A high concurrency XPath-based locking protocol for XML databases

Kuen-Fang Jea*, Shih-Ying Chen

Department of Computer Science, National Chung-Hsing University, 250, Kuo-Kuang Road, Taichung 40227, Taiwan, ROC

Received 27 January 2005; revised 24 July 2005; accepted 2 August 2005
Available online 16 September 2005

Abstract

Providing efficient access to XML documents becomes crucial in XML database systems. More and more concurrency control protocols for XML database systems were proposed in the past few years. Being an important language for addressing data in XML documents, XPath expressions are the basis of several query languages, such as XQuery and XSLT. In this paper, we propose a lock-based concurrency control protocol, called XLP, for transactions accessing XML data by the XPath model. XLP is based on the XPath model and has the features of rich lock modes, low lock conflict and lock conversion. XLP is also proved to ensure conflict serializability. In sum, there are three major contributions in this paper. The proposed XLP supports most XPath axes, rather than simple path expressions only. Conflict conditions and rules in the XPath model are analyzed and derived. Moreover, a lightweighted lock mode, P-lock, is invented and integrated into XLP for better concurrency.

© 2005 Elsevier B.V. All rights reserved.

Keywords: XML document databases; Concurrency control; Locking protocol; XPath

1. Introduction

XML has become a standard format for data exchange on the Internet. Many applications such as Science, Biology and Business require XML to represent data in their disciplines. Since data in these areas are usually very large, it is common to store them in databases for efficient retrieval and storage.

Concurrency control is one of the most important techniques to achieve efficient access (including read and write operations) in database systems. This is especially true of current web-based XML applications, as it can provide higher transaction throughput and better scalability for XML database servers. By scheduling incoming transactions, concurrency control techniques allow transactions to be executed concurrently and thus diminish the waiting time. The correct execution of concurrent transactions scheduled by a concurrency control protocol can be ensured by serializability [8,12,20,22].

Many concurrency control protocols were proposed for traditional database systems. Among them, the lock-based protocols [1,12,17–19,22] are widely used. In these protocols, write locks and read locks are two fundamental types of locks. A transaction can proceed if the requested lock on the desired object is compatible with locks held by other transactions on the same object; otherwise, the transaction must wait until other transactions release the incompatible locks. The most famous lock-based protocol is the two-phase locking protocol (2PL) [7,12,18,22]. In 2PL, a transaction acquires locks only in the growing phase and releases locks in the shrinking phase to ensure serializability.

Graph-based locking protocols [17] treat data items in the database as a partial ordering set $D = \{d_1, d_2, \ldots, d_n\}$. The set $D$, called database graph, forms a directed acyclic graph (DAG) where an edge $d_i \rightarrow d_j$ indicates item $d_i$ must be accessed before item $d_j$. The tree locking protocol [21], whose database graph displays a tree structure, is a special case of graph-based protocols. A transaction can release a lock and subsequently obtain another lock. However, if a transaction previously released a lock on a data item, it can no longer relock that data item. Multi-granularity locking protocols [11,19] consider data items as an ordering abstraction with different granularity. A data item with
coarse granularity includes many smaller ones with finer granularity. Transactions only need to acquire the lock on data items of coarse granularity, and then they can access all of the descendant data items.

Unfortunately, traditional concurrency control protocols are not tailored to XML database systems and therefore not optimized for the execution of XML transactions. The research in this area has rapidly gained importance in recent years, and several techniques [2–6,10,13–16] have been proposed. In general, these concurrency control techniques deal with three different types of access methods, including DOM [23], DataGuides [9] and XPath [3].

DOM [23] supports a standard set of application program interfaces (APIs) to manipulate XML documents. It allows programs to dynamically access and update the content and structures of XML documents. By exploiting the DOM access methods, [13,14,16] proposed protocols with a rich set of locking modes to achieve high concurrency. In contrast, DataGuides [9] is the data abstraction of an XML document. By exploiting the structural relationships among nodes in DataGuides, a 2PL-based DGLOCK protocol [10] was proposed. In addition to shared locks and exclusive locks, the intension locks are included in the protocol.

Besides DOM and DataGuides, XPath [3] is a popular language for addressing data in XML documents. Several query languages for XML databases are XPath-like query languages, such as XQuery [25] and XSLT [24]. By storing and comparing the database states for each concurrent transaction, [2] proposed a validation-based concurrency control protocol [12,22] for transactions including XPath expressions. The result of each write access in a transaction is reflected in its local database state. A conflict check proceeds to see if there exists any conflict with local database states of other transactions. If a transaction commits, its local database state becomes the new database state. The protocol may be expensive for storing database states and checking conflicts.

