Lock-based Concurrency Control for XML Document Models

Kuen-Fang Jea Shih-Ying Chen Sheng-Hsien Wang

Institute of Computer Science
National Chung-Hsing University
Taichung, Taiwan, R.O.C.

{kfjea, sychen, s9056014}@cs.nchu.edu.tw

Abstract

XML has become the most important technique to exchange data in WWW. Providing efficient access to XML document databases is thus crucial. Concurrency control protocols allow transactions to be executed concurrently to improve performance. In XML, XPath and DOM (Document Object Model) are two widely used models to access XML documents. XPath is a language to locate nodes in XML documents. DOM provides an interface allowing programs to dynamically access and update both the content and structure of XML documents. XPath reveals finer access behavior than DOM, but DOM provides a more convenient interface to access XML documents. Based on XPath and DOM, we propose and compare two lock-based concurrency control protocols for XML documents, called XLP and DLP, respectively. DLP is an extension of our former research on XLP [7]. Both of them have the features of richer lock modes, refined lock granularity, lower lock conflict and lock conversion. Our simulation results show that, for XML documents of different sizes, XLP outperforms DLP by 109.24% on average. Other performance comparison between XLP and DLP is also made in this study for various transactions with different destination node sizes and read/write ratios.

Keywords: concurrency control, locking, XML, XPath, DOM

1. Introduction

With the features of self-describing and ease of exchanging data, XML has been used in

This research is supported in part by NSC in Taiwan, R.O.C. under Grant No. NSC-90-2213-E005-014.

more and more applications, such as Science, Biology and Business. Data in these fields are usually very large. As a result, providing efficient access to XML documents is crucial.

Concurrency control is one of the most important techniques to improve the performance of database systems by allowing transactions to be executed concurrently. Lock-based protocols [1,4,5,6,8,9,10] are widely used in concurrency control since they are simple to implement and provide acceptable level of concurrency. Lock-based protocols use different types of lock to determine if a transaction can proceed. A transaction proceeds if the requested lock on the desired object is compatible with locks held by other transactions on the same object.

Two-phase locking (2PL) protocol [1,6,9,10] is the most widely used one and suitable for all databases. 2PL requires that each transaction only get locks in the growing phase and release locks in the shrinking phase to ensure serializability. The major defect of 2PL is that a transaction cannot release locks earlier so that the degree of concurrency is limited. In XML, many data items (e.g. tags) are only tested but not really accessed. If 2PL is applied for XML transactions, long duration locks on these data items may degrade the access performance for XML documents.

Graph-based locking protocol [5,10], rather than distinguishing two phases to ensure serializability, needs additional information from the database. By forming a partial ordering set $D=\{d_1, d_2,\cdots, d_n\}$ for data items d_i, called database graph, graph-based locking protocol accesses data item d_i first before d_j if there exists a relation $d_i \rightarrow d_j$ in set D. Tree locking protocol, whose database graph forms a tree structure, is one special case of graph-based protocol. In tree locking protocol, a data item can only be locked once by the same transaction and a transaction can release a lock and subsequently obtain another lock. However, if a transaction released a lock on a data item, it can no longer relock the data item. Due to these properties, the graph-based protocol is not suitable for XML documents. For example, in XPath or DOM, nodes in XML documents may be tested or accessed frequently. Long duration locks on nodes without early release may degrade the performance.

Multi-granularity locking protocol (MGL) [4,8] considers data items as an ordering abstraction with different granularity. A data item with coarse granularity includes many smaller ones with finer granularity. Locks on data items of coarse granularity allow a transaction to access all of the decedent data items under the locked data items. Multi-granularity locking protocol is not suitable for XML documents. For example, predicates in XPath refine the results of each location steps in XPath. That is, not all decedent nodes of a node are accessed. The locking policy mentioned above locks unnecessary nodes and thus lowers down the performance.

Graph-based locking, tree locking or multi-granularity locking protocols are not suitable for XML documents since the access behavior of XML documents are not considered in these protocols. For accessing XML documents, XPath and DOM are the two most important methods. XPath [3] is a language to locate desired nodes in XML documents. DOM [11] is a platform- and language-neutral interface that provides a standard set of objects to represent XML documents and allows programs to dynamically access and update the content and structure of XML documents. They both are recommendations of W3C.

Knowing the access behavior of XPath or DOM can help to achieve better performance for accessing XML documents. In [7], a concurrency control protocol based on XPath, called XLP, is proposed. XLP has the features of richer lock modes, refined lock granularity, lower lock conflict and lock conversion. In [7], XLP is shown to outperform 2PL by 84.4% and the tree locking protocol by 95.6% on average. Generally speaking, XPath reveals finer access behavior than DOM, but DOM provides more convenient interface to access XML documents. The goal of this study is to propose another new concurrency control protocol for DOM and to compare the two protocols based on XPath and DOM, respectively.

