World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:12, 2008

XML Data Management in Compressed Relational Database

Hongzhi Wang, Jianzhong Li, and Hong Gao

***Abstract*—**XML is an important standard of data exchange and representation. As a mature database system, using relational database to support XML data may bring some advantages. But storing XML in relational database has obvious redundancy that wastes disk space, bandwidth and disk I/O when querying XML data. For the efficiency of storage and query XML, it is necessary to use compressed XML data in relational database. In this paper, a compressed relational database technology supporting XML data is presented. Original relational storage structure is adaptive to XPath query process. The compression method keeps this feature. Besides traditional relational database techniques, additional query process technologies on compressed relations and for special structure for XML are presented. In this paper, technologies for XQuery process in compressed relational database are presented..

***Keywords*—**XML, compression, query processing

I. INTRODUCTION

BECAUSE its extensibility and ability of representation of semistructured data, XML[1] database can represents more complex semantics than traditional database. But the application of XML database needs the support of efficiency storage. The storage strategies of XML includes three types, flat text, OO DBMS and relational DBMS.[2] As a type of mature DBMS, relational database has its special advantages such as efficiency query process, concurrency control and so on. Commercial relational database products are widely used by many users. Using them to support the storage and query of XML data will result in many advantages. There are several research about storing XML in [3-9, 15, 16].

A notable problem of existing storage structure of XML is the redundancy of XML. An XML document can be divided into two parts, one is the context of data it represents, the other is structure of XML which is compounded by tags. Structure information is important for the semantics of XML document. However, lots of repetitive tags result in redundancy. And the context that XML document represents also has redundancy. Storing XML data in relational may result in additional redundancy because of the decomposition of the schema. In order to support XML database efficiently, it is necessary to compress XML document in relational database.

Hongzhi Wang is with the Harbin Institute of Technology, Harbin, 150001, China (phone: +86-451-86415280-22; fax: +86-451-86415827; e-mail: wangzh@hit.edu.cn).

Jianzhong Li is with the Harbin Institute of Technology, Harbin, 150001, China (e-mail: wangzh@hit.edu.cn).

Hong Gao is with the the Harbin Institute of Technology, Harbin, 150001, China (e-mail: honggao@hit.edu.cn).

Several XML compression technologies have been presented [10-14]. Some of them are only design for the compression the XML document without the requirement of query [10, 11]. The compression methods considering query in [12, 13, 14] are all the methods on original XML document. The query efficiency of them depends on the query process technologies on XML data. In [14], a compression method of the structure of XML is presented without considering of context.

In this paper, the compression technique special for XML document in relational database is presented. At first, a relational structure supporting XML document in various sizes and schemas is presented. This storage structure supports path query very well and adapt for query process on it. In order to make the relational adaptive to be compressed, vertical partition of the relation is used as the structure. And various types of context data are compressed respectively to gain large compress ratio.

The query process technology on compression relations is presented. Without considering compression instance, the query translation technology of XQuery[28] to query plan with relational algebra for common relation database is presented. Since the query optimization and query execution technologies of relational database are quite mature, they needs only a little improvement to support the compression.

The contributions of this paper include:

- The paper presents a relational storage structure of XML data adaptive for both compressing and querying XML data. The path is kept in the relation. Primary part of XPath[29] query can be processed in main memory. Vertical partition based physical storage structure is also designed to support compression.
- Compression technology for XML data in relational database is presented. A practical compression method is designed for the structure information of XML data in relational database.
- Context is separated and compressed based on data type and range of value which are obtained without schema information.
- Query translation technology from XQuery to relational is designed for XML query process on relational database. This technology is independent to the compression structure can be applied for querying XML in common relational database.
- Considering the usage of existing query optimization and execution technologies of relational, a little improvement is done for the special feature of our compress structure for XML data.

The remainder of this paper is organized as follows. In

section 2, we summary related work. In section 3, special relational storage for XML is presented, as well as compression technology on this sturcure. Query processing technology on compressed data is designed in section 4. Section 5 contains the result of our experiments. Finally, in Section 6, we summarize our work.

```
<!ELEMENT bookshop (name, department*)>
<!ELEMENT department (name, book*)>
<!ELEMENT book (title, author*,publisher, price>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT number (#PCDATA)>
```

Fig. 1 DTD of Example

Motive Example: In order to explain our storage and query process strategy, an example of a bookshop is presented. The DTD of the example is shown in fig.1. In the DTD, price has float type, number has integer type, name, title, author and publisher are in string type.