On the other hand, [4–6] proposed two locking protocols: path locks propagation (PL-PROP) and path locks satisfaction (PL-SAT). Both are similar except for the read locks being set. PL-SAT requires fewer read locks on the nodes in an XPath expression and leads to more complex checks of conflicts. By contrast, PL-PROP propagates read locks from the root of an XPath expression to its descendents and results in more read locks but simpler checks of conflicts. However, both schedulers can guarantee serializability [4,5]. Based on PL-PROP, a commit scheduler [4,6] and a conflict scheduler [5,6] were proposed. A commit scheduler lets a transaction wait if its request cannot proceed, while a conflict scheduler keeps a transaction proceeding unless it fails.

The work in [13], although based on the DOM model, also proposes a locking procedure to support part of axis operations (including the parent, child and sibling axes) similar to the XPath’s access. Ref. [13] provides ten lock modes comprising shared, exclusive and intension locks: seven lock modes are applied on nodes, while three are on edges between nodes. To access a node, nodes in its path to the root are locked in the up-to-root direction. Edges are locked in both directions. Lock conflicts are checked against the compatibility matrices for both node and edge locks. Its lock granularity includes locks on the context node, the context node plus its direct-child nodes and the context node plus its edges. A lock conversion to a more restrictive one is also allowed.

In this paper, a new lock-based concurrency control protocol, namely XLP (XPath Locking Protocol), is proposed for XML database systems. The access behavior and operation conflicts in XPath expressions are analyzed to increase the concurrency of XPath-like queries in XML transactions. The protocol is intended not only to minimize the number of locks held by releasing them as soon as they are no longer needed, but also to support various concurrency enhancement features including rich lock modes, low lock conflict, and lock conversion.

The rest of this paper is organized as follows. Section 2 reviews the XPath model and describes the terms and notions used in XLP. Section 3 analyzes the conflicts in operations in the XPath model. Section 4 presents the new XLP protocol, while Section 5 analyzes XLP and makes a comparison with other protocols. Finally, Section 6 concludes this study and discusses the future work.

2. Preliminary

2.1. XPath model and XPath expression

XPath [3] models an XML document as a tree of nodes. There are seven types of nodes: the root node, element nodes, text nodes, attribute nodes, namespace nodes, processing instruction nodes, and comment nodes. On the other hand, being a query language, XPath expressions are used to indicate the requested nodes in the XML tree. Basically, an XPath expression includes structural constraints and predicates. The XPath expression consists of a location path [3], which in turn consists of a sequence of one or more location steps separated by the symbol ‘/’. Each location step starts from a set of nodes, called context nodes. A location step is represented by $\text{Axis}::\text{Node-Test}\{\text{Predicate}\}$, where $\text{Axis}$ specifies the node relationships (e.g. parent, child or sibling, etc.) between the context nodes and the nodes whose type is identified by $\text{Node-Test}$. The location path and location steps are considered as structural constraints. In contrast, $\text{Predicate}$ is used to sieve out the results from nodes satisfying the structural constraint $\text{Axis}::\text{Node-Test}$ of a location step. As a result, the set of nodes satisfying both the structural constraint and the predicate of a location step becomes the result of that location step, which in turn becomes the context nodes of the next location step. The result of an XPath expression is
a set of nodes satisfying the structural constraint of the XPath expression.

Three important observations are made from the XPath’s access behavior that may be useful in designing an efficient concurrency protocol for XPath-like queries. First, each axis in the location steps specifies the direction of navigating the path; for example, the child axis navigates down and the ancestor axis navigates up. An efficient protocol must track the changes in axes between successive location steps in order to lock as few nodes as possible. Second, for an element node in an XML document tree, the Node-Test ‘checks’ its tag name as a structural constraint rather than reads/writes its value. As a result, providing different lock nodes to distinguish the access states of nodes may increase concurrency. We note that only the nodes specified in the last location step are actually accessed for processing while nodes specified in other steps are only tested by tag names. Supporting ‘cheaper’ lock nodes for the latter nodes will certainly increase concurrency. Third, the Predicate in a location step possibly filters out some nodes, whose locks can be released earlier for better concurrency.