The rest of this paper is organized as follows. Section 2 presents the XPath and XLP protocol. Section 3 proposes a concurrency control protocol DLP based on DOM. Section 4 analyzes both XLP protocol and DLP protocol. Section 5 illustrates the simulation results of XLP and DLP. Finally, Section 6 concludes this study.

2. XPath Locking Protocol

XPath is a language to locate desired nodes in XML documents. XPath locking protocol (XLP) takes into consideration the access patterns in XPath. The XPath model and XLP protocol are described shortly in this section. Correctness and serializability of XLP can be found in [7].

2.1 XPath

XPath models an XML document as a tree of nodes. XPath consists of a location path [3]. A location path consists of a sequence of one or more location steps [3], which are separated by /. Each location step begins with a set of nodes, called context nodes. A location step then generates its result, a set of nodes, which in turn are the context nodes of the next location step in the path.

Each location step is represented by: *Axis*::*Node-Test*[*Predicate*], where *Axis* specifies the tree relationship between the context node of the previous location step and the selected node type is identified by *Node-Test*. *Predicate* is used to refine the results.

For example, the location path, */child::name*[*firstname="CHEN"*]*/child::age*, starts from the root / and consists of two location steps with the first one predicated. The context node of the first step is the root and the context nodes of the second step are predicated nodes of *name* starting from / to *name*. Results of this location path are nodes of *age* starting from / to *name* and *age* with a predication on *firstname*.

According to the XPath model, we classify three types of nodes: pass_by nodes, context nodes and destination nodes. Pass_by nodes are those nodes involved in an XPath. Context nodes with respect to each location step are those nodes to be tested by *Node-Test* and to be predicated by *predicate* in an XPath expression. Destination nodes are the nodes to be processed for reading, updating, inserting or deleting. Formal definitions for the three types of nodes are described as follows.

Definition: The set of *context nodes* of a location step $S_{i,j}$ of location path L_j, denoted by $C(S_{i,j})$, includes nodes that $S_{i,j}$ begins with. In fact, $C(S_{1,j})$ is the root if $S_{1,j}$ starts from the root. The set of

mid-result nodes of a location step $S_{i,j}$ of location path L_j, denoted by $M(S_{i,j})$, includes those nodes satisfying Node-Test in $S_{i,j}$. The set of *result nodes* of $S_{i,j}$, denoted by $R(S_{i,j})$, is the selection of $M(S_{i,j})$ after refining by the predicate of $S_{i,j}$. In fact, $R(S_{i,j}) = C(S_{i+1,j})$.

Definition: The set of *pass-by nodes* in a location path L_j, denoted by $N_p(L_j)$, includes those nodes involved in L_j. The set of *destination nodes*, denoted by $N_d(L_j)$, is the search results of L_j. In fact, $N_d(L_j)=R(S_{|Lj|,j})$, where $| L_j |$ is the length of L_j.

In the next subsection, we propose our XPath locking protocol (XLP) for concurrently executing transactions on XML documents.

2.2 XPath Locking Protocol (XLP)

XLP is a lock-based protocol. In XLP, a transaction consists of location paths, which consists of location steps. A transaction in XLP is defined as follows.

Definition: A *transaction T*, denoted by $T=\{(O_i, L_i)\}$, in XLP is defined as an ordered set of pairs (O_i, L_i) where the operation O_i operates on location path L_i, and $O_i \in$ {Read, Write, Insert, Delete}.

Lock modes in XLP and the XLP protocol is described as follows.

2.2.1 Lock Modes in XLP

There are five lock modes in XLP [7]. They are P-, R-, W-, I- and D- locks. Pass lock (P-lock) is invented for $N_p(L_j)$. Only node types (i.e. tag name of the node) are checked for nodes in $N_p(L_j)$. P-locks are invented to provide better concurrency for nodes in $N_p(L_j)$. Read lock (R-lock), write lock (W-lock), insertion lock (I-lock) and deletion lock (D-lock) are applied to $N_d(L_j)$ in a location path L_j for reading, writing, inserting and deleting, respectively. The five lock modes in XLP are stated as follows. Their lock compatibility matrix is listed in Table 1. The matrix is symmetric.

- P-lock: In a location step $S_{i,j}$, nodes in $C(S_{i,j})$ are locked by P-lock before the execution of $S_{i,j}$. Nodes in $M(S_{i,j})$ are locked by P-lock if they do not conflict with locks held by other

transactions. P-locks on nodes in the difference of two sets, $M(S_{i,j})$-$R(S_{i,j})$, are released after the execution of $S_{i,j}$ because these nodes are no longer used. $R(S_{i,j})$, whose nodes are locked by P-lock, becomes $C(S_{i+1,j})$. P-locks on nodes in $C(S_{i,j})$ can be released after $S_{i,j}$ for better performance.