II. Related Work

Data compression is an important field of information theory. Data compression has some advantages such as reduction of storage size, saving network bandwidth and accelerating query processing by reducing times of I/O. With these advantages, compression technology plays an important role in database.

For XML data, there are two types of compression method. One is to separate context and tag. They are coded respectively. The two parts are assembled after coded. This type of compression technologies keeps homomorphism between compressed XML document and original XML document. Therefore, most query process technologies directly to XML document are still not efficient to compressed XML document. XGRIND[12] and XPRESS[13] are of this type. These technologies have large compression ratio. But an obvious problem of them is that in order to support efficient XPath Query, some index is necessary, as will result in additional storage cost. When XML document is large, existing technologies of query processing are inefficient.

The other type of compression is to compress structure and context separately and store them respectively. XMILL[11] belongs to this type. XMILL stores data with same type as a unit and compress each unit using a special compression method. The compression ratio of XMILL is high. But query can not be processed on compressed data without decompressing the whole document. [15] presents a technology that compresses only structure of XML document. Compression of structure has excellent compression ratio. The problem of this technology is that the connection of structure information and context information results in additional storage. And without the compression of context, compression ratio of the whole document is limited.

There are many relational storage structures supporting XML documents. [3] and [4] considers large XML document and store XML document in the relations that are partitioned based on the schema of XML document. [5] and [15] are the extension of this method to self-tuning of the storage structure. These methods are not adaptive to the instance of storing many XML documents with various schemas and bring redundancy. Query process in these storage structure results in many join operations. It is hard to translation XQuery to SQL for various instance of schema. More adaptive storage structure and query translation is presented in [5] and [6]. The method is to store each edge of the XML document. The shortcoming is that the query process efficiency on this storage is low. [7, 16] use interval encoding to represent the ancestor and descendence. XRel [5] is to encode the path and store the code and the extent of each node. But the translation detail of XQuery is not presented in [5]. In [9], a storage structure is presented to converting XQuery to range query. It accelerates XML steps. But none of the above structures consider the problem of compression.

It is noted compression method that can be used to compress database should lossless one. There are much research work about compressed relational database[17-23]. [17] gives many methods to compress database. A block-based compression method is presented in [18]. In [19], an order preserving string compression method is presented, which is useful for the compression of database with text. [20] uses vector quantization to compress the repetitive in recorders. The query optimization strategy is presented in [12]. With the strategy, compressed database can be changed to a self-change one. [22] presents a loss compression method with the usage of data mining technology. The compression method special for 2-ary relation is introduced in [23]. Some technologies of these papers are referenced during the design of our compression method of relational database.

There are also many technologies for lossless compression of general data. Huffman[24] is a classical compression technology. It assigns shorter code to most frequently appearing symbols and longer code to less frequently appearing symbols. BWT[25] is a new technology of compression text. (introduction to BWT) Dictionary coding is effective when only a small number of words exist. The idea of dictionary coding is to assign an integer value to each new word from input data. The differential encoding method, also called delta encoding, replaces a data item with a code value that defines its relationship to a specific data item. Differential encoding is adaptive to compression of numerical value. All these methods can be selected to compress some part of XML data lossless.

III. Compression of Relational Database with XQuery Support

In this section, the storage of relational database with XQuery support is presented. Based on the logical structure, a special physical storage with compression technology is designed.

A. Store XML in Relational Database

Beyond existing relational structure storing XML document, a storage structure adapt for XQuery process on XML data. Some redundancy is permitted because compression technology will be applied to eliminate the redundancy without effect to the query process.

Each node of XML document can be uniform identified by its preorder rank. Each path from root to leaf node can be represented as the list of the preorder ranks of nodes in this path. Based on this fact, the storage structure of XML document is to store the preorder ranks of a path with the value. The paths with same path schema s are stored together as a relation, which is called path relation of s, r_s for short. If l_s is the length of path schema s, r_s has l_s+1 columns, the first ls columns are the preorder rank of the path with names the name of corresponding tag and the last column with the name value is the value of the leaf node. It is suppose that for a record a in r_s, the column i of it is a(i).