By considering the access behavior of XPath expressions, our proposed protocol can deal with queries on XML structure, such as elements’ sequence orders and parent/child relationships. In this paper, we mainly deal with the element nodes. Since the seven types of nodes in the XPath model have the same access nature, they (including attribute nodes) can be treated in the same way as the element nodes in our protocol.

2.2. Symbols and definitions

In this study, we use the symbols $S_{i,j}, L_j$ and $l_j$ to model an XPath expression. $S_{i,j}$ denotes the $i$th location step in location path $L_j$ with length $l_j$ (i.e. number of location steps in $L_j$). Hence, location path $L_j$ with $m$ location steps can be denoted by $\langle S_1, S_2, S_3, \ldots, S_m \rangle$, where $m = l_j$.

We define the three sets $C(S_{i,j}), M(S_{i,j})$ and $R(S_{i,j})$ to model nodes explicitly indicated in an XPath expression. $C(S_{i,j})$ denotes the set of context nodes of $S_{i,j}$. With respect to $C(S_{i,j})$, $M(S_{i,j})$ denotes the set of (mid-result) nodes that satisfy the structural constraint Axis::Node-Test of $S_{i,j}$. On the other hand, $R(S_{i,j})$, the set of result nodes of $S_{i,j}$, is the set of nodes in $M(S_{i,j})$ satisfying the Predicate of $S_{i,j}$. In fact, the result nodes of $S_{i,j}$ become the context nodes of $S_{i+1,j}$. That is, $R(S_{i,j}) = C(S_{i+1,j})$.

Further, we use the symbols $M(S_{i,j})$ and $R(S_{i,j})$ to denote the sets of nodes not explicitly indicated in location step $S_{i,j}$ but implicitly visited by the query evaluator when navigating $M(S_{i,j})$ and $R(S_{i,j})$, respectively. The nodes in these sets are called the implicit pass-by nodes. The nodes included in $M(S_{i,j})$ depend on $C(S_{i,j}), M(S_{i,j})$ and the axis in $S_{i,j}$. For the preceding, preceding-sibling, following and following-sibling axes of XPath expression, $M(S_{i,j})$ includes the nodes in paths starting from the root ($/\$) to the nodes in $M(S_{i,j})$ but excluding the root and the nodes in $M(S_{i,j})$, since

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>$L_j$</td>
</tr>
<tr>
<td>$S_{i,j}$</td>
</tr>
<tr>
<td>$l_j$</td>
</tr>
<tr>
<td>$C(S_{i,j})$</td>
</tr>
<tr>
<td>$M(S_{i,j})$</td>
</tr>
<tr>
<td>$M(S_{i,j})$</td>
</tr>
<tr>
<td>$R(S_{i,j})$</td>
</tr>
<tr>
<td>$R(S_{i,j})$</td>
</tr>
<tr>
<td>$N_d(L_j)$</td>
</tr>
</tbody>
</table>

Example 1. Fig. 1 shows an XML document, where circles represent the tag names of elements and rectangles represent the values of elements. We use this example to illustrate the terms defined above. Consider location path $L_1 = \langle /\text{child}: \text{a} /\text{position}() = 2 \rangle$. $L_1$ is decomposed and represented by $L_1 = \langle /\text{S}_1, \text{S}_2, \text{S}_1, \text{S}_2 \rangle$, where $\text{S}_1, \text{child} = /\text{child}: /\text{a}$ and $\text{S}_2, /\text{position}() = 2$. In location step $S_{1,1}$, $C(S_{1,1})$ is the root since location path $L_1$ starts from the root. Then, since the axis of $S_{1,1}$ is child, $M(S_{1,1})$ is $\emptyset$ and $R(S_{1,1})$ is $\emptyset$. By the Node-Test of $S_{1,1}$, $M(S_{1,1})$ is $\{ \langle a \rangle \}$. And, $R(S_{1,1})$ is $\{ \langle a \rangle \}$ since the Predicate in $S_{1,1}$ is omitted. In location step $S_{2,1}$, $C(S_{2,1})$ is $\{ \langle a \rangle \}$. Owing to the descendant axis and the Node-Test
of \( S_{2,1} \), \( M(S_{2,1}) \) is \{(b_2), (b_3), (c_1), (c_2)\} and \( M(S_{2,1}) \) is \{(d_1), (d_2), (d_3)\}. According to the **Predicate** of \( S_{2,1} \), \( R(S_{2,1}) \) is \{(b_2), (c_1)\} and \( R(S_{2,1}) \) is \{(d_2)\}. Finally, \( N_4(L_1) \) equals \( R(S_{2,1}) \), which equals \{(d_2)\}. Since XML allows the same tagname to appear in the sibling nodes, here we use \( \langle \text{tagname}_n \rangle \) to denote the \( n \)th node element (numbering from left to right) of the same tagname at the same XML tree level.