P-locks on nodes in $C(S_{|Lj|,,j})$ are upgraded to R-, W-, I- or D-locks as necessary. P-lock is compatible with P-, R-, W- and I-lock but not compatible with D-lock.

- R-lock: Operation *(Read*, L_j*)* in a transaction T must receive an R-lock on destination nodes in $N_d(L_j)$. Only the last step $S_{|Lj|,j}$ may read destination nodes in $N_d(L_j)$. Both of their node types and contents (value and attributes) in $N_d(L_j)$ are read. R-lock is compatible with P-lock, R-lock and I-lock.
- W-lock: Operation *(Write*, L_j*)* in a transaction T must receive a W-lock on $N_d(L_j)$. A W-lock also implies an I-lock or D-lock. W-lock is compatible with P-lock and I-lock.
- I-lock: Operation *(Insert*, L_j*)* in a transaction T must receive an I-lock on $N_d(L_j)$.

  I-lock is not compatible with I-lock or D-lock since they both modify the structure of a node.
- D-lock: Operation *(Delete*, L_j*)* in a transaction T must receive a D-lock on $N_d(L_j)$. All child nodes under the node are also deleted. D-lock is not compatible with D-lock.

Table 1. Lock compatibility matrix of XLP.

lock being held / requested lock	P	R	W	I	D
P	○	○	○	○	×
R	○	○	×	○	×
W	○	×	×	○	×
I	○	○	○	×	×
D	×	×	×	×	×

2.2.2 XLP Protocol

XLP is defined by the following six rules.

Two-phase Locking Rule. XLP observes the two-phase locking protocol, except for P-locks.

Compatibility Rule. Each location step S_i can receive a particular type of lock if the compatibility matrix in Table 1 is respected. Once the appropriate lock is received, a location step S_i can be executed.

Granularity Rules.

(1) Locking granularity of P- or I- locks on a node is the node itself only.

(2) Locking granularity of R- or W- locks on a node includes the contents and attributes of the node.

(3) Locking granularity of D-locks on a node is the node itself as well as the subtree rooted at itself.

Lock Rule. Nodes in $M(S_{i,j})$ in the location step S_i are all locked in P-lock.

Upgrade Rules.

(1) The locks on $N_d(L_j)$ are upgraded to I-locks directly before inserting nodes into $N_d(L_j)$.

(2) The locks on $N_d(L_j)$ are upgraded to R- or W-locks if Granularity Rule (2) is satisfied.

(3) The locks on $N_d(L_j)$ are upgraded to D-locks if Granularity Rule (3) is satisfied.

Release Rules.

(1) P-locks on $M(S_{i,j})$-$R(S_{i,j})$, the difference of $M(S_{i,j})$ and $R(S_{i,j})$, are released after the location step $S_{i,j}$ finishes.

(2) P-locks on $\cup C(S_{i,j})$, the union of $C(S_{i,j})$ for all i and j, are released after the location path L_j finishes.

(3) R-, W-, I- or D-locks are released only in the shrinking phase of a transaction.

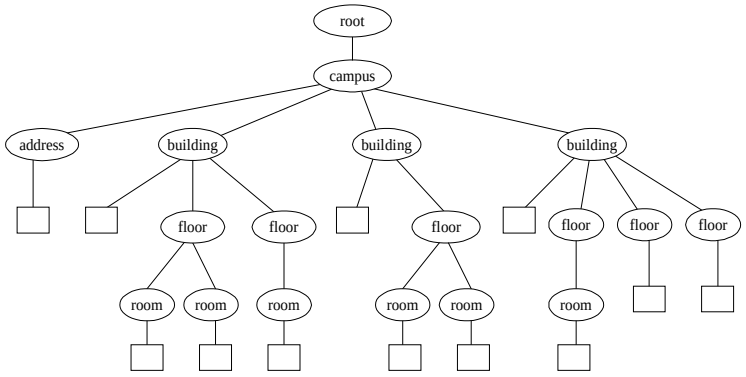

Figure 1. An XML document example for a campus.

Consider Figure 1 as an example. Figure 1 represents the buildings in a campus. There are three buildings in the campus, where the first building has two floors, the second one has one floor and the third one has three floors. Also, there are two rooms in the first floor of the first building. Suppose that there are two transactions *T1* and *T2*, where *T1* reads the content of the floor 2 in the building 1, for example, the description of the floor 2. *T2* updates the content of the building 1, for example, the history of the building 1. *T1* and *T2* are represented as follows.

$T1$={($Read$, L_1)}, where L_1={$S_{1,1}$, $S_{2,1}$, $S_{3,1}$}, $S_{1,1}$="*child*::*campus*", $S_{2,1}$="*child*::*building*[*position()=1*]", and $S_{3,1}$="*child*::*floor*[*position()=2*]". $T2$={($Write$, L_2)}, where L_2={$S_{1,2}$, $S_{2,2}$}, $S_{1,2}$="*child*::*campus*", and $S_{2,2}$="*child*::*building*[*position()=1*]".

