A Clustering-based Scheme for Labeling XML Trees

Sadegh Soltan,[†] *and Masoud Rahgozar*[††]*,*

University of Tehran

Summary
Tree labeling plays a key role in XML query processing. In this paper, we propose a new labeling scheme, called *Clustering-based Labeling*. Unlike all previous labeling methods, In this labeling scheme elements are separated into various groups, and a label is assigned to a group of elements instead of one element. Based on *Clustering-based Labeling* we design a new relational schema, similar to OrdPath scheme, for storing XML documents in relational database. Grouping Sibling nodes into one record reduces number of relational records needed for XML document storage.
Our experimental results shows that our storing scheme significantly is better than tree well-known relational XML storing methods in terms of number *of stored records, document reconstruction time* and *query processing performance*.
Key words:
XML, Tree Labeling, Query Optimization.

Introduction

With the increasing popularity of XML, the need for efficient storage and retrieval of XML documents becomes more and more important. Using relational databases to manage XML data is an attractive option[6],[7],[9]. Mapping data from XML tree structure into relational tables and finding relation between different elements, to answer queries, are the core tasks for these types of databases. Every XML document can be represented by a rooted, labeled tree[2]. There is one node in XML tree for each document's element, attribute and value. Relation between nodes is presented by tree edges. In a relational database after the storage of an XML document, relation between elements is reserved by node labels. Labels show the exact position of each element in the XML tree (document). Every query processing algorithm deeply depends on the way in which XML tree is labeled Many XML tree labeling scheme have been proposed [14],[15][16],[18],[9]. Some methods use simple ways for obtaining relation between elements. For example in [8] Parent-child relation is found by 2-tuple label (ChildId ,ParentId) or in [7], each label consists of a 2-tuple (start, end). Start and End show the occurrence position of each element in XML document. Recent approaches are focused on labeling to help processing of XML queries. For example in [18], each element label provides the path from the root to that element. There is one common aspect for all previous labeling methods. In all previous methods labels are assigned to all elements. It means there is one label for each element. Assigning a separate label for each element and storing each element in separate record slows down the process of finding elements relations.
In this paper, we propose a new clustering-based scheme for XML tree Labeling. In this scheme, in contrast with other methods, a group of element is labeled instead of a single element. Putting all sibling elements into one group, and assigning labels to each group is the base idea of clustering-based labeling. This method of labeling improves the process of finding element relations.
To show the effect of our method in XML document storage and XML query processing, we design a new relational schema similar to OrdPath Schema. Then we compare our scheme to three famous relational-based XML storage method: *XRel*, *XParent* and *OrdPath*. This comparison is performed in the term of number *of stored records*, *document reconstruction time* and *query processing performance*. Clustering-based approach saves more storage space and performs much better in parent – child and structural queries compare to other schemes.
Organization The rest of the paper proceeds as follows: First we discuss preliminaries in section 2. The *Clustering-based labeling* scheme is presented in section 3. We discuss about implementation and experimental results in section 4 and conclude this paper in section 5.

1. Preliminaries

1.1 XML Tree Model and Dewey Labeling

Every XML document can be modeled as a rooted, labeled Tree. Fig. 1 presents a simple XML tree with Dewey labeling. Dewey labeling is proposed in [17] for labeling XML trees. In this method, each node label is a combination of its parent label and an integer number. If *u* is the *x-th* child of *s* in XML tree then label of *u*, label(*u*), is concatenation of label of *s* and *x* which is presented as label(*s*).*x*. For example if element label for *u* is 2.5.3 then its 5th child label will be 2.5.3.5. The advantage of this

Manuscript received September 5, 2006.
Manuscript revised September 25, 2006.

method is that for any element label, we can easily extract node labels of its ancestors. For instance if an element label is 5.2.3.1 then its parent label is 5.2.3, its first ancestor label is 5.2 and so on.

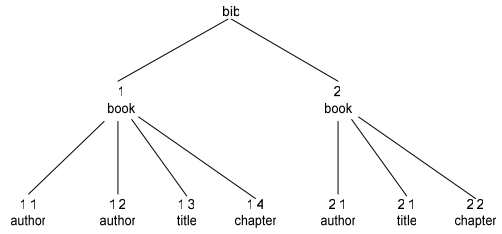

Fig. 1 XML tree with Dewey Labels

1.2 XML Storage in Relational Databases

Storing XML documents in relational databases is one of most common methods for XML storage. Many relational schemas have been proposed but most of them are not efficient enough to process XML Queries [6], [7], [12]. XRel, XParent and OrdPath are recent methods which have good performance in relational XML storage an retrieval. In this section we briefly describe Relational schema and labeling scheme for each of these methods.