### 3. Operation conflicts in XPath

In this section, we analyze the conflicts between operations in XPath-like queries. Conflicts in operations affect the serializability of a concurrency control protocol.

We define a **transaction** \( T \) as a sequence of pairs of operations \( O_j(x) \) and location path \( L_j \), where each operation \( O_j \) manipulates the destination nodes \( x \in N_4(L_j) \) in \( L_j \). \( O_j \) uses the symbols \( R, W, I \) and \( D \) for **Read**, **Write**, **Insert**, and **Delete** operations. For example, \( T=\langle \langle R(x), L_1 \rangle; (W(x), L_2) \rangle \) means a transaction \( T \) includes a read operation on location path \( L_1 \) followed by a write operation on \( L_2 \).

There exist five different types of operations when evaluating a location path. The **Pass-by** operation is used for the **Node-Test** and **Predicate** in each location step, while the **Read**, **Write**, **Insert**, and **Delete** operations are used for processing the destination nodes. We use \( P_j(x), R_j(x), W_j(x), I_j(x), D_j(x) \) to denote, respectively, the **Pass-by**, **Read**, **Write**, **Insert**, and **Delete** operations on node \( x \) in transaction \( T \). If a rename operation on an element’s tagname is required, it is treated by a delete operation followed by an insertion operation.

In general, a **conflict** \([8,22]\) means a swap of a pair of consecutive operations (from different transactions) in a schedule will lead to a different result from the original one. We use the notation \( \langle A_i(x); B_j(y) \rangle \) to denote a pair of consecutive operations \( A_i \) and \( B_j \) in a schedule which are from transactions \( T_i \) and \( T_j \) and work on nodes \( x \) and \( y \), respectively. Node \( y \) can be \( x \)'s ancestor, \( x \)'s descendent, \( x \) itself, or other nodes. For any two transactions \( T_i \) and \( T_j \) in a schedule, we have the following three **non-conflict conditions** according to the definition of conflict.

1. For any \( x \) and \( y \), the operations in the following pairs, \( \langle P_i(x); P_j(y) \rangle, \langle P_i(x); R_j(y) \rangle \) and \( \langle R_i(x); R_j(y) \rangle \), do not conflict because none of the values of \( x \) and \( y \) are changed after executing these pairs of operations.
2. For any \( x \) and \( y \), the operations in the two pairs \( \langle R_i(x); I_j(y) \rangle \) and \( \langle W_i(x); I_j(y) \rangle \) do not conflict because \( I_j(y) \) modifies only the structure of \( y \) rather than its content. What \( I_j(y) \) concerns is the structure only, while \( R_i(x) \) or \( W_i(x) \) concerns the value of the \( x \)'s content.
3. For any \( y \), if there exists a location step \( S \) in location path \( L \) in \( T \), such that \( x \in R(S) \) and \( x \notin R_i(S) \), the operations in the following three pairs, \( \langle P_i(x); W_i(y) \rangle, \langle P_i(x); I_j(y) \rangle \) and \( \langle P_i(x); D_j(y) \rangle \), do not conflict. This is because node \( y \) is not in the navigation path from the root to \( N_4(L) \), and operations \( W_j(y), I_j(y), \) and \( D_j(y) \) have nothing to do with \( N_4(L) \).

On the contrary, any swap of a pair of operations leads to a conflict in the following four **conflict conditions**. We say they are **exclusive operations** from each other.