The tags in XML document is encoded and stored in a separate relation, r_{tag}. Each path schema can be presented as the list of tags. All Path schemas of leaf nodes in XML document are also encoded. The content of path schema with its code is stored in a relation r_p, in which the content of path schema is represent as a string with each char having ASCII value of the tag nodes' codes. The schemas of r_s, r_{tag} and r_p are shown in table1, table2 and table 3.

Two properties of XML document stored in this structure are presented as Theorem1 and Theorem2. Before the two theorems are presented. A property of preordered rank stored in r_s is shown in Lemma1.

Lemma1 Two records a and b in an r_s. when i<ls+1, if a(i)=b(i), a(i-1)=b(i-1).

Proof a(i)=b(i) represents the ith nodes of path corresponding to record a and b are the same. It is supposed to be n_i. In a XML tree, the parent of a node is uniform. The preorder rank of n_i's parent is preorder(n_i-1)=a(i-1)=b(i-1). □

All the records in r_s can be sorted by the first l_s columns with the order from 1 to l_s. Thus, a property of sorted rs is shown in Therom1

Theorem1 all records r_s which has been sorted by the first l_s columns with the order from 1 to l_s with the same values of the ith (0<i<l_s+1) column must be coterminous.

Proof The theorem is to be proved in mathematic induction.

When i=1, the 1st column of r_s represnts the preorder rank of root of XML tree, which is uniform in a XML tree.

If when i=n, the theorem holds, it is to be proved that when i=n+1, the theorem holds.

By Lemma1, the values of the parents of the ith column with the same value of all the records must be same. By the induction hypothesis and the sorted rules that the records with the same values of the ith column are sorted by the i+1 column, records in sorted r_s with the same values of the ith (0<i<l_s+1) column must be coterminous. □

A property of the order of nodes in the same level is presented in Theorem2.

Theorem2: In the same column of a sorted relation, the order of the node to the same parent node is kept.

Proof: Since the model of XML is an ordered labeled the tree, the prerank of a node is smaller than that of its following sibling. Hence in an ordered column, the prerank for the node precedes that of its following sibling. □

B. Compression of the Relational Database

In this section, the compression method of the relational database is presented. Based on special physical storage for compression, the columns of the path and that of the value are compressed respectively.

1) The Physical Storage adaptive for compression

The data with the same data type and range of value is easy to compress. An observation of r_s for various s following the storage schema in 3.1 shows that the redundancy exists in that the some leaves may share the same ancestor, the code of which is stored many times. To solve this problem, the physical storage is the vertical partition of the relation so that the preorder rank of the same type of data can be gathered together. And the values of the leaf nodes with the same path schema often have the same data type and range of value. Storing the values together makes the context information easy to compress. Another advantage of vertical partition is that the acceleration of projection operation, which is often used during the process of XQuery in relations to select special nodes binding to variable in variable forest.

2) The compression of Path property

With the assurance of Therom1, on the physical storage of vertical partition, the value of each path property can be compressed the following compression method. For the storage of the ith column, the difference of each value and the former value to it is obtained. And the value with the number of the number of times the value continues to exist is stored. Although the method is simple, the wide the column can be shortened and the repetitive values can be eliminated.

Example 3.1: The array 1, 2, 3, 4, 5, 7, 8, 9, 11, 13, 15 is converted to 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2 and compressed as 1, 5, 2, 1, 1, 2, 2, 3. □

3) The compression of the value column

The value column of each rs can also be compressed. The

TABLE I THE SCHEMA OF R_S

name	type	meaning
p_1	long	Preorder rank of 1st node of the path
p_2	long	Preorder rank of 2nd node of the path
...	....	...
P_{ls}	long	Preorder rank of l_sth node of the path
value	Integer/string/double	The value of path