1.2.1 XRel

In This method [7], four tables are used for storing XML documents. XRel relational schemas is presented in Fig. 2

Path (PathID, PathExp)
Element (DocId, PathID, Start, End, Index, Reindex)
Text (DocId, PathID, Start, End, Value)
Attribute (DocId, PathID, Start, End, Value)

Fig. 2 XRel relational schema

Elements, attributes and text data are stored in *Element*, *Attribute* and *Text* table. All document paths (path means the sequence of elements from root to any element) are stored in Path table. Table 1 shows different paths extracted from XML tree of Fig. 1.

Table 1 : Path table for XML tree Fig. 1

Paths
Bib
Bib/book
Bib/book/author
Bib/book/title
Bib/book/chapter

The idea of saving all XML paths in one table is first proposed in this paper for better process of XML queries. Relations between nodes are extracted from Start and End fields.

1.2.2 XParent

Like XRel, this method also uses four tables [8]. Parent relational schema is showed in Fig. 3.

LabelPath (ID, Len, Path)
DataPath (Pid, Cid)
Element (PathID, Did, Ordinal)
Data (PathID, Did, Ordinal, Value)

Fig. 3 XParent relational schema

Schema is very similar to XRel but relation between XML objects is derived from distinct table, called DataPath. In *DataPath* table all relations saved as (Pid, Cid), which are stands for ParentId and ChildId.

1.2.3 OrdPath

In OrdPath method [9] XML data are stored in two tables. Fig. 4 shows the OrdPath relation schema .In OrdPath paper, new XML tree labeling scheme have been proposed by the authors. The labeling method is insert-friendly version of Dewey tree labeling. Dewey Labeling suffered from Problem of dynamic updating. After inserting a new node into the XML tree labeled by Dewey, many nodes should be relabeled.

Node (OrdPathCode, Tag, NodeType, Value, PthID)
Path (PthID, PathExp)

Fig. 4 OrdPath relational schema

In OrdPath method this problem is solved by reserving even numbers for future insertion. Fig. 5 shows an XML tree with OrdPath labels. Only odd numbers are used for labeling original tree. After inserting new node an even number will be used as an intermediate node and the new node will be the child of intermediate node.

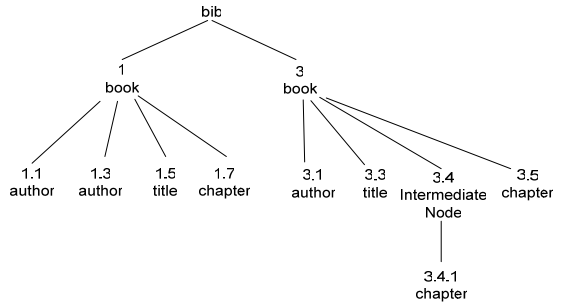

Fig. 5 XML tree with OrdPath Labels

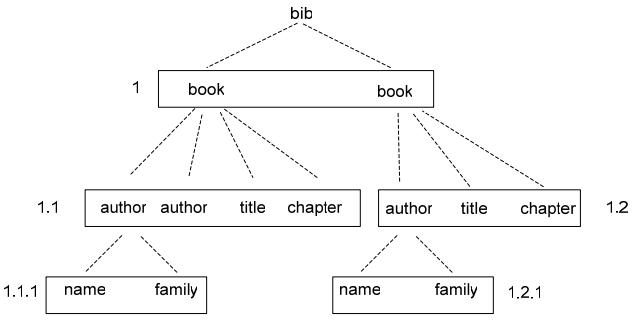

Fig. 6 An XML tree with clustering-based labels

2. Clustering-based Labeling

In all previous labeling methods a unique code is assigned to each element. Using one code per element causes some problems in large XML documents. Two main problems are: 1 number of storage records becomes very large. 2 to find a node related to some other node many node labels should be scanned.

We proposed a new labeling method to overcome these problems. In clustering-based method every sibling node is clustered into a group and the label is assigned to that group. Each group label is consisted of its parent group label and a number called *rank*. *Rank* shows the position of parent node in parent group. With this method of labeling, we can put many elements together in one unit of storage. Finding relation between siblings become faster, compare to other methods. Fig. 6 presents this labeling method. An XML tree is labeled with clustering-based method as follows:

1. Root label is null.
2. All child of root node get labeled with number one ("1").
3. for any other node, *x*, if *x* is the R_{th} element in the group with the label "L" then, all of the child of *x* are clustered into one group and will Labeled "L.R".