Figure 2(a) illustrates a possible schedule for transactions *T1* and *T2*. For simplicity we write *Node-type*$|_{predicate}$ to denote the selection of nodes of *Node-type* on *predicate*. $S_{1,1}$ and $S2_{2,1}$ both require P-locks on the nodes of type *campus* and *building*$|_{position()=1}$. However, $S_{2,2}$ upgrades P-locks to W-locks on *building*$|_{position()=1}$. According to Table 1, P-locks and W-locks are compatible. Therefore, *T1* and *T2* can be executed concurrently.

Consider Figure 1 for another example. Suppose that there are two other transactions *T3* and *T4*, where *T3* executes before *T4*, *T3* writes the last *floor* of *building* 3, and *T4* deletes *building* 3. T3 and *T4* are represented as follows.

$T3$={($Write$, L_3)}, where L_3={$S_{1,3}$, $S_{2,3}$, $S_{3,3}$}, $S_{1,3}$="*child*::*campus*", $S_{2,3}$="*child*::*building*[*position()=3*]", and $S_{3,3}$="*child*::*floor*[*position()=last()*]". $T4$={($Delete$, L_4)},

where L_4={$S_{1,4}$, $S_{2,4}$}, $S_{1,4}$="*child*::*campus*", and $S_{2,4}$="*child*::*building*[*position()=3*]."

Figure 2(b) illustrates a possible schedule for transactions *T3* and *T4*. *T3*and *T4* cannot be executed concurrently since there is P-lock by $S_{3,3}$ and a D-lock by $S_{2,4}$. According to Table 1 and the Granularity Rules (1) and (3) in XLP, P-locks and D-locks are not compatible. The deletion can thus not be executed until *T3* finishes and releases its P-locks on the root, $C(S_{2,3})$, $C(S_{3,3})$ and W-locks on $N_d(L_3)$. Note that $S_{2,4}$:Upgrade-D(*building*$|_{position()=3}$) must be executed after $S_{3,3}$:Write(*floor*$|_{position()=last()}$) since they are conflicting operations.

T1	T2
$S_{1,1}$:Lock-**P**(*campus*)	
	$S_{1,2}$:Lock-**P**(*campus*)
	$S_{2,2}$:Lock-**P**(*building*$|_{position()=1}$)
$S_{2,1}$:Lock-**P**(*building*$|_{position()=1}$)	
$S_{1,1}$:**unlock**(*campus*)	
	$S_{1,2}$:**unlock**(*campus*)
	$S_{2,2}$:**Upgrade-W**(*building*$|_{position()=1}$)
$S_{3,1}$:Lock-**P**(*floor*$|_{position()=2}$)	
	$S_{2,2}$:**Write**(*building*$|_{position()=1}$)
	$S_{2,2}$:**unlock**(*building*$|_{position()=1}$)
$S_{2,1}$:**unlock**(*building*$|_{position()=1}$)	
$S_{3,1}$:**Upgrade-R**(*floor*$|_{position()=2}$)	
$S_{3,1}$:**Read**(*floor*$|_{position()=2}$)	
$S_{3,1}$:**unlock**(*floor*$|_{position()=2}$)	

Figure 2(a). A schedule for transactions T1 and T2 under XLP.

T3	T4
$S_{1,3}$:Lock-**P**(*campus*)	
	$S_{1,4}$:Lock-**P**(*campus*)
$S_{2,3}$:Lock-**P**(*building*$|_{position()=3}$)	
	$S_{2,4}$:Lock-**P**(*building*$|_{position()=3}$)
$S_{1,3}$:**unlock**(*campus*)	
	$S_{1,4}$:**unlock**(*campus*)
$S_{3,3}$:Lock-**P**(*floor*$|_{position()=last()}$)	
$S_{2,3}$:**unlock**(*building*$|_{position()=3}$)	
$S_{3,3}$:**Upgrade-W**(*floor*$|_{position()=last()}$)	
$S_{3,3}$:**Write**(*floor*$|_{position()=last()}$)	
$S_{3,3}$:**unlock**(*floor*$|_{position()=last()}$)	
	$S_{2,4}$:**Upgrade-D**(*building*$|_{position()=3}$)
	$S_{2,4}$:**Delete**(*building*$|_{position()=3}$)
	$S_{2,4}$:**unlock**(*building*$|_{position()=3}$)

Figure 2(b). Another schedule for transactions T3 and T4 under XLP.

3. DOM and DOM Locking Protocol

In this section, we will describe the locking protocol for DOM model for accessing XML documents by programs. The DOM locking protocol (DLP) is fundamentally modified from XLP.