TABLE II THE SCHEMA OF R_{TA}

name	type	meaning
name	string	name of the tag
code	int	code of the tag

TABLE III THE SCHEMA OF R_{PATH}

name	type	meaning
path	string	The context of the path
code	int	code of the path

TABLE IV COMPRESSORS TO VARIOUS PATH TYPES

Compressor	Path type
u8	Integer max-min<2^8
u16	Integer max-min<2^{16}
u32	Integer max-min<2^{32}
f32	Float
Dictionary encoder	Enumerate typed data
Suffix encoder	String
BWT encoder	Text

value column is compressed by its data type and value of range. The type of the column of value is judged by statistics. The data type supported by our compressor includes enumeration type, integer, float, string and text. Obviously, enumeration typed data is easily to be compressed with dictionary compressor. Hence at first, each path type is supposed as enumeration type. The values of the path class are stored. If the number of the values is larger than a threshold, the path type is considered not enumeration typed and judged by existing value. If existing value is all number but the first character is not '0', path type is considered as integer. If all characters is '0' to '9' or '.', path type is considered as float. If the length of existing values is smaller than a threshold, it is string. Otherwise, path type is text. The range of value should be recorded with path type integer and float.

Compression method to path class can be determined with path type. Referencing compressors to various data types in [4] and [5], the compressor to various data type is presented in table1. u8, u16 and u32 are the differential encoders for integer. F32 is float encoder built according to IEEE float code standard. Enumerate typed data is compressed by dictionary compressor. String is encoded with suffix compressor. While for long text, BWT[10] compression method is used.

IV. Query Processing on Compressed Database

In this section, query process technology on the database compressed with the method of is presented.

A. An Overview of Query Processing

Basic idea of query process is to express query plan with relational operators. XQuery is expressed as relational The steps of query process are shown in fig.2

Before query process, XQuery expression should be parsed and preprocessed. Besides computing expression with only constant and reduce constant restriction,

The first step of query translator is to extract the variables and their relationship from the query. They are represented as a tree, as is defined as variable tree. Each node represents a variable with its path in the query. More than one tree can be extracted from the query. The results of the trees are to be join in semantics. All of variable trees of a query form a forest, as is defined as variable forest.

And then the condition of the query is transformed into the form of DNF. Each clause of DNF is CNF. Each clause is to be generated on forest. For the optimization, some clause should be merged.

The next step is to add each CNF clause of restriction clause to the forest. Thus, several restricted VFs generate. The final result is the unions of these restricted VFs. Additional nodes are added to the forest. Restrictions are added to existing nodes. The generation and adding restriction process of VF is presented in section4.2.

Query plan is generated from restricted VFs. The query plan generation method is presented in section 4.3. And the query plan is executed to obtain the information from compressed relational database. The query plan is represented by relational

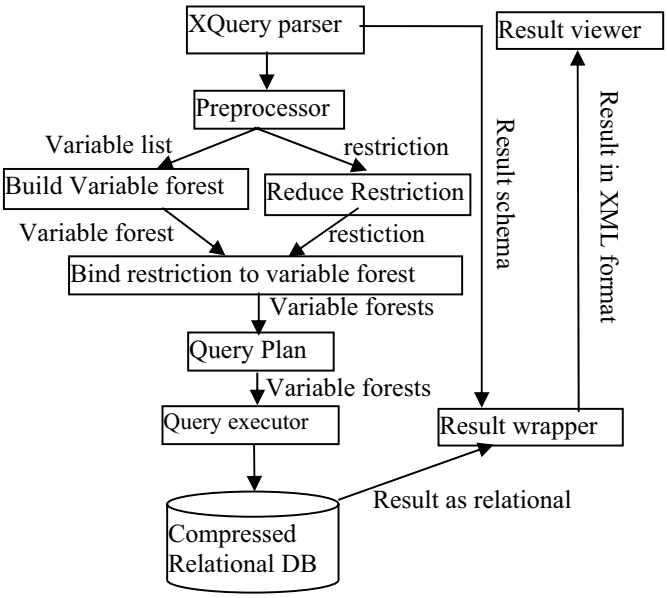

Fig. 2 Query Process steps

operators. The implementations of these operators on compressed storage are quite like those on common relational database. The improvements of the implementation of operators are introduced in section4.4. The last step is to wrap the final result to the format of XML with the result schema.

B. Generation of Variable Forest

1) Generation of Original Variable Forest

The first step is to convert the query to variable forest containing several variable trees, which represents the relationship between the variables in the query.