For example suppose group with label "1.2.1" in fig. This group is child of "author" element because prefix label("1.2") shows that parent of this group is in the group with "1.2" label and postfix label "1" shows that parent of this group is the first element in parent group . The "author" is the first element in "1.2" group so its parent of "1.2.1" group.

3. Clustered Relational XML Storage and Retrieval

3.1 Clustered Storage

In this section we propose relational schema for storing XML documents in relational database with the use of clustering labeling. Schema is similar to the OrdPath schema but sibling elements are clustered into one record. XRel relational schemas is presented in Fig. 7.

Node	(Cb, Tags, NodeType, Value, PthID)
Path	(PthID, PathExp)

Fig. 7 Clustered relational schema

Cb denotes clustering-based label. Like *OrdPath*, path from root to any element is stored in *path* table except that for clustered elements, PthID is pointed to one upper level path. Table 2 presents the *node* and *path* relational table in after storing XML tree of Fig. 6 .

Table 2: Node table for clustered storing method

Cb	Tags	NodeType	Value	PthID
Null	#,bib	Element	-	1
1	#,book#, book	Element	-	1
1.1	#,author#, author#, title #$XML database#, chapter	Element	-	2
1.2	#,author#, title#, chapter	Element	-	2
1.1.1	#,Name#, family	Element	-	3
1.2.1	#,Name#, family	Element	-	3

Table 3 : Path table for clustered storing method

PthID	PathExp
1	#/bib
2	#/bib#/book
3	#/bib#/book#/author
4	#/bib#/book#/title
5	#/bib#/book#/chapter
6	#/bib#/book#/author#/name
7	#/bib#/book#/author#/family

Text values are stored beside their parent elements. For example, in record with "1.1" label, text value "XML database "is stored beside "title" element and it's separated by "#$". Elements are separated by "#,".The reason that we don't choose only "," or "$" for element separation is the problem which may be accrued during string matching. Suppose we want to find all groups which have both "author" and "chapter" elements. If we don't use the "#" for separation And performing Sql operator "like %/chapter%author" on the *tags* field then incorrectly, groups with the "chapters" or the "authors" tags will be also returned as a query result. By using the "#," for separation and performing "like %#,chapter#%,author#", this problem will be eliminated.

3.2 Query Processing

Simple queries with "/" or "//" can be easily answered by searching the *path* and the *node* table and then performing some string matching on the *PathExp* field. Queries with predicates which the node relations are parent-child relation also can be answered easily by performing string matching on the *tags* field. Processing Queries with predicates and ancestor-descendent relations is like the methods discussed in section 1.2. Here Queries should be divided to the small, distinct, path expressions. After finding answers for each path expression, a join is executed on the candidate answers to find the final answer. In chapter 4 we will show that our method act much better compared to other methods on parent-child and structural queries.

4 Experimental Results

We have implemented clustered Storage method, and performed a series of performance experiments in order to check the effectiveness of the method. In this section, we report the outlines of the implementation and the experimental results.
We used PC with (Pentium 4 3.0 GHz Dual Core Cpu,1 GB memory and 80GB SATA 7200 RPM disk) running Windows XP+ Service Pack 2 and Microsoft SQL Server 2000 (Personal Edition). We used sun JDK 1.4.2 and SAX parser for loading XML documents into relational tables. Data set is SigMod XML document (467Kb ,year:2001) .
We compare our method to XRel, XParent and OrdPath methods with three aspects of : number of stored records, document reconstruction time and query processing performance.

4.1 Number of Stored Records

Table 4, shows the number of stored records in each four relational storage methods. Grouping sibling elements, into one record effectively decreases number of records for clustered method.

Table 4: Comparison of records number in four methods

Schema	Table Name	Num of Records
XParent	LabelPath	12
	DataPath	15,262
	Element	15,263
	Data	12,113
XRel	Path	12
	Element	11,526
	Text	11,526
	Attribute	3,737
OrdPath	NodeTable	15,263
	PathTable	12
Clustered	NodeTable	6880
	PathTable	12

4.2 Document Reconstruction Time

Document reconstruction time, is the time needed for building the original XML document from data, stored in the relational tables. Fig. 8 shows the elapsed time for reconstruction SigMod XML document in each method.

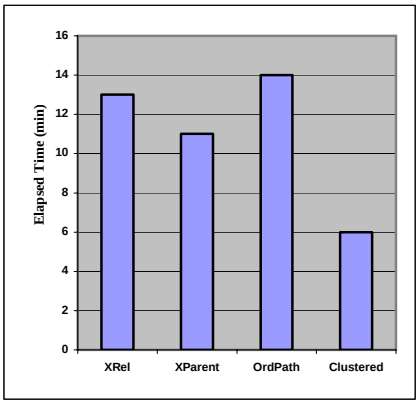

Fig. 8 Elapsed time for document reconstruction

In document reconstruction we need to scan whole database and step by step building XML document. With clustered storage we access the child of a node by only one record access. So compared to other methods in each step bigger part of XML tree can be built in our method. This reduces the time needed for document reconstruction.