3.1 Document Object Model (DOM)

DOM [11], which is one of the W3C recommendations, is a platform- and language-neutral interface that provides a standard set of objects to represent XML documents and allows programs to dynamically access and update the content and structure of XML documents [11]. Specifications of DOM are defined in OMG IDL (Interface Definition Language)[12] for language-independency.

Basically speaking, DOM models documents as a logical structure like a forest, which can contain more than one tree. Interfaces of DOM dominate the access behavior for accessing the forest structure, including methods *getparent*(), *getfirstchild*(), *nextsibling*() and *previoussibling*(), etc..

Unlike the regular behavior of accessing XML documents in XPath, DOM accesses XML documents in programming ways. In other words, its access behavior, which depends on the statements in programs, is numerous. This makes its access behavior among objects undetectable. But, its access behavior to a single object is useful to get better degree of concurrency. We point out three phases for DOM to access an object: entering, processing and exiting phases. Our DOM locking protocol extends XLP by distinguishing the three phases for different access modes.

In the next section, we propose our DOM locking protocol (DLP) for concurrently executing transactions on XML documents.

3.2 DOM Locking Protocol (DLP)

There are three phases for DOM to access an object: entering phase, processing phase, and exiting phase. In the entering phase, DOM locates the node that is to be processed later for being

read, written, deleted or inserted in the processing phase. In the exiting phase, DOM finishes the processing phase.

DLP is an extension of XLP. The five lock modes in Table 1 are adopted in DLP. For better concurrency, DLP locks nodes by different lock modes for different phases. In the entering phase, a node is locked by P-lock first. In the processing phase, a node is to be read, written, deleted or inserted. The node is locked by respective lock modes of R-, W-, D-, or I- locks. In the exiting phase, the lock in the node is released. We classify accessing nodes of attributes *parentnode*, *firstchild*, *nextsibling* or *previoussibling* described in DOM Level 1 [11] into the entering phases. Execution of methods of *replacechild*(), *appendchild*(), *removechilde*() or *insertbefore*(), etc. is classified into the processing phase. And, unfortunately none operation in DOM Level 1 exactly indicates the end of accessing nodes. As a result, the exiting phase of a node is at the end of a transaction. The scope of a transaction in DLP is defined to be the scope of a program.

DLP modifies XLP and includes the following six rules.

Two-phase Locking Rule.

DLP observes the two-phase locking protocol, except for P-locks.

Compatibility Rule.

Each node can receive a particular type of lock if the compatibility matrix is respected. Once the appropriate lock is received, the node can be processed.

Granularity Rules.

(1) Locking granularity of P- or I- locks on a node is the node itself only.

(2) Locking granularity of R- or W- locks on a node includes the contents and attributes of the node.

(3) Locking granularity of D-locks on a node is the node itself as well as the subtree rooted at itself.

Lock Rule.

A node is locked by P-lock for its entering phase.

Upgrade Rules.

(1) The lock of a node is upgraded to I-locks before inserting a new child node to the node if Granularity Rule (1) is satisfied.

(2) The lock of a node is upgraded to R- or W-locks if Granularity Rule (2) is satisfied.

(3) The lock of a node is upgraded to D-locks if Granularity Rule (3) is satisfied.

Release Rules.

(1) P-lock of a node can be released in the exiting phase of the node and at any time of the transaction.

(2) R-, W-, I- or D-locks of a node are released only in the exiting phase of the node and the shrinking phase of the transaction.

Take Figure 1 as an example. Suppose that there are two transactions *T1'* and *T2'*, where *T1'* reads the content of the floor 2 in the building 1, and *T2'* updates the content of the building 1. Programs for accessing objects in *T1'* and *T2'* are listed in Figure 3(a).

In *T1'*, the program first gets the node 'campus' and searches all its children to get the node 'building' which is the first building. Then, the program searches all the children of the first building to get the node 'floor' which is the second floor. Finally, the program reads the content of the desired floor. In *T2'*, the program first gets the node 'campus' and searches all its children to get the node 'building' which is the first building. Then, *T2'* writes the new content to building 1. The comments shown in italic characters indicate the proper locks that the involved nodes need.

A possible schedule for *T1'* and *T2'* is shown in Figure 3(b), in which nodes with the same tag name are distinguished by numbering for clarity. In the schedule, Upgrade-W(building 1) for writing building 1 and Upgrade-R(floor 2) for reading floor 2 in building 1 do not conflict according to Table 1. Therefore, *T1'* and *T2'* can execute concurrently.

