An Effective XML Storage Method Based on Deleting XPath Query Minimization

Zhen-Fang Li

Abstract—**The storage method for XML data in database will affect the efficiency of XML keywords querying, indexing and updating significantly. In most cases, the XPath sentences provided by user are not refined. These queries can often achieve minimization by deleting redundant parts. It means the minimized query is one of subsets of this query. Such kind of minimization query is method based on deleting queries. The steps are determining nodes relations, finding redundant nodes, deleting redundant nodes and sub-tree and obtain final minimization query tree. In this paper, we propose an improved XPath query minimization method based on deletion using simulation concept. Experiments analysis results show that with introduction of appropriate indexing and new algorithm, the XML keywords indexing efficiency can be significantly improved.**

Index Terms—**XML keyword query, XPath query, XML storage, XML database.**

I. Introduction

XML query optimization involves many aspects as algorithm execution, physical plan selection and etc. Execution efficiency of XML query has close relation with an important factor except for above factors, namely complexity of query itself and query size.

In the XML query and keyword indexing, the database to store XML data can be divided into two types, namely Native XML database [1], [2] and non-Native XML database [3], [4]. The feature of former is that management on XML data based on XML data model [5], while latter map XML data model into other data model, such as relation model [6]. As to semi-structured characteristics, mapping from XML data to other models always make problem complicated and bring problems of update as well as query efficiency. XML document is an ordered tree. Storage in the unit of node or sub-tree not only needs to record data in XML node, but also should consider structural relationship among nodes. Therefore, if nodes with structural relationship are not stored in same record, it always needs to save certain number of structural information to ensure correctness of data reduction. Generally, if parent node or child node of one is not kept in same record, their record address should be stores, so that records are not mutually independent. Once recorded address was changed, it will affect record that keeps this record address. However, XML data update always introduces change of node record address. The dependent relation among records will impact on XML update efficiency.

Manuscript received November 9, 2012; revised January 12, 2013.

Zhen-Fang Li is with the School of Computer and Information Technology, Shanxi University, China (e-mail: lizfsmile@163.com).

In this paper, we propose an improved XPath query minimization method based on deletion using simulation concept. Experiments analysis results show that with introduction of appropriate indexing and new algorithm, the XML keywords indexing efficiency can be significantly improved. The specific arrangement of paper is as follows: section 2 introduces related works; section 3 gives minimization method based on deleting; section 4 performs experimental analysis and section 5 concludes our work.

II. Related Works

XPath is the core and basis for many languages as XQuery and XSLT. Computation on XPath expression is actually seeking for mapping from Tree Pattern Query (TPQ) to document tree, or called match for tree pattern query. Generally, efficiency of tree pattern query matching is directly related to scale of query. Larger is tree pattern query scale, lower is matching efficiency, so we should simplify tree pattern query as possible before tree pattern query matching, namely to minimize query expression.

In order to formally describe XPath query minimization problem, some definitions are provided here firstly. We always assume existence of node identifier character set N as well as node tag or type Σ. If there is no special statement, $a,b,c,d,e,f \in \Sigma$ and are not equal; $r,s,u,v,w,u',v_i \in N$; p is the query and Σ_p is node tag set in p. Meanwhile, under the premise of not leading to confusion, call node whose tag is u in p as node u; the node whose identifier is a also as node a.

Definition 1: An XML document tree t is a four-tuple $(r_t, N_t, E_t, \lambda_t)$. Where, $N_t \subseteq N$ is set of all nodes; $r_t \in N_t$ is the only root element node; $E_t \subseteq N_t \times N_t$ is set of all edges in the tree. The $\lambda_t : N_t \to \Sigma$ is a function to determine type of each node. A typical XML document tree is shown in Fig. 1.

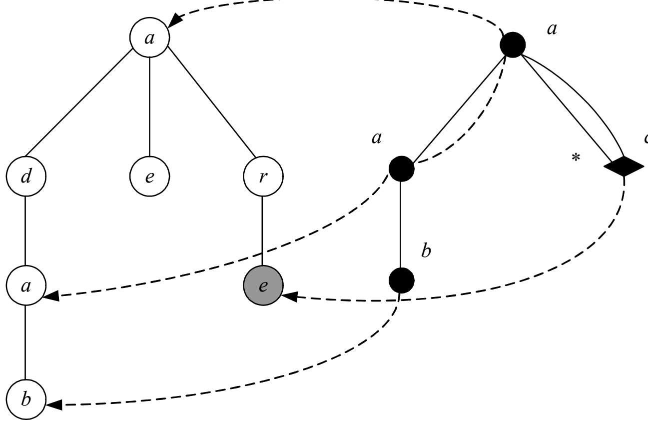
Definition 2: A Tree Pattern Query (TPQ) p is $<t_p, o_p>$. Where, $t_p =< r_p, N_p, E_p, \lambda_p >$ is a tree. The E_p is divided into three disjoint sets C_p, D_p and DS_p to represent all child edges c-edge, decent edges d-edge and descendants or themselves ds-edge. The o_p is the only output node.