1. For any \( x \), swapping operations in the pairs \( \langle R_i(x); W_i(y) \rangle \) and \( \langle R_i(x); D_j(y) \rangle \) leads to a conflict. Since \( W_i(y) \) and \( D_j(y) \) update the \( x \)'s value, they conflict with \( R_i(x) \) on the same node \( x \).
2. For any \( x \), swapping operations in any of the two pairs \( \langle W_i(x); W_j(y) \rangle \) and \( \langle W_i(x); D_j(y) \rangle \) leads to a conflict. The reason is the same as the previous case.
3. For any \( x \), swapping operations in any of the three pairs \( \langle I_i(x); I_j(y) \rangle, \langle I_i(x); D_j(y) \rangle \) and \( \langle D_j(y); D_i(x) \rangle \) leads to a conflict since both **Insert** and **Delete** operations modify the structure of a node.
4. For any \( x \), if there exists a location step \( S \) in location path \( L \) such that \( x \in R(S) \) or \( x \notin R(S) \) in transaction \( T \), \( x \) is a node not sieved out by the **Predicate** of \( S \), swapping operations in any of the following pairs, \( \langle P_i(x); W_i(x) \rangle, \langle P_i(x); I_j(x) \rangle \) and \( \langle P_i(x); D_j(x) \rangle \), leads to a conflict. This is because \( P_i(x) \) tests the node \( x \) (by **Node-Test**) or reads the content of \( x \) (by **Predicate**), while \( W_i(x), I_j(x), \) and \( D_j(x) \) modify the content of \( x \) or the subtree rooted at \( x \).

### 4. XPath locking protocol (XLP)

This section describes the XLP protocol, including its lock modes, lock compatibility matrix, and protocol rules.

#### 4.1. Lock modes in XLP

XLP is a lock-based protocol with five lock modes, denoted by \( P-, R-, W-, I- \), and **D-locks**, which must be acquired before the **Pass-by**, **Read**, **Write**, **Insert**, and **Delete** operations, respectively. The five lock modes of XLP are explained in detail as follows. The conflict analysis in Section 3 leads to the compatibility among these locks.

- **P-lock**: The P-lock is a shared lock designed for the **Pass-by** operation. While executing a location step \( S_{ij} \) of location path \( L \), nodes in the M-set of \( S_{ij} \) (i.e. \( M(S_{ij}) \cap M(S_{ij}) \)) are locked by P-locks. At the final location step of the path, P-locks on the destination nodes are eventually upgraded to R-, W-, I-, or D-locks, depending on the type of operation on the destination nodes. P-locks (for the **Pass-by** operations) are compatible with R-locks (for the **Read** operations) due to the non-conflict condition (1) in Section 3. However, they are conditionally compatible with W-, I- and D-locks (for the **Write**, **Insert**, and **Delete** operations), depending on...
whether the non-conflict condition (3) or the conflict condition (4) holds.

- **R-lock**: The operation \((R(x), L_j)\) in a transaction must acquire R-locks on the destination nodes \(x \in N_d(L_j)\). R-locks are upgraded from P-locks. An R-lock (for the Read operations) is compatible with a P-lock (for the Pass-by operations) and an I-lock (for the Insert operations) due to the non-conflict conditions (1) and (2), respectively.

- **W-lock**: The operation \((W(x), L_j)\) in a transaction must acquire W-locks on the destination nodes \(x \in N_d(L_j)\) in \(L_j\). W-locks are upgraded from P-locks. The W-lock (for the Write operations) is compatible with the I-lock (for the Insert operations) by the non-conflict condition (2), but conditionally compatible with the P-lock (for the Pass-by operations) depending on whether the non-conflict condition (3) or the conflict condition (4) holds.

- **I-lock**: The operation \((I(x), L_j)\) must acquire I-locks on the destination nodes \(x \in N_d(L_j)\). I-locks are upgraded from P-locks. They are compatible with R- and W-locks (for the Read and Write operations) by the non-conflict condition (2), but incompatible with I-locks and D-locks (for the Insert and Delete operations) by the conflict conditions (1) and (2), respectively.

- **D-lock**: The operation \((D(x), L_j)\) must acquire D-locks on the destination nodes \(x \in N_d(L_j)\). D-locks are upgraded from P-locks. When deleting a node, all of its child nodes are also deleted. As a result, D-locks (for the Delete operations) are incompatible with other types of locks by the conflict conditions (1) to (4) except the P-locks, which satisfy the non-conflict condition (3).

**4.2. XLP protocol**

The following six types of rules define XLP.

- **Two-phase Locking Rule**: All lock modes, except P-locks, that are acquired or released must observe the two-phase locking protocol (2PL).

- **P-lock Rule**: Nodes in the \(M\)-set of \(S_{i,j}\) are all locked by P-locks before performing the Node-Test and Predicate of location step \(S_{i,j}\).

- **Granularity Rules**
  
  (1) Lock granularity of P-, R-, I-, or W-locks on a node is only the node itself.
  