4.3 Query Processing Performance

To evaluate query processing performance, we measure logical read for six types of queries. Table 5 shows the queries which are used for performance evaluation.

Table 5 Queries for evaluation of query processing performance

	Query	Specification
Q1	/sigmodRecord/issue	Simple path
Q2	/sigmodRecord/issue/articles/article/authors/author	Long simple path
Q3	//article/title	One "//"
Q4	//articles//author	Two "//"
Q5	/sigmodRecord/issue/articles/article[title='Database Directions III Workshop Review.']/authors	Predicate
Q6	/sigmodRecord/issue/articles/article[//author=' Michael Stonebraker']/title	Predicate with "//"

To answer Q1 and Q2 we first find path ids for "/sigmodRecord/issue" and "/sigmodRecord/issue/articles/article/authors/author" in the *Path* table . Next we find records in the *Node* table which has path ids equal to path ids found in *Path* table.
Answering to Q3 and Q4 is similar to Q1 and Q2 in spite of we should use Sql "like" operator instead of "=" operator(for example like '% #/article#%/author' for Q4). Answering to Q5 and Q6 is more complicated. Fig. 9 shows the tree representation of Q5 and Q6 queries.

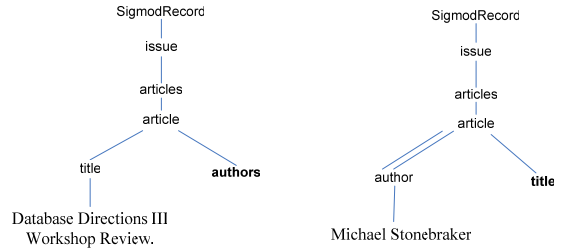

Fig. 9 Tree representation of Q5 and Q6

In Q5 we ought to find the authors names for the article which is titled "Database Directions III Workshop Review" . To answer this query in our method, first we find all the element groups in *Node* table which their path is "/SigmodRecord/issue/articles/article". After that we should check if "title" element with text value, " Database Directions III Workshop Review " and "author" element exist in founded groups or not. We check this by performing Sql command:
"Like %,title#$Database Directions III Workshop Review %,author" on *tags* field.
In other three methods, records with :
"/SigmodRecord/issue/articles/article/title",
"/SigmodRecord/issue/articles/article/author" and
"/SigmodRecord/issue/articles/article/" paths
are extracted. Then we join these records to answer the query.

In Q6 we should find title of the article which its author is "Michael Stonebraker". Answering Q6 in XRel, XParent a OrdPath and our method is similar to Q5 except that relation between the article and the author is ancestor-descendent relation and we should use *like* operator for this path.
Fig. 10, Fig. 11, Fig. 12 shows the logical read comparison in four discussed methods.

