5	3.76608
SigmodRecord.xml	SIGMOD Record in XML	467KB	11526	3737	6	5.14107
Dblp.xml	DBLP Bibliography	127MB	3332130	404276	6	2.90228

Table 8. Loading Time of the Approaches

Dataset	ME Labeling (ms)	ORDPATH (ms)	ORD-GAP (ms)
Yahoo.xml (24KB)	829	967	432
SigmodRecord.xml (467KB)	15241	20581	6224
Dblp.xml (127MB)	4679892	6551554	1802553

5. CONCLUSION AND FUTURE WORKS

In this paper, we have reviewed some existing works on labeling and mapping schemes. Subsequently, we have proposed our labeling scheme named ORD-GAP, which is persistent to re-labeling, as this approach extend the ORDPATH labeling to cater for any dynamic updates. In addition, preliminary experimental evaluation has shown that ORD-GAP works well to transform XML into RDB storage in terms of the loading time. In our future works, we will further experiment with other types of datasets (in terms of size, structure of the dataset, the depth level, and so on). In addition, the key criterions for good database are, it should be able to support storage and retrieval efficiently. Thus, we will also evaluate the time taken to retrieve various types of simple and complex queries.

ACKNOWLEDGEMENTS

This work is supported by the funding of TM R&D from the Telekom Malaysia, Malaysia.

REFERENCES

[1] K. Schweinsberg, and L. Wegner, "Advantages of complex SQL types in storing XML documents", *Future Generation Computer Systems*, 2016 (in press).

[2] G.Z. Qadah, "Indexing techniques for processing generalized XML documents", *Computer Standards & Interfaces*, vol. 49, pp. 34-43, 2017.

[3] S.C. Haw, and C.S. Lee, "Node Labeling Schemes in XML Query Optimization: A Survey and Open Discussion", *IETE Technical Review*, vol. 26 (2), pp. 89-101, 2009.

[4] N. Ferro, and G. Silvello, "Descendants, ancestors, children and parent: A set-based approach to efficiently address XPath primitives", *Information Processing & Management*, vol. 52(3), pp. 399-429, 2016.

[5] J.Kopke, "Annotation paths for matching XML-Schemas", *Data & Knowledge Engineering*, 2017 (in press).

[6] E. Barros, *et al.*, "LCA-based algorithms for efficiently processing multiple keyword queries over XML streams", *Data & Knowledge Engineering*, vol. 103, pp.1-18, 2016.

[7] Z. Qin, *et al.*, "Efficient XML query and update processing using a novel prime-based middle fraction labeling scheme", *China Communications*, vol. 14(3), pp. 145-157, 2017.

[8] S. Subramaniam, and S.C. Haw, "ME Labeling: A Robust Hybrid Scheme for Dynamic Update in XML Databases", *IEEE International Symposium on Telecommunication Technologies,* 2014.

[9] J. Liu, *et al.*, "Efficient labeling scheme for dynamic XML trees",. *Information Sciences*, vol. 221, pp. 338-354, 2013.

[10] P. Fraigniaud, and A. Korman. "An Optimal Ancestry Labeling Scheme with Applications to XML Trees and Universal Posets", *Journal of the ACM*, vol.63 (1), pp. 6:1-6:30, 2016.

[11] H. Zhu, *et al.*, "*Mini-XML: An efficient mapping approach between XML and relational database*", IEEE/ACIS 16th International Conference on Computer and Information Science, 2017.

[12] T.A. Ghaleb, and S.Mohammed, "A Dynamic Labeling Scheme Based on Logical Operators: A Support for Order-Sensitive XML Updates", *Procedia Computer Science*, vol. 57, pp. 1211-1218, 2015.

[13] J. Liu, and X.X. Zhang. "Dynamic labeling scheme for XML updates", *Knowledge-Based Systems*, vol. 106, pp. 135-149, 2016.

[14] S.C. Haw, and A. Amin, "Node Indexing in XML Query Optimization: A Review", *Indian Journal of Science and Technology*, vol. 8(32), pp. 1-9, 2015.

[15] C. Zhang, *et al.*, "*On supporting containment queries in RDB management systems*", ACM SIGMOD International Conference on Management of Data, pp. 425-436, 2001.

[16] S. Mohammad, and P. Martin, "*LLS: Level-bases Labeling Scheme for XML Databases*", Conference of the Center for Advanced Studies on Collaborative Research, 2010, pp. 115-127.

[17] I. Tatarinov, *et al.*, "Storing and Querying Ordered XML using a RDB System", *ACM SIGMOD*, pp. 204-15, 2002.

[18] L. Fu, and X, Meng, "*Triple Code: An Efficient Labeling Scheme for Query Answering in XML Data*", Web Information System and Application Conference, pp. 42-47, 2013.

[19] Y. He, "*A Novel Encoding Scheme for XML Document Update-supporting*", International Conference on Advances in Mechanical Engineering and Industrial Informatics, pp. 1844-1849, 2015.

[20] D. Gopinathan, and K. Asawa, "New Path Based Index Structure for Processing CAS Queries over XML Database", *Journal of Computing and Information Technology*, vol. 25(3), pp. 211-225, 2017.

[21] J. Ahn, et al.,"A dynamic and parallel approach for repetitive prime labeling of XML with MapReduce", *The Journal of Supercomputing*, vol. 73(2), pp. 810-836, 2017.

[22] D.H. Sun, and S.C. Hwang, "A labeling methods for keyword search over large XML documents", *Journal of KIISE*, vol. 41(9), pp. 699-706, 2014.

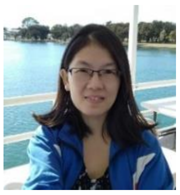
[23] S. Mohammed, A.F. Barradah, E.M. El-Alfy, "Selectivity estimation of extended XML query tree patterns based on prime number labeling and synopsis modeling", *Simulation Modelling Practice and Theory,* vol. 64, pp. 30-42, 2016.

[24] A. Qtaish, and K. Ahmad, "XAncestor: An efficient mapping approach for storing and querying XML documents in relational database using path-based technique", *Knowledge-Based Systems*, vol. 114, pp. 167-192, 2016.

[25] UW, 2018. http://www.cs.washington.edu/research/xmldatasets/www/repository.html

BIOGRAPHIES OF AUTHORS

Aisyah Mohd Amin is a Master student at Multimedia University. Her research field are XML node labeling, relational databases, dynamic updates and query optimization.



Su-Cheng Haw is Associate Professor at Faculty of Computing and Informatics, Multimedia University, where she leads several funded research on the XML databases. Her research interests include XML databases, query optimization, data modeling, semantic web, ontology, data management, and data warehousing. She has published more than 120 articles in reputable journals and conferences.

Samini Subramaniam is a lecturer and a PhD student at Multimedia University. Her research areas are XML technologies, relational databases, query optimization, distributed query processing and IoT. Samini has led a few funded projects on XML and IoT. Samini is a certified trainer and has been providing trainings on SQL and NoSQL databases to various organizations. She is also an active author who has published articles in many reputable conferences and journals.