  (2) Lock granularity of D-locks on a node includes the whole subtree rooted at the node.

- **Upgrade Rules**
  
  (1) The P-locks on \(N_d(L_j)\) are upgraded to I-locks before inserting nodes into \(N_d(L_j)\).
  
  (2) The P-locks on \(N_d(L_j)\) are upgraded to R- or W-locks before reading or writing.
  
  (3) The P-lock on a node in \(N_d(L_j)\) is upgraded to D-lock before deleting the node only if P-locks on all the nodes in its subtree are acquired; that is, the Granularity Rule (2) is satisfied.

- **Compatibility Rule**: A particular type of lock on location step \(S_i\) can be granted as long as the compatibility matrix in Table 2 is respected.

- **Release Rules**
  
  (1) R-, W-, I- or D-locks on \(N_d(L_j)\) (i.e. \(R(S_{i,j})\)) can only be released in the shrinking phase of a transaction; that is, releasing them must observe the two-phase locking rule.
  
  (2) P-locks on nodes in the set \(\{x | x \in R(S_{i,j}) \cup R(S_{i,j}) \cap x \in R(S_{i,j}) \}\) for location path \(L_j\) are released only in the shrinking phase; that is, releasing P-locks on these nodes must observe the Two-phase Locking Rule.
  
  (3) P-locks on \((M(S_{i,j}) - R(S_{i,j})) \cup (M(S_{i,j}) - R(S_{i,j}))\) are released after location step \(S_{i,j}\) finishes.

The XLP protocol works as follows. When a location step \(S_{i,j}\) of location path \(L_j\) is executed in a transaction, nodes in the \(M\)-set of \(S_{i,j}\) are locked by P-locks by the P-lock Rule. For those nodes sieved out by the Predicate (i.e. in the set \((M(S_{i,j}) - R(S_{i,j})) \cup (M(S_{i,j}) - R(S_{i,j}))\)), their P-locks can be released immediately after \(S_{i,j}\) by Release Rule (3). P-locks on the non-sieved out nodes in the set \(\{x | x \in R(S_{i,j}) \cup R(S_{i,j}) \cap x \in R(S_{i,j}) \}\) for location path \(L_j\) are held till the shrinking phase of the transaction by Release Rule (2). The P-locks on the rest of non-sieved out nodes, i.e. \(N_d(L_j)\) in the final location step, are upgraded to R-, W-, I- or D-locks according to the operation type on this path by Upgrade Rules (1), (2) and (3). Acquiring and upgrading these locks must observe the Compatibility Rule and Granularity Rule, and releasing these locks must follow Release rules

**Table 2**: Lock compatibility matrix of XLP

<table>
<thead>
<tr>
<th>lock being requested on x</th>
<th>Lock held on node x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td>P</td>
<td>□</td>
</tr>
<tr>
<td>R</td>
<td>□</td>
</tr>
<tr>
<td>W</td>
<td>□</td>
</tr>
<tr>
<td>I</td>
<td>□</td>
</tr>
<tr>
<td>D</td>
<td>□</td>
</tr>
</tbody>
</table>
We shall prove any schedule under XLP satisfies the
conflict serializability [8,12,20,22], by showing that none
of the R-, W-, I- and D-locks generate cycles in
the precedence graph [8,12,20,22] (Lemma 1), nor does
the P-lock (Lemma 4). Further, we use the precedence
operator ‘<’ to denote the execution order of two
operations. For example, \( O_i(x) < O_j(y) \) denotes that operation
\( O_i(x) \) on node \( x \) (in transaction \( T_i \)) is executed before
operation \( O_j(y) \) on node \( y \) (in transaction \( T_j \)).

According to [22], we have Proposition 1.

Proposition 1. The two-phase locking protocol ensures serializability. The two-phase locking protocol with lock conversions also ensures serializability if upgrading locks
takes place only in the growing phase and downgrading
locks takes place only in the shrinking phase.

Lemma 1. The R-, W-, I- and D-locks under XLP generate
no cycle in the precedence graph.