Definition 4.1 Variable Tree (VT for short): A Variable Tree of a query Q T_Q=(r, V, E), where r is the root of the tree, V is the set of nodes of the tree and E is the edge of the tree such that:

Each v V can be represented as (ID, var, path, res, isresult, agg), where ID is the identify of the v which is uniform in the whole variable forest, var is the name of varable v binding to, path is the XPath expression of the var in Q and res is the restriction of this var. If v is a leaf node, res is the comparison of var and a constant. Otherwise res is the existing restriction of some child of v in T_Q or the relationship between the children of v in T_Q. property *isresult* represents if the variable exists in the final result. Property agg represents the aggregation function with id of group by nodes on this node.

Each e E is labeled pc(for parent child relationship), ad(for ancestor descendant relationship), ad* (for self-or descendant

relationship)or the description of XPath. □

Definition 4.2 Variable Forest (VF for short): A set of variable trees F_Q= (S_T, R), where S_T is the set of variable trees in F_Q and R is the set of relationships between the trees in S_T in F_Q.R is represented as the restriction to connect the VTs. □

Example 4.1 For the query to "for each book published by MIT Press find the number of book published by Prince Hall with the same title as it

```
for $a in document("bookshop.xml")//book                    (Q1)
    let $c:=for $b in document("bookshop.xml")//book
       where $a/title=$b/title and $b/publisher='Prince Hall'
```

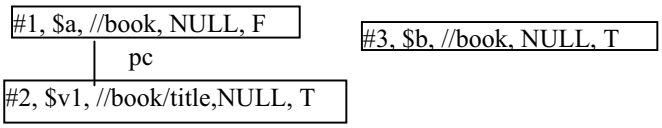

Fig. 3 Initial VF of Q1

```
      Result <item>$b</item>
    where $a/Publisher="MIT Press"
    result <result>
  <book1>$a/title</book1>
       <number>count($c)</number>
        </result>
```

The VF of Q1 has two VTs. The initial VF of Q1 is shown in fig.3, in which $v1 represents a virtual variable. □

The generation of a VF is to extract the variables from query and construct VTs. At first, for each independent variable that is obtained from for and let clause, a VT is built with ID, var and path expression. It is noted that the path expression is absolute path from root to the node var standing for.

2) Binding Restrictions to VF

Each DNF r_i=r_{i1}∧r_{i2}∧……∧r_{imi} of restriction is then added to the VF. The restricted VF generated by r_i is VF_i. Internal plan p_i is translated from each VF_i respectively. Final result is the union of the result of all the internal plans.

The process of adding r_i to VF is to apply each r_{ij} to VF. The application is processed according to the variables in each r_{ij}.

- If r_{ij} contains just one variable or XPath expression that has been bound to some node n in VF, the restriction rij is add to res property of n.
- If r_{ij} contains variables or XPath expressions that have not been bound to any node n in VF, a node n binding to a virtual variable is added to corresponding VT in VF. In this instance, if r_{ij} has only one virtual variable, the restriction is added to res property of n directly.
- If r_{ij} contains more than one variable or XPath expression, all variables or XPath expressions are bound to the VF at first. And then, each variable v is check to determine which VT it belongs to. If all the variables belong to the same VT T_k, restriction r_{ij} is added to the nearest ancestor of them in T_k. Otherwise, a connection node is built to connect all the VTs that r_{ij} has variable binding to.

Example 4.2: When the restriction in where clause of Q1 is applied to VF in Example 4.1, the VF is converted to the form as fig.4□

C. Construct Query Plan from Restricted Variable Forest

The query plan of compressed relational database is in the same form as that of common relational database[26].

1) Generation of Logical Plan

The generation of logical plan is to traverse bound VF and generation corresponding logical plan for each node. The translation result may have more than one operator, such that the relationships among them are defined. The translated result of one node n in VF can be considered as a box with its inputs the result of the children of n and the output of n is used as the input of n's parent.

The traversal is the hybrid of preorder and postorder. Each internal node in VF is to be accessed two times during the traversal. During the traversal, a stack is used to record the variable to be projected in the path. When first time to access an internal node n, if var of n has more than one child or agg property of n is not NULL or it appears in res property of its ancestors, the tag number corresponding to n is pushed in to project stack. At the same time, the set of various ids that all its ancestors need are recorded in n.

