Using Index to Manage the Ordering Structure of XML Data in the Nested Relational Sequence Database System

Lau Ho Lam and Wilfred Ng

Department of Computer Science

The Hong Kong University of Science and Technology, Hong Kong

lauhl@ust.hk wilfred@cs.ust.hk

ABSTRACT

In this poster, we introduce the mechanism of handling the ordering structure of XML data by using the Nested Relational Sequence Database System (NRSD), which is built upon the Nested Relational Sequence Model (NRSM) we developed earlier on. We demonstrate that the storing and querying of XML data are desirable over an NRS relation, which we incorporate an index system into the NRSD. Our preliminary experimental results show that the NRSD provides benefits of minimizing the storage size while maintaining the following three types of order in XML data: the ancestor order, the sibling order and the value order.

Keywords

Nested Relation Sequence Model, XML Databases, XML Query.

1. INTRODUCTION

We have proposed the *Nested Relational Sequence Model* (NRSM) in [2, 3], which is an extension of the well-established *Nested Relational Data Model* (NRDM) [4] in order to cater for nesting structure and node ordering of XML data. The NRSM supports composite and multi-valued attributes, which are essential for representing hierarchically structured data objects in XML documents. In addition, the NRSM extends the NRDM to support ordering of XML data elements by allowing *nested tuple sequences* to be defined in an NRS relation.

We developed the Nested Relational Sequence Database System (NRSD) [2,3], which is built upon the NRSM. An important feature in the NRSD is that XML data that has the same label along the same path can be merged together and stored in the same index table. This eliminates a substantial amount of redundancy of XML data in the database. Another feature of the NRSM is that it preserves the following three types of order: the value, sibling and ancestor orders. Figure 1 illustrates an XML tree and its corresponding NRS relation showing these three types of order. In the sequel we will explain how these orders are preserved in the NRSD by using an index system. We also present the experimental results concerning the resources needed for storing XML documents in the NRSD.

2. INDEX AND ORDER

An NRS relation is stored as a corresponding set of *index tables* in the NRSD. The schema of the NRS relation is mapped as a *global index table* and the data value are mapped as various *value index tables*. The NRS relation in Figure 1 is mapped into the tables shown in Figure 2.

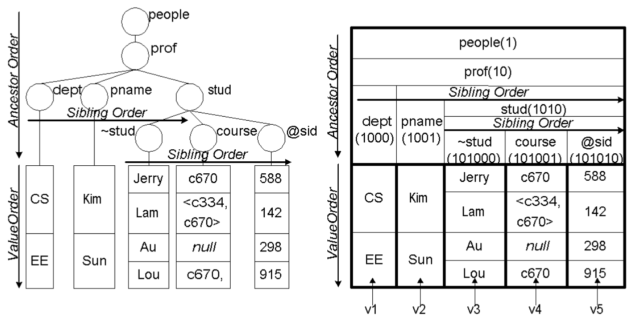

Figure 1. A XML tree and its corresponding NRS relation.

global

index	name	child
1	people	2
2	prof	8,9,10
8	dept	v1
9	pname	v2
10	stud	40, 41, 42
40	~stud	v3
41	course	v4
42	@sid	v5

v1

index	name
1.1.1	CS
1.1.2	EE

v2

index	name
1.1.1	Kim
1.1.2	Sun

v3

index	name
1.1.1.1	Jerry
1.1.1.2	Lam
1.1.2.1	Au
1.1.2.2	Lou

v4

index	name
1.1.1.1	c670
1.1.1.2.1	c334
1.1.1.2.2	c670
1.1.2.2	c670

v5

index	name
1.1.1.1	588
1.1.1.2	142
1.1.2.1	298
1.1.2.2	915

Figure 2. The index tables corresponding to the NRS relation given in Figure 1.

The ordering feature of data is an important aspect in an XML document. However, existing database systems provide inadequate facilities to manage the order structures in XML data. In the NRSD, we use indexes to keep track the order of XML data. In [1], the order between nodes is represented in a separate column with the index. In our approach, we capture different kinds of order by generating distinct and meaningful index value and store all the index values in a single column, So our approach can further reduce the storage size compared with the mapping suggested in [1]. The NRSM incorporates three types of order: (1) the *value order* resulting from the sequence of data elements in a data node; (2) the *sibling order* resulting from the left to right sides for those label nodes which share the same parent; (3) the *ancestor order* resulting from the tree levels of the label nodes. When we map XML documents into RDBMS, we use two mechanisms to handle these three types of order, which are supported by the following tables: one ordering scheme is stored in the *global index table* and another one is stored in a collection of *value index tables*.

The *global index table* represents the overall structure of an XML document. We assign the root tag with index 1. For each child tag, we assign $\lceil \log_2 k \rceil$ bits to represent its sibling order, where k is the number of its sibling nodes, and concatenate it after the index of its parent. For example, in Figure 1, "*stud*", whose index is 1010, has three children, then we assign two bits for each of its children, therefore, the binary indexes of its children is 101000, 101001 and 101010 and we have the corresponding decimal indexes 40, 41 and 42 as the child of "*stud*" in the *global index table* as shown in Figure 2. We do not need to search the whole index but just the prefix of an index. If necessary we can incrementally extend the prefix until we can identify a unique node. When we look up for the parent node, we can simply use the longest bit match algorithm to find out the corresponding parent. By using this ordering scheme, we maintain the ancestor order and the sibling order of the XML document in a straightforward way.