Proof. According to XLP, the R-, W-, I- and D-locks
observe the 2PL protocol. XLP only has lock upgrade
but without lock downgrade, and the lock upgrade takes place
only in the growing phase by Upgrade Rules and Release
Rule (1). In sum, P-locks on the sieved out nodes in the
set \( (M(S_i) - R(S_i)) \cup (M(S_j) - R(S_j)) \) can be released
erlier, since they are surely not the context nodes of the
next location step. On the other hand, the P-locks on non-
sieved out nodes and the non-P-locks on \( N_e(L_i) \) are
released following the Two-phase Locking Rule, since
there exists a Read, Write, Insert or Delete operation on
these nodes or on their descendants. We use Example 2
to explain how transactions are executed concurrently
under the control of XLP.

Example 2. Suppose that there are two transactions \( T_3 \) and
\( T_4 \) which access the XML document in Fig 1. Also suppose
that \( T_3 \) starts before \( T_4 \). \( T_3 \) wants to write the last c i.e. \( c_4 \)
of \( b_3 \), and \( T_4 \) deletes \( b_3 \). \( T_3 \) and \( T_4 \) are represented as follows.
\( T_3 = \{ \text{Write}(x, L_3) \} \), where \( L_3 = \{ S_{1,3}, S_{2,3}, S_{3,3}, S_i = \text{child}:: a', S_{2,3} = \text{child}:: b[\text{position} = 3]\} \), and \( S_{3,3} = \text{child}:: c[\text{position} = \text{last}() \} \). \( T_4 = \{ \text{Delete}(x, L_4) \} \), where \( L_4 = \{ S_{1,4}, S_{2,4}, S_{1,4} = \text{child}:: a', S_{2,4} = \text{child}:: b[\text{position} = 3]\} \). Fig. 2
illustrates a possible schedule for \( T_3 \) and \( T_4 \) under XLP.
\( T_3 \) and \( T_4 \) cannot be executed concurrently only at location
steps \( S_{3,3} \) and \( S_{2,4} \) since there exists a P-lock on \( b_3 \) by \( S_{3,3} \)
and a D-lock on \( b_3 \) by \( S_{2,4} \). According to the compatibility
matrix in Table 2 and the Granularity Rules, P-locks and
D-locks are not compatible on node \( b_3 \). The deletion can
thus not be executed until \( T_3 \) finishes and releases its W-lock
on \( c_4 \) and its P-locks on \( b_3 \) and \( a \). Note that \( S_{2,4}:\text{Upgrade-}
D(\{b_3\}) \) must be executed after \( S_{2,3}:\text{Unlock}(\{b_3\}) \), as they are
conflicting operations.
executed, there is no way for the next $Z_i(x)$ in the sequence to acquire a P-lock in the shrinking phase of $T_i$. Similarly, the second sequence $O_j(x) < ... < Z_i(x) < ... < O_j(x)$ does not exist because, once the lock for $Z_i(x)$ is acquired and $T_j$ enters its shrinking phase, there is no way for the next $O_j(x)$ in the sequence to acquire the required lock for execution. The lemma is thus proved.

**Lemma 4.** The P-locks under XLP generate no cycle in the precedence graph.

**Proof.** Without loss of generality, consider a schedule $H$ under XLP consisting of two transactions $T_i$ and $T_j$. Assume there exists a cycle in the precedence graph of $H$. We shall prove this lemma by showing that the above assumption leads to a contradiction.

Consider the five types of lock modes in XLP. According to Lemma 1, the R-, W-, I- and D-locks generate no cycle; therefore the cycle must result from P-locks conflicting with the other four types of locks. Moreover, since P-locks and R-locks are compatible, the cycle must result from P-locks conflicting with the W-, I- and D-locks. Suppose $Z_i(x)$ is a Pass-by operation in $T_i$ and $O_j(x)$ is a Write, Insert or Delete operation in $T_j$. The existence of a cycle implies that there exists a conflict on some node $x$ between operations $Z_i(x)$ and $O_j(x)$ such that either of the two sequences $Z_i(x) < ... < O_j(x) < ... < Z_i(x)$ or $O_j(x) < ... < Z_i(x) < ... < O_j(x)$ exists.

There are two cases of the node $x$: one is $x \in M(S)\sim R(S)$ or $x \in M(S)\sim R(S)$ for some location step $S$ in location path $L$ in $T_i$; the other is $x \in M(S)\sim R(S)$ or $x \in M(S)\sim R(S)$ in $T_j$. For the first case, $Z_i(x)$ does not conflict with any operation in both sequences owing to Lemma 2, and none of both sequences exist. For the second case, by Lemma 3, none of the sequences $Z_i(x) < ... < O_j(x) < ... < Z_i(x)$ or $O_j(x) < ... < Z_i(x) < ... < O_j(x)$ exist. Therefore, there exists no cycle in the precedence graph due to P-locks, and we prove the lemma.