When a leaf node is met, the names of the relations of leaves of each VT in VF are obtained by matching the path property of the leaf node to the path context column in r_p. These relations are represented as r_{t1}, r_{t2},…, r_{tn}. A projection operation is added to each r_{it} with projection columns the union of current project stack t, the tag number of the leaf and value. A selection operator is then added above projection operators if there is some restriction binding to the node. A projection precedes selection is because that the cost of projection operator is constant. If more than one relation matches the path expression of this node, an addition union operation is added to the top of the query plan for each single relation.

When the second time visiting an internal node n, a join operation is added with the objects the results of the operations corresponding to its children. The join restrictions include not only the res property but also the restriction of the equation of the column corresponding to the tag number of n. The selection of the order of the join is left for the generation of physical plan.

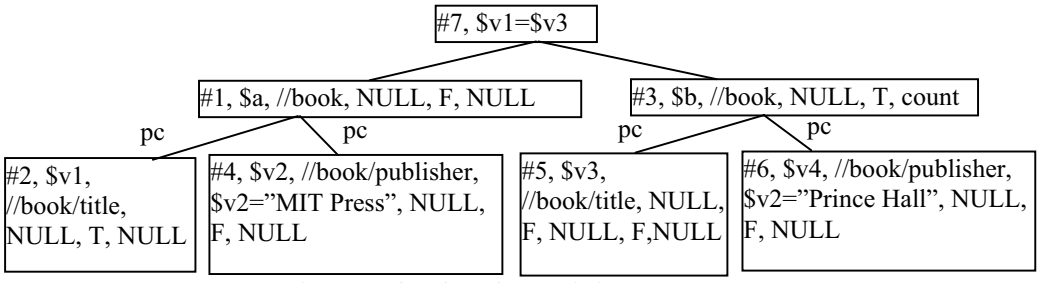

Fig. 4 VF of Q1 bound to restrictions

A projection operation is added to the top of

A join operator may be generated from the top level restriction node with the restriction stated in this node. The projection of final result and aggregation operation is added to the top level.

Example 4.3: The logical query plan of the VF in example4.2 is shown in fig.5

2) Generation of Physical Plan

The generation of physical plan is the improvement of that of common relational database with the data properties in the storage structure. And the cost estimated also needs a little improvement. Here only the improvements are presented.

First of all, based on Theorom1, arbitrary column except value is ordered. Thus, the join with restriction of only columns of preorder ranks is naturally merge sort join with the cost only that of merge operation without that of sort. In order to keep the order during the process, if there is union above the logic plan according to a node in VF, the physical operator of the union is multi-way merge union.

Since our operator is on compressed data, the additional operators of compression and decompression are necessary. In the scan for selection operator or join operator by the restriction of value column, decompression is necessary only when the string matching operation is applied to the column with type text. Otherwise, the value in restriction is compressed to compare with the compressed value in value column. The cost of compression and decompression operator is also depends on the compress method and the size of compression data. The cost of scan operation should consider the reduction of I/O. The reduction depends on the estimate of compression ratio of the compression method and the statistics of the data.

D. Execution of Query Plan

The implementation of query operators in a query plan is also quite similar to that of common relational database except compression operators should be added to the implementation.

The sequence scan operation on compressed data should be considered. When the operation scan is executing, on the compressed column of preorder rank in path, the preorder rank of the node can be extracted from compressed data in a cup time of adding. And when the operation scan is by the restriction of value, the value in restriction is compressed in the compression method of corresponding value column. It is noted that performing scan on the vertical partitioned table needs to keep the synchronization of the block of all necessary columns.

V. Experiments

A. Experimental Environment

The experiments are performed on a PC with AMD 1G CPU and 256M main memory. Operation system is Microsoft windows XP Professional and develop platform is Visual C++ 6.0. Relations and temporal tables are stored as files in local computer.

Data Set: XMark[27] is a famous benchmark of XML data management. It covers many features of XQuery. We use XMark as our data set with various factor.

Queries: All the 20 queries presented in XMark are used for experiments.