An example of TPQ is shown in Fig. 1 (b). Where, the single line represents c-edge, double line as d-edge, dashed and solid line as ds-edge. The diamond-shaped node is output node.

Definition 3: Given $u,v \in N_p$, if edge $(u,v) \in E_p$, then we can call v is child of u. Particularly, if $(u,v) \in C_p$, v is c-child

DOI: 10.7763/IJFCC.2013.V2.188

of u. If $(u,v)\in D_p$, v is d-child of u. If $(u,v)\in DS_p$, v is ds-child of u. Assume u reach v along edges $e_1,e_2,\cdots,e_n(n\geq 0)$, where $e_i\in E_p$, v is quasi-descendant of u. Particularly, u is the quasi-descendant of itself. If $n\geq 1$ and there is $e_i(1\leq i\leq n)$ not belong to ds-edge, v is called descendant of u. If v is quasi-descendant of u, u is quasi-ancestor of v. Particularly, v is quasi-ancestor of itself. If v is descendant of u, u is ancestor of v. In addition, *child*(u) is used to represent set of all child nodes in u. The corresponding set of all c-child nodes of u is represented by *c-child*(u).



(a)XML treat (b) the pattern query
Fig. 1. XML tree, tree pattern and its insertion.

The solving of TPQ is realized by mapping from tree pattern query to document tree, which is called embedding.

Definition 4: Given tree pattern query p and document tree t, the embedding of p into t is a function $e{:}N_p\rightarrow N_t$ that meet following:

1) Maintain node identifier: $\forall x\in N_p$, if $\lambda_p(x)=a(a\neq *)$, then $\lambda_t(e(x))=a$.
2) Maintain edge relationship: $\forall(x,y)\in C_p$, $e(y)$ in t is child of $e(x)$; $\forall(x,y)\in D_p$, $e(y)$ in t is descendant of $e(x)$; $\forall(x,y)\in DS_p$, $e(y)$ in t is descendant or itself of $e(x)$.

Given tree pattern query p and tree t, $p(t)$ is used to represent solution of p on t. It means $p(t)=\{x\in N_t \mid \exists e,e(o_p)=x\}$.

An example of TPQ embedding is shown in Fig. 1. The left is an XML document tree and right part is a TPQ. The dotted line part shows mapping from TPQ to document tree node. We can see from the figure that query result is node c.

Definition 5: Given two TPQs p_1 and p_2, p_2 contains p_1, if and only if $p_1(t)\subseteq p_2(t)$ for any document tree t, denoted as $p_1\subseteq p_2$.

Definition 6: If and only if $p_1\subseteq p_2$ as well as $p_2\subseteq p_1$, two TPQs p_1 and p_2 are equivalent, denoted as $p_1\equiv p_2$.

Definition 7: Node $p\in Q$, if the query Q' obtained by deleting p and all its descendants from Q is equivalent to Q, the node p is a redundant node.

Definition 8: As to a query Q, if there is no other query equivalent to it and smaller than Q, then Q is called minimized query. In a TPQ, the node number of query is used to show query size.

III. XPath Query Minimization Based on Deleting

In most case, the XPath sentences given by users are not refined. These queries are usually can be minimized by deleting redundant part, namely the minimized query of it is one of its subsets. The steps are as following: determining node relationship, finding redundant nodes, deleting redundant nodes and sub-tree, ultimately obtain final minimized query tree. The paper proposes an improved XPath query minimization algorithm based on deleting. Firstly the concept of simulation is provided here.

Definition 9: As to a TPQ $p=<t_p,o_p>$, where $t_p=(r_p,N_p,E_p,\lambda_p)$, the simulation relation is defined as following: simulation is the maximum binary relation among all nodes in p. As to two nodes u and v, $u,v\in N_p$, v simulates u if and only following conditions are satisfied:

1) Maintain node identifier: it needs $\lambda_p(u)=\lambda_p(v)$. If $u=o_p$, then $v=o_p$.
2) Maintain node relationship:

If $pc(u,\ u')$, then $v\in cpar(sim(u'))$.

If $ad(u,\ u')$, then $v\in anc(sim(u'))$.