In the *value index tables*, the indexes are assigned by using the dot notation. We illustrate the idea by using the example shown in Figure 2: the index for the "*stud Lam*" in *v3* is *1.1.1.2*. From the g*lobal index table*, we can trace that *v3* has the path "*people/prof/stud/~stud*". The index "1.1.1.2" means that the value "*Lam*" belongs to the first "people" tag in the XML document, the first "prof" tag of "people" and so on.

3. QUERYING IN THE NRSD

We now discuss how the index system helps to process querying XML data in the NRSD. We have defined a set of NRS operations in the papers [2,3] for querying XML data stored in the NRSM. Basically, the implementation of the NRS operations in the NRSD is translated into the following sequence of actions. (1) Lookup the corresponding row by tracing the path expression. (2) Perform queries over the reference of the *child* attribute. (3) Return the data required.

Since the processing of queries in the NRSD requires referencing the *global index table* frequently, to avoid heavy overhead of this process, we create an in-memory tree for representing the structure of an XML document before querying. In general, the size of structure is much smaller than the size of data. For example, the size of schema of a 28M DBLP XML document is only about 1KB. Storing this global information as an in-memory tree does not impose any significant burden on memory, since in general we store just less then 0.1% of the given XML document.

For example, if we perform the project operation on "*stud*", $\pi_{[/prof/stud]}$(people). We lookup the child of "*stud*" and know that it has three children: 40, 41 and 42. Therefore, we need to join the values from these three *index tables* by their indexes. Note that, during our mapping, the value belongs to the same tuple are assigned with same index prefix but different indexes. For example, the two "*course*" of "*stud Lam*" are assigned with indexes "1.1.1.2.1" and "1.1.1.2.2". So when we retrieve the two "*course*" of "*stud Lam*", the value order can be correctly preserved.

4. EXPERIMENTAL RESULTS

In order to show the effectiveness of our system, we have been running some experiments using real life DBLP [5] XML data on the NRSD. All queries are conducted on a computer of *Pentium III* 550MHz with 256MB RAM and 30GB hard disk. We observe that in DBLP XML documents, the maximum number of child tags per element is twelve and the deepest nesting level is four.

In the experiments, we load XML documents having different sizes into NRSD. Table 1 shows the results of our experiments, from which we can check that the table spaces required for storing the XML documents is approximately 85% of the original size of the documents. With the growth of size, the number of index tables and the size of the NRS schema become stable. It is due to the fact that the NRSM collapses data having the same tags label into the same data nodes, resulting in minimal data redundancy. In addition, the combination of index and order in handling XML data further help to reduce the storage.

Table 1. Experimental results of NRSD with different XML documents

Size of NRS schemas (bytes)	Number of index tables	Input XML file size (KBs)	Required disk space to store the input data
414	23	2,404	82.65%
1368	61	9,318	87.56%
1380	63	14,251	87.00%
1404	66	18,817	88.87%
1425	67	28,791	83.62%

We remark that the size reduction is obtained without performing any compression on the database. This work can serve as a starting point for applying existing XML data compression technology on NRS databases. However, we emphasize that we are able to formulate queries by using a set of algebraic operators, which is difficult to perform in a compressed domain. We are improving the grouping algorithm and trying to further decrease the table space of storing XML data in the NRSD. The data size reduction in the NRSD is useful in practice for exporting and exchanging XML database objects on the Web.

5. CONCLUSIONS

In this poster, we discussed two important issues related to the use of indexes in the NRSD: (1) how to manage different kinds of order in XML data and (2) how does it affect the storage size in NRS databases. We explained the relationship between the indexes and value order and presented the empirical result, which demonstrates the fact that the table space required for storing XML data with the NRSD is significantly less than the size of the original XML documents. This is mainly due to the fact that the NRSM eliminates the redundant data from XML documents.

REFERENCES

[1] D. Florescu and D. Kossman. *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database.* In Proc. of VLDB'99, (1999).

[2] H. L. Lau and W. Ng. *Querying XML Data Based on Nested Relational Sequence Model.* In: Proceedings of Poster Track of WWW, (2002).

[3] H. L. Lau and W. Ng. *The Development of Nested Relational Sequence Model to Support XML Databases.* In: Proc. of the 2002 Int. Conf. on Information and Knowledge Engineering (IKE'02), pp. 374-380, (2002).

[4] M. Levene. *The Nested UniversityRelation Database Model.* LNCS Vol. 595, Springer-Verlag, (1992).

[5] M. Ley. *Digital Bibliography & Library Project.* In: http://dblp.uni-trier.de/, (2002).

[6] W. Ng. *Maintaining Consistency of Integrated XML Trees.* Int. Conf. on WAIM'2002. LNCS Vol. 2419, pp. 145 -157, (2002).