B. Performance Result and Analysis

Three measurements are used to represent the performance of the compression:

RE ratio(relation expand ratio)=relations size/original size-1

RC ratio(relation compression ratio)=1-size of compressed relations/ size of relations

DC ratio(document compression ratio)=1-size of compressed relations/size of original XML document

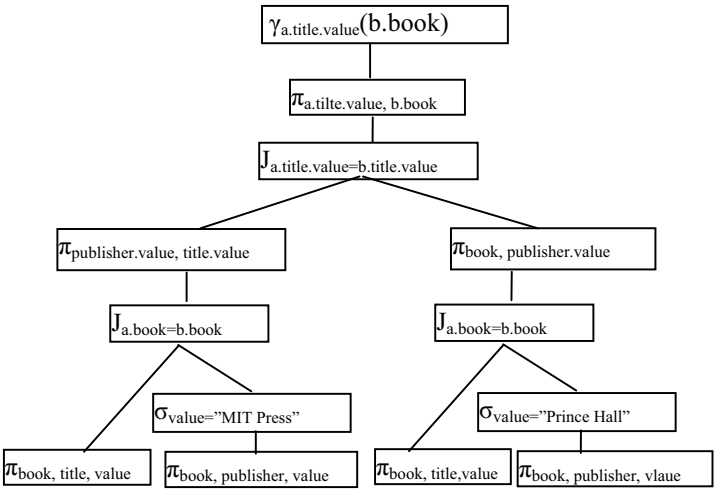

Fig. 5 logic plan of Q1

The result of compression is in table5.

The compression ratio of XMark data sets with factor above to 0.1 is a similar. The reason is that they are automatically generated and have similar ratios of various data types and tags.

To test the efficiency of query process on compressed data, the 20 queries in XMark is applied to XMark data set with factor 1.0. The result of query process is shown in table 6.

From the result, the efficiency of the query processing on compressed data is acceptance. Q8-Q11 are slower because the join of XML trees results many joins with many temporal tables. Especially 14 temporal tables are resulted during processing of Q10. Q14 is slower because that we do not optimize for the string matching.

If the compression and query processing technology is embedded to an existing database system with optimization of execution of joins and temporal tables, higher efficiency can be gained.

VI. Conclusions

In this paper, the strategy of storing and querying XML document in compressed relational compressed database is presented. On one hand, the strategy has the relation database's advantage of easy query process. On the other hand, it reduces the storage needed to store XML data, as well as the disk I/O during query processing. A special relational structure for storing XML is presented. And the compression on it obtains a

TABLE V RESULT OF COMPRESSION

DS	Factor	Ori. size	Rel. size	RE ratio	Com. size	RC ratio	DC ratio
XMark	0.0001	33.13K	36.532K	10.2%	29.98K	17.9%	9.5%
XMark	0.1	11.13M	11.71M	5.3%	5.26M	55.0%	52.70%
XMark	0.2	22.43M	23.63M	5.4%	10.44M	55.84%	53.46%
XMark	0.4	46.89M	44.53M	5.3%	20.54M	56.19%	53.87%
XMark	0.8	89.23M	93.95M	5.3%	40.88M	56.48%	54.18%
XMark	1.0	111.1M	116.9M	5.3%	50.8M	56.51%	54.22%
Shakes	/	7.50M	7.97M	6.2%	2.98M	62.60%	60.3%

TABLE VI THE PERFORMANCE OF QUERY PROCESS ON

Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Time(s)	3.012	28.352	2.565	97.002	3.546	0.03	0.52	79.691	358.21	1102.35
Query	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Time(s)	411.12	409.55	1.713	475.35	8.152	1.058	0.581	1.465	3.390	0.791

good compress ratio. The technique of converting XQuery to logical query plan of relational is designed as the basic of query on the XML data in compressed relations. During query process, most techniques of query process in relational database continue to be used with a little improvement, which is also introduced in this paper.

This compression method adapts to very large XML databases with the application of XML warehouse and to be embedded to existing relational databases for better supporting of XML data.