If $sad(u,\ u')$, then $v\in q-anc(sim(u'))$.

Lemma 1: Known a tree pattern query p and non-redundant node u in p, there are:

1) If and only if there is another c-child node $w\in sim(v)$ of u in p, the c-child node v of u in p is redundant.
2) If and only if there another descendant node $w\in sim(v)$ of u in p, the d-child node v of u in p is redundant.
3) If and only if there another quasi-descendant $w\in sim(v)$ of u in p, the ds-child node v of u in p is redundant.

As the research is conducted in Berkeley DB XML query system, the XPath query minimizing optimization has removed redundant descendant-or-self. The paper gives an improved XPath query minimization algorithm as following.

Input: Tree pattern query $p=<t_p, o_p>$.

Output: Tree pattern query p after minimization.

```
Simulate(p);
Minimize(rp);
Simulate(p)
//Access to all nodes in Np in down-top order
{
Sort nodes in Np with down-top order;
for(Each node u in Np) {
if(u is output node)
sim(u)={u};
else if(u is leaf node)
sim(u)={v| v∈N_p and λ_p(v)=λ_p(u) };
else
sim(u)={v| v∈N_p and λ_p(v)=λ_p(u)
        and( ∀u'∈ c-child(u), v∈cpar(sim(u')))
        and( ∀u'∈ d-child(u), v∈anc(sim(u')))};
```

```
compute cpar(sim(u)) and anc(sim(u));
}
Minimize(u)
for (each child node v in u) {
if(v is c-child){
  if(there is another un-deleted c-child node w and
w∈sim(v))
        delete v and its sub-tree;
      else
        Minimize(v);}  //node v is not redundant node
      else if(v is d-child) {
        if(there is another un-deleted node w in u and
w∈sim(v) ∪ anc(sim(v)))
          delete v and its sub-tree;
      else
          Minimize(v);}
  }
```

Simulate() firstly sort all nodes in down-top order so that all child nodes of node *u* emerges before *u*, which can be obtained by preorder reverse *p* and consumed time is *O*(*n*). Calculate *sim*(*u*), *cpar*(*sim*(*u*)) and *anc*(*sim*(*u*)) for each node *u*. Each one can be represented as a *n*-triple Boolean matrix, which is indexed and located by node $u \in N_p$. In the computation of *sim*(*u*), the time for leaf node is *O*(*n*). As to immediate node, the consume time to compute *sim*(*u*) is related to child node number of *u*. To each child *v*, it needs to traverse *cpar*(*sim*(*v*)) or *anc*(*sim*(*v*)), while *O*(*cpar*(*sim*(*v*)))=*O*(*cpar*(*sim*(*v*)))= *O*(*n*), so the time to compute *sim*(*u*) is $O(n \times |child(u)|)$. Where, *child*(*u*) is child number of *u*. In the calculation of *cpar*(*sim*(*v*)) and *anc*(*sim*(*v*)), it only needs one more traverse from down to top, so the complexity is *O*(*n*). In short, the time of once for cycle is $O(n \times (|child(u)|+1))$, the consume time of Simulate() is $O(n \times \Sigma u \in N_p(|child(u)+1|)) = O(n^2)$.

Minimize() checks if there is redundant down-top. If not, check its child nodes. As to each child node *v* of *u*, it needs *O*(|*child*(*u*)|) to check whether there is redundant in *v*, namely determine whether there is child node belongs to *sim*(*v*) or *anc*(*sim*(*v*)), the consumed time is *O*(|*child*(*u*)|×|*child*(*u*)|), the consumed time of whole minimize(u) is $O(n \times \Sigma u \in N_p(|child(u)|^2)) = O(n^2)$. To sum up, the time complexity of this algorithm is $O(n^2)$.

IV. Experiment Analysis

The experiment was carried out on machine whose CPU is P4 2.9G. The development tool is VC6.0++. The experiment tool software is Native XML database management system Berkeley DB XML. The execution time is selected as evaluation index. The execution time given here is obtained from average time after many times of experiments and removing highest and lowest values. Data is generated by XMark. It is an opening source specialized for XML system test [7, 8]. Its XML generator Xmlgen can produce XML document with good structure. The data model is simulated Web site auction model. All data are in a large single document and size of data is determined by parameters.