T1'	T2'
campus=document.documentelement; *//P-Lock* xnode=campus.firstchild; *//P-Lock* // find building 1 bu=0; **While** not (bu=1 or xnode=null) { **If** (xnode.nodename='building') {bu=bu+1;}; **If** not (bu=1) {xnode=xnode.nextsibling;}; *//P-Lock* }; **If** (xnode=null) {return;}; // find floor 2 in building 1 ynode=xnode.firstchild; *//P-Lock* f=0; **While** not (f=2 or ynode=null) { **If** ynode.nodename='floor' {f=f+1;}; ynode=ynode.nextsibling; *//P-Lock* }; **If** (ynode=null) {return;} **Else** { **read**(ynode); *//Upgrade to R-Lock* }; *// release all locks here at the end of the program*	campus=document.documentelement; *//P-Lock* xnode=campus.firstchild; *//P-Lock* // find building 1 bu=0; **While** not (bu=1 or xnode=null) { **If** (xnode.nodename='building') { bu= bu+1;}; **If** not (bu=1) {xnode=xnode.nextsibling;}; *//P-Lock* }; **If** (xnode=null) {return;} **Else** { **write**(xnode); *//Upgrade to W-Lock* }; *// release all locks here at the end of the program*

Figure 3(a). Two transactions T1' and T2' by DOM.

T1'	T2'
Lock-**P**(*campus*)	
Lock-**P**(*address*)	
Lock-**P**(*building 1*)	
	Lock-**P**(*campus*)
Lock-**P**(*building 2*)	
	Lock-**P**(*address*)
	Lock-**P**(*building 1*)
Lock-**P**(*floor 1*)	
	Upgrade-W(*building 1*)
Lock-**P**(*floor 2*)	
Upgrade-R(*floor 2*)	
unlock(*floor 2*)	
unlock(*floor 1*)	
unlock(*building 2*)	
unlock(*building 1*)	
unlock(*address*)	
unlock(*campus*)	
	unlock(*building 1*)
	unlock(*address*)
	unlock(*campus*)

Figure 3(b). A schedule for transactions T1' and T2' in Figure 3(a) under DLP.

Consider Figure 1 for another example. Suppose that there are two other transactions *T3'* and *T4'*, where *T3'* executes before *T4'*. *T3'* writes the last *floor* of *building* 3, and *T4'* deletes *building* 3. Programs for accessing objects in *T3'* and *T4'* are listed in Figure 4(a).

In *T3'*, the program first gets the node 'campus' and searches all its children to get the node 'building' which is the third building. Then, the program searches all the children of the third building to get the node 'floor' which is the last one. Finally, the program writes the new content to the last floor. In *T4'*, the program first gets the node 'campus' and searches all its children to get the node 'building' which is the third building. Then, *T4'* deletes building 3.

A possible schedule for *T3'* and *T4'* is shown in Figure 4(b). In the schedule, Upgrade-W(floor 3) for writing the last floor in building 3 and Upgrade-D(building 3) for deleting building 3 conflict according to Table 1 and the Granularity Rules (2) and (3) in DLP. Therefore, *T3'* and *T4'* cannot execute concurrently.

In fact, the degree of concurrency for DLP will be influenced by the access manner in the program. For example, if *T1'* is written by DFS(depth first search), then all nodes of the XML document are locked. But in fact only a few nodes are actually required and accessed. Comparisons between XLP and DLP are discussed in the next section.

T3'	T4'
campus=document.documentelement; *//P-Lock*	campus=document.documentelement; *//P-Lock*
xnode=campus.firstchild; *//P-Lock*	xnode=campus.firstchild; *//P-Lock*
//find building 3	//find building 3
bu=0;	bu=0;
While not (bu=3 or xnode=null) {	**While** not (bu=3 or xnode=null) {
If (xnode.nodename='building') {bu=bu+1;};	**If** (xnode.nodename='building') {bu=bu+1;};
If not (bu=3) {xnode=xnode.nextsibling;}; *//P-Lock*	**If** not (bu=3) {xnode=xnode.nextsibling;}; *//P-Lock*
};	};
If (xnode=null) {return;};	**If** (xnode=null) {return;}
	Else {
//find the last floor in building 3	**delete**(xnode); *//Upgrade to D-Lock*
ynode=xnode.firstchild; *//P-Lock*	};

```
f=0;
While not (ynode=null) {
   If ynode.nodename='floor' { f=f+1;};
   ynode=ynode.nextsibling;            //P-Lock
};
If (f=0) {return;}
Else {
    write(ynode);            //Upgrade to W-Lock
};

// release all locks here at the end of the program
```

```
// release all locks here at the end of the program
```

Figure 4(a). Two transactions T3' and T4' by DOM.

T3'	T4'
Lock-**P**(*campus*)	
Lock-**P**(*address*)	
Lock-**P**(*building 1*)	
	Lock-**P**(*campus*)
Lock-**P**(*building 2*)	
	Lock-**P**(*address*)
	Lock-**P**(*building 1*)
Lock-**P**(*building 3*)	
Lock-**P**(*floor 1*)	
	Lock-**P**(*building 2*)
	Lock-**P**(*building 3*)
Lock-**P**(*floor 2*)	
Lock-**P**(*floor 3*)	
Upgrade-W(*floor 3*)	
unlock(*floor 3*)	
unlock(*floor 2*)	
unlock(*floor 1*)	
unlock(*building 3*)	
unlock(*building 2*)	
unlock(*building 1*)	
unlock(*address*)	
unlock(*campus*)	
	Upgrade-D(*building 3*)
	unlock(*building 3*)
	unlock(*building 2*)
	unlock(*building 1*)
	unlock(*address*)
	unlock(*campus*)

Figure 4(b). A schedule for transactions T3' and T4' in Figure 4(a) under DLP.