REFERENCES

[1] T.Bray, J.Paoli, C.M.Sperberg-McQueen. Extensible markup language(XML)1.0. W3C Recommendation. Feb.1998.http://www.w3.org/TR/REX-xml.
[2] Feng Tian, David J. DeWitt, Jianjun Chen, Chun Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. SIGMOD Record special issue on "Data Management Issues in E-commerce", March 2002
[3] A Deutsch, M. F. Fernandez, D. Suciu, Storing Semi-structured Data with STORED, SIGMOD Conference 1999.
[4] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, J. F. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB 1999.
[5] S. Abiteboul, P. Buneman, D. Suciu. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers. 2000.
[6] Ioana Manolescu, Daniela Florescu, Donald Kossmann. Answering XML queries on heterogeneous data sources. In Proc. of VLDB 2001.
[7] M. Yashikawa et al. : XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases. TOIS2001
[8] P. Bohhanon et al. From XML Schema to Relations: A Cost-Based Approach to XML Storage. In the Proc. of ICDE 2002
[9] Torsten Grust. Accelerating XPath Location Steps. In Proc of SIGMOD 2002.
[10] James Cheney, Compressing XML with Multiplexed Hierarchical Models, in Proceedings of the 2001 IEEE Data Compression Conference, pp. 163-172
[11] Hartmut Liefke, Dan Suciu. XMill: an Efficient Compressor for XML Data. In Proc of ACM SIGMOD2000.
[12] Pankaj M. Tolani, Jayant R. Haritsa. XGRIND: A Query-friendly XML Compressor. In Proc. of the 18th International Conference on Data Engineering, 2002
[13] JunKi Min, Myung-Jae Park, ChinWan Chung. XPRESS: A Queriable Compression for XML Data. In Proc. of ACM SIGMOD 2003.
[14] Peter Buneman, Matin Grohe, Christoph Koch. Path Queries on Compressed XML. In Proc of the 29th VLDB conference, 2003.
[15] Zhengchuan Xu, Zhimao Guo, Shuigeng Zhou, Aoying Zhou. Dynamic Tuning of XML Sgorage Schema in VXMLR. In Proc. of IDEAS 2003.
[16] David DeHaan, David Toman, Mariano P. Consens, M. Tamer Ozsu. A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding. In Proc. of SIGMOD 2003.
[17] T.Westmann, D.Kossmann, S.Helmer, G.Moerkotte, The Implementation and Performance of Compressed Databases, SIGMOD RECORD, Vol.29, No.3, Sept. 2000
[18] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. Proceedings of the IEEE Conference on Data Engineering, pages 370-379, 1998
[19] G. Antoshenkov, D. Lomet, J. Murray. Order Preserving String Compression. 12th International Conference on Data Engineering.
[20] W.K. Ng, C.V. Ravishankar. Relational database compression using augmented vector quantization. 11TH International Conference on Data Engineering
[21] Zhiyuan Chen, Johannes Gehrke, Flip Korn. Query Optimization in Compressed Database Systems.SIGMOD2001.
[22] Shivnath Babu, Minos Garofalakis, Rajeev Rastogi. SPARTAN: A Model-Based Semantic Compression System for Massive Data Tables. SIGMOD2001.
[23] S. J. O'Connell, N. Winterbottom. Performing Joins without Decompression in a compressed Database System. SIGMOD Record 32(2), June, 2003.
[24] D. A. Huffman. A Method for the Construction of Minimum Redandancy Codes. In Proceedings of the Institute of Radio Engineers 40, pages 1098–1101, September 1952.
[25] M. Nelson, "Data compression with the Burrows--Wheeler transform," Dr. Dobbs' J., Sept. 1996.
[26] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. Database System Implementation. Prentice Hall. 2000.
[27] Albrecht Schmidt, Florian Waas, Martin Kersten, Micheal J. Carey, Ionana Manolescu, Ralph Busse. XMark: A Benchmark for XML Data Managemetn. In Proc of the 28th VLDB conference, 2002.
[28] World Wide Web consortium. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/
[29] World Wide Web Consortium: XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20/

Hongzhi Wang Hongzhi Wang received his PHD in computer science from Harbin Institute of Technology in 2008. He is a lecturer of Department of Computer Science and Technology, Harbin Institute of Technology. His research area is XML data management and information integration.
Jianzhong Li is a professor of Department of Computer Science and Technology, Harbin Institute of Technology. His research area includes parallel database, sensor network, data mining, data warehouse, compressed database and XML data management.
Hong Gao Jizhou Luo received her PHD in computer science from Harbin Institute of Technology in 2004. He is a professor of Department of Computer Science and Technology, Harbin Institute of Technology. His research area is graph database and data warehouse.