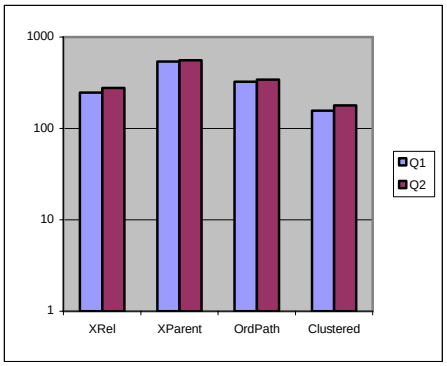

Fig. 10 Logical read for Q1 and Q2

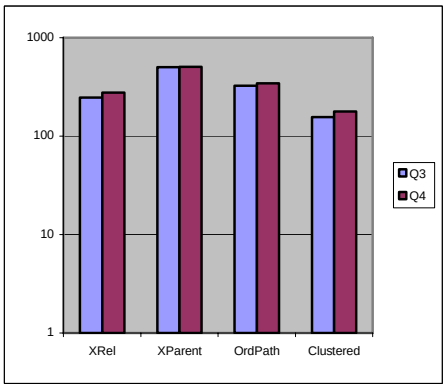

Fig. 11 Logical read for Q3 and Q4

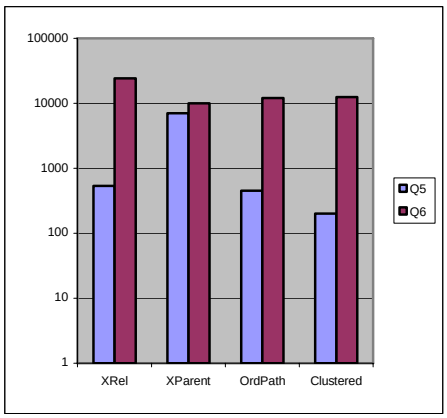

Fig. 12 Logical read for Q5 and Q6

Due to the easy access to sibling elements in clustered storage, queries: Q1 through Q5 were executed with fewer logical read. Besides, storage of the text values in the same record of their parents significantly decrease the number of logical read for accessing text data.
However, as the relationships between the elements in Q6 are ancestor-descendent, the execution of this query had more logical read compare to other methods.

5 Conclusion and Future Work

XML tree labeling is a key issue for XML query processing. In this paper we proposed a novel clustering-based XML tree labeling. This labeling method enabled us to cluster sibling elements into one group and store them in one relational record. To evaluate performance of our work we conducted some experiences in a sample data collection. Results showed that numbers of relational records were notably reduced in clustered storage compare to other schemas. Document reconstruction is much faster than other approaches. We also showed that this method of storage reduces the number of logical reads needed for query processing in parent-child and structural queries.
Our schema storage was similar to OrdPath method schema. We plan to design new schema which fits to our labeling scheme. Compression of the clustered elements and reserving extra space for dynamic updates are among our future work.

Acknowledgments

This work is supported by grants from TAKFA (National Information and Communication Technology Agenda; High Council of Informatics; Iran).

References

[1] François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler. “Extensible Markup Language (XML) 1.0,” (3rd edition) W3C Recommendation 4 February 2004.
[2] Document Object Model (DOM) Events Specification, Version 1.0 W3C Recommendation 13 November, 2000
[3] Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, Norman Walsh. “XQuery 1.0 and XPath 2.0 Data Model”, W3C Working Draft, last release 23 July 2004
[4] Jayavel Shanmugasundaram, H. Gang, Kristin Tufte, Chun Zhang, David DeWitt, Jeffrey F. Naughton, “Relational Databases for Querying XML Documents: Limitations and Opportunities,” Proc. Of 25th Intl. Conf. on Very Large Data Bases (VLDB), Edinburgh, Scotland, UK, pp. 302-314, September 1999.
[5] P. Bohannon, J. Freire, P. Roy, J. Simeon, “From XML schema to relations: a cost-based approach to XML storage,” Proc.18th ICDE 2002, San Jose, California, USA, pp. 64 -75, March 2002.
[6] D. Florescu and D. Kossman. Storing and Querying XML Data using an RDBMS. DataEngineering Bulletin, 22(3), 1999.
[7] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, Shunsuke Uemura: XRel: a path-based approach to storage and retrieval of XML documents using relational databases. ACM Trans. Internet Techn. 1(1): 110-141 (2001)
[8] Haifeng Jiang, Hongjun Lu, Wei Wang, Jeffrey Xu Yu: XParent: An Efficient RDBMS-Based XML Database System. ICDE 2002: 335-336
[9] O’Neil, E.; O’Neil, P.; Pal, S.; Cseri, I.; Schaller, G.; Westbury, N.: ORDPATHs: Insert-Friendly XML Node Labels. ACM SIGMOD Industrial Track, 2004
[10] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In VLDB, 1999.
[11] Cohen, E.; Kaplan, H.; Milo, T.: Labeling Dynamic XML Trees. In Proc. of PODS 2002
[12] Supporting Efficient Streaming and Insertion of XML Data in RDBMS, Timo Böhme, Erhard Rahm
[13] Q. Li & B. Moon, “Indexing and Querying XML Data for Regular Path Expressions”, Proceeding of 27th VLDB Conference, 2001, pp. 361-370.
[14] E. Cohen, H. Kaplan, T. Milo, “Labeling Dynamic XML Trees”, Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 1992, pp. 272-281.
[15] J. Lu & T.W. Ling, “Labeling and Querying Dynamic XML Trees”, APWeb, LNCS 3007, 2004, pp. 180-189.
[16] X. Wu, M.L. Lee, W. Hsu, “A Prime Number Labeling Scheme for Dynamic Ordered XML Trees”, Proceedings of the 20th Int Conference on Data Engineering, 2004.
[17] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasun- daram, E. J. Shekita, and C. Zhang:. Storing and querying ordered XML using a relational database system. In *Proc. of SIGMOD*, pages 204{215, 2002.
[18] From Region Encoding To Extended Dewey: On Efficient.. - Lu, Ling, Chan, Chen (2005)