4. Comparisons of XLP and DLP

In this section, we compare XLP and DLP in two aspects: how early can a lock be released and how many nodes should be locked.

First, XLP may release locks earlier than DLP. XLP has the finer control of a transaction than DLP. In XLP, a transaction consists of location paths, each of which consists of location steps. Each location step clearly identifies the types of nodes to be accessed. Also predicates help refine the result of a location step. In DLP, the scope of a transaction is the scope of the program. Only at the end of a program can DLP make sure that there is no more access to objects. No further refinement of a transaction can help DLP release locks earlier. As a result, according to Release Rule (1) in XLP, P-locks on undesired nodes may be released earlier at the end of a location step. But, for DOM Level 1, P-locks in DLP can only be released at the end of a transaction since there is no way to indicate the exiting phase in DOM Level 1. Also, Release Rule (2) in XLP releases unnecessary P-locks after each location path of a transaction, but DLP does not have the rule.

Secondly, XLP locks fewer nodes than DLP. Each location step clearly identifies the types of nodes to be accessed, but the number of locks by DLP depends on the access manner in a program. The question is twofold. The first is the access manner programmed by a programmer. If a programmer uses the depth-first search for a simple query, for example transaction T2' in Figure 3(a), every node in an XML document is at least locked by P-lock and some nodes may be upgraded to proper lock modes later. Too many unnecessary locks thus happen in DLP. The second is the access manner for the desired results. For example, a user wants to read those buildings having at least one floor, which in turn has at least one room. This query requires traversing almost every 'building' node in the XML document. The depth-first search perhaps is the simplest method for the query, but it costs too much since every node is locked by at least P-lock and cannot be released earlier. In XLP, such a query can be expressed in the location path "/*child*::*campus*/*child*::*building*[*child*::*floor*/*child*::*room*]." Only involved nodes are locked and

unnecessary nodes can be released earlier. The advantage comes from the richer semantics implied in the location path of XPath.

In the next section, XLP and DLP are compared by simulation.

5. Simulation Results

To illustrate the performance of XLP and DLP, we write a simulator to compare the two protocols. In the simulation model, there are 100 transactions executing for each experiment. Transactions by DOM use the depth-first search to traverse the XML document. Document size ranges from 13 nodes to 137257 nodes. Three experiments are conducted to compare their performance under different size of destination nodes, different read/write ratios and different sizes of XML documents.

The first experiment illustrates the impact of different size of destination nodes (i.e. the result nodes of a transaction). The result is shown in Figure 5 where 100 transactions are executed with the write/read ratio of 0.33 in a transaction and the document size of 781 nodes. The transaction execution time for both protocols is exponentially dependent on the destination node size. This is because more locks are required if the number of destination nodes get larger and also more conflicts result in performance degradation. In Figure 5, XLP performs faster than DLP about 69% on average. The reason is that by Release Rules (1) and (2), XLP may release locks earlier on nodes other than destination nodes after a location path finishes, but DLP must hold locks on all nodes till the end of the transaction.

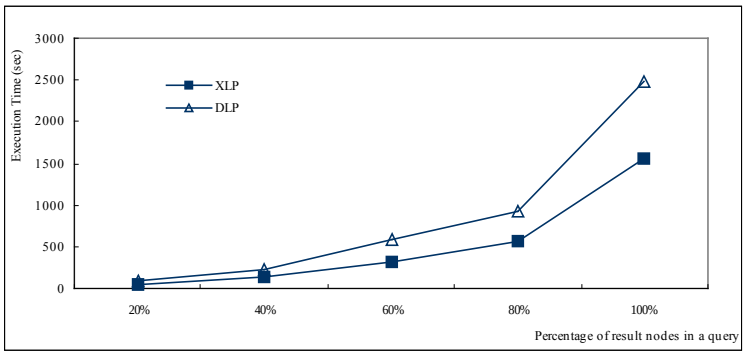

Figure 5. Concurrency of XLP and DLP for different size of destination nodes.

The second experiment illustrates the impact of different write/read ratio. Write operations in the experiment includes update, insert and delete operations. The result is shown in Figure 6 where 100 transactions are executed with XPath length of 5. The XML document size for such XPath length is about 800 nodes. Execution time of both protocols grows as the write/read ratio grows. Two curves of XLP and DLP are almost parallel. This is because lock modes for the two protocols are the same and execution time spent on unnecessary locks of nodes is small when there are few destination nodes and low delete ratio. In Figure 6, XLP outperforms DLP by 68.7% on average.