According to above experiment schema design (See Table I), many experiments were conducted. Here the query is analyzed one by one. Q1 queries detail information whose name is “Mehrdad Reinhard” or “Dejan Alsio”. There are 7650 person in the document and 2 meet query conditions. Therefore, the index can be used to rapidly find name “Mehrdad Reinhard” and “Dejan Alsio” to form name list. In the mapping with nodes in person list, B+ tree structure indexing can be used to skip ancestor nodes not engaged in connections. Q2 queries first increase amount in the ongoing auction. If the first price increase exists, there is certainly price increasing, so we should firstly refine query sentences. There are 3600 open_auction in the document and each one contains different increase times. With B+ tree structural indexing, the descent nodes not engaged in connection can be rapidly skipped. Q3 queries name of all auction objects. The query sentence is a path query without branches. There are uncertain query paths that can be decomposed by above path decomposition method. Q4 queries how many objects have price more than 40. From comparison on original query system and improved one, the query time difference was recorded as shown in Fig. 2. We can see that with introduction of appropriate indexing and new algorithm, the system query efficiency can be significantly improved.

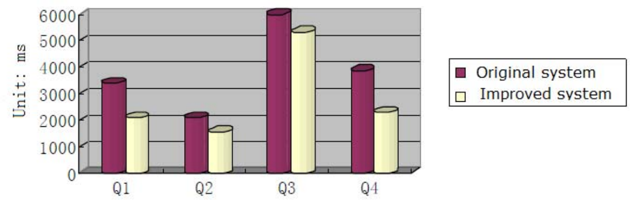


Fig. 2. Magnetization as a function of applied field.

TABLE I: Test Queries

Query	Content
Q1	query ‘for $b in (doc(“chenyinmei.dbxml/chenyinmei.xml”) /site/people/person [name=”Mehrdad Reinhard” or name=”Dejan Alsio”]) return <info>{$b}</info>’
Q2	query‘for $b in (doc(“chenyinmei.dbxml/chenyinmei.xml”) /site/open_auctions/open_auction) return <increase> {$b[bidder]/bidder[1]/increase/text()}</increase>’
Q3	query ‘for $b in (doc(“chenyinmei.dbxml/chenyinmei.xml”) /site/regions//item/name) return <name>{$b/text()}</name>’
Q4	query ‘count(for $i in (doc(“chenyinmei.dbxml/chenyinmei.xml”) /site/closed_auctions/closed_auction) where $i/price/text()>=40 return <price>{$i/price}</price>)’

V. Conclusion

Query optimization is important issue in the database optimization, which is also key to achieve efficient query. The paper addressed on keyword index optimization of XML database. Simulation technology was utilized to propose an improved XPath query minimization method based on deletion. It effectively shortens query time path expression and improves query efficiency. We also find in the research

that the enhanced simulation technology can be considered to provide integrity constrains as possible so as to better reduce query size and then improve query efficiency, which is also our research direction in the next.

REFERENCES

[1] P. Ramanan, "Efficient Algorithms for Minimizing Tree Pattern Queries," in *Proc. 21th ACM SIGMOD International Conference on Management of Data*, ACM Press, 2002, pp. 299-309.

[2] S. Flesca, F. Furfaro, and E. Masciari, "On the Minimizing XPath Queries," in *Proc. 29th VLDB International Conference on Very Large Database*, Morgan Kaufmann Publishers, 2003, pp. 153-164.

[3] S. Al-Khalifa, H. V. Jagadish, and J. M. Patel, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," in *Proc. 18th International Conference on Data Engineering*, *IEEE Computer Society*, 2002, pp.141-154.

[4] C. W. Chung, J. K. Min, and K. Shim, "APEX: an adaptive path index for XML data," in *Proc. 2002 ACM International Conference on Management of Data SIGMOD*, ACM Press, 2002, pp. 121-132.

[5] Q. Chen, A. Lim, and K. W. Ong, "D(k)-index: An Adaptive Structural Summary for Graph-structured Data," in *Proc. 2003 ACM SIGMOD International Conference on Management of Data*, ACM Press, 2003, pp.134-144.

[6] C. X. Wan and Y. S. Liu, "Efficient supporting XML query and keyword search in relational database systems," in *Proc. 3rd International Conference on Web-Age Information Management*, 2002, pp. 1-12.

[7] XMark – An XML Benchmark Project. [Online]. Available: http://monetdb.cwi.nl/xml.

[8] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse, "XMark：A Benchmark for XML Data Management," in *Proc. 28thVLDB*, 2002, pp. 974-985.

Zhen-Fang Li was born in Dec. 1979, Shanxi Province. She received bachelor degree in computer software and theory from Lanzhou University in Jun. 2001 and master degree in computer software from Lanzhou University in Jun. 2004. Since 2006, she is working for doctoral degree in computer application in Shanxi University. The main researches include XML database and XML keyword search.