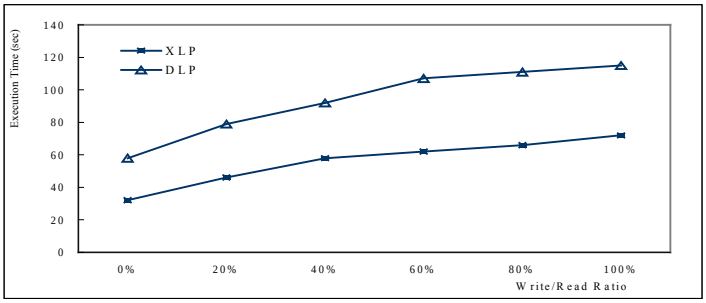

Figure 6. Performance comparison under different read/write ratio.

The last experiment illustrates the overall performance under different XML document sizes. The simulation result is shown in Figure 7. The write/read ratio for this experiment is 0.33 and the number of destination nodes is 20% of total nodes in an XML document. In Figure 7, XLP is not very sensitive to the document size but DLP is. XLP outperforms DLP about 109.2% on average. This is because in this simulation the depth-first search is adopted to traverse the whole document. Since DLP does not release locks earlier, such a situation generates more conflicts when the document size increases.

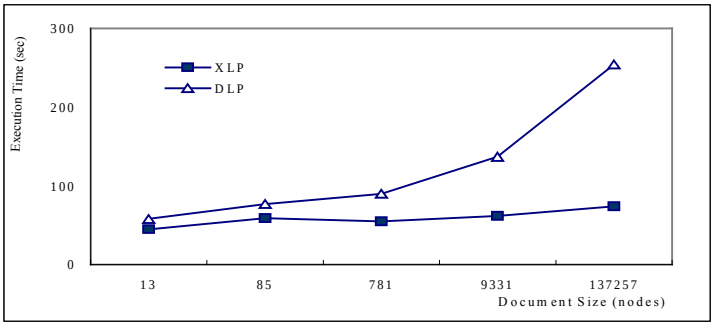

Figure 7. Overall performance under different document size.

6. Conclusions

XML has become the most important technique to exchange data in WWW. Providing a high degree of concurrency in XML databases is crucial in many applications. In this paper two concurrency control protocols, XLP and DLP, are proposed and compared for XML documents. XLP, based on XPath suggested by W3C, has the features of richer lock modes, refined lock granularity, lower lock conflict and lock conversion. DLP, based on DOM model suggested also by W3C, is extended from XLP. Simulation results show that XLP outperforms DLP. For XML documents of different sizes, XLP outperforms DLPL by 109.2% on average. Future work includes take the access behavior provided by DOM Level 2 and Level 3 into consideration for DLP.

References

[1] P. Bernstein, V. Hadzilacos and N. Goodman, Concurrency Control and Recovery in Database System, Addison-Wesley, 1987.

[2] P. Bernstein, D. Shipman and W. Wong, "Formal Aspects of Serializability in Database Concurrency Control," IEEE Trans. on Software Engineering, Vol. 5, No. 3, pp.203-216, 1979.

[3] J. Clark, S. DeRose, "XML Path Language (XPath) Version 1.0," World Wide Web Consortium (W3C) Recommendation, 1999, available at http://www.w3.org/TR/XPath.

[4] J. Gray, R. Lorie, G.. Putzolu and I. Traiger, "Granularity of Locks and Degrees of Consistency in a Shared Database," Modeling in Database Management System, Elsevier North-Holland, pp. 365-395, 1976.

[5] H. Korth, "Deadlock Freedom Using Edge Locks," ACM Trans. Database System, Vol. 7, No. 4, pp. 632-652, 1982.

[6] H. Korth, "Locking Primitives in a Database System," J.ACM, Vol.20, No. 1, pp. 55-79, 1983.

[7] K.-F. Jea, S.-H. Wang, S.-Y. Chen, "Concurrency Control in XML Document Databases:

XPath Locking Protocol," Proc. of 2002 Symposium on Digital Life and Internet Technologies (Abstract Collections), pp.112, 2002.

[8] S.-Y. Lee and R.-L. Liou, "A Multi-granularity Locking Model for Concurrency Control in Object-oriented Database Systems," Proc. 2nd FAR-FAST Workshop on Future Database Systems, pp.158-167, 1992.

[9] R. Ramakrishnan and J. Gehrke, Database Management, 2nd ed., McGraw-Hill, 2000.

[10] A. Silberschatz, H. Korth and S. Sudarshan, Database System Concepts, 4th ed., McGraw-Hill, 2001.

[11] "Document Object Model (DOM)," http://www.w3.org/DOM/.

[12] OMG (Object Management Group), http://www.omg.org/corba/corbiiop.htm.