**Theorem 1.** XLP ensures conflict serializability.

**Proof.** By Lemma 1 and Lemma 4, all lock modes under XLP generate no cycle in the precedence graph. Therefore, the theorem is proved.

Note that, although XLP ensures conflict serializability, the phantom phenomenon may exist in some schedules under XLP, just like that in the 2PL protocol [8,20,22]. For example, consider two operations $O_1$ and $O_2$ in transactions $T_1$ and $T_2$, respectively and $O_1 < O_2$, where $O_1$ reads nodes satisfying $L_1 = \{a|b|position() = 2\}c/df$ in Fig. 1 and $O_2$ inserts a node $d_i$ into the nodes satisfying $L_2 = \{a|b|position() = 2\}c$ in Fig. 1. In the path from the root to $c_1$ via $a_1$ and $b_2$, both $O_1$ and $O_2$ must obtain P-locks on the root, $a_1$, $b_2$ and $c_1$ under XLP. Moreover, $O_2$ must upgrade the P-lock on $c_1$ to I-lock for inserting $d_i$ into $c_1$. However, the phantom phenomenon appears in the following execution order. Suppose that $O_1$, after obtaining the P-lock on $a_1$, is suspended for execution due to some reason (such as CPU scheduling by the operating system), and then $O_2$ is executed. Without interruption, $O_2$ successfully obtains P-locks on the root, $a_1$, $b_2$, and $c_1$, then upgrades the P-lock on $c_1$ to an I-lock and inserts node $d_i$ into $c_1$. Finally $O_2$ commits and releases all the locks it holds. The phantom (i.e. $d_i$) appears after $O_1$ resumes its execution, because the node $d_i$ just inserted by $O_2$ also satisfies $L_1$ at this time and now it can be seen by $O_1$. In fact, the phantom phenomenon can be avoided by using intention locks or larger lock granularity in XLP. Both approaches, however, may reduce concurrency largely. Therefore, XLP is designed only to ensure conflict serializability but allow the phantom phenomenon.

5.2. Implementation issues

We discuss the implementation issues in this subsection, including the implementation of lock mechanisms and the deadlock problem.

An efficient locking mechanism is necessary for implementing XLP since XLP is a lock-based protocol. [8,12,22] suggested several efficient mechanisms for lock-based protocols to store and to access locks in constant time. Among those discussed mechanisms, the hash table provides an efficient structure for implementing locks in XLP. The cost to check lock conflicts in the hash table is constant time, since only one look-up operation in the hash table is required.

Another important issue is the deadlock problem. In general, lock-based protocols may result in deadlocks. It is no exception of our XLP and other methods [4,5,13], although they [4,5,13] do not mention the deadlock problem. If the deadlock happens under XLP, a traditional deadlock detection and recovery algorithm [22] can be used. However, a graph for detection should be maintained in the system, which costs system resources. A more suitable method is to design a deadlock-free protocol, which avoids deadlocks at the lock dispatching time. This idea is part of our future work.

5.3. Concurrency comparison

In this subsection, we compare XLP with other protocols, including the 2PL [7,12,22], tree locking [21] and XPath-based protocols in [2,4,5]. Also, the protocol [13] (which is based on DOM) is compared in the aspect of its partial XPath support. Other protocols based on DOM [14] or DataGuides [10] are not discussed here, since their access models to XML trees are different from the XPath model.

5.3.1. Compared with 2PL

2PL [7,12,22] was originally developed and used in relational database systems. To the best of our knowledge, none of the current XML database systems use 2PL for their concurrency control scheme. However, if the concept and protocol of 2PL are used in XML database systems, XLP is assured to outperform 2PL [7,12,22], owing to the earlier
release of P-locks and lower lock conflicts according to its lock compatibility matrix. Suppose that 2PL is applied to the database system with XPath-like queries. Table 3 shows the lock compatibility matrix of 2PL. For an XPath expression, shared-locks are requested for nodes in each location step. However, due to the nature of 2PL, the shared-locks cannot be released earlier even if they are no longer needed. In contrast, although the Two-Phase Locking Rule in XLP requires R-, W-, I- and D-locks to follow the two-phase protocol, P-locks may be released as early as possible if Release Rule (3) in XLP is satisfied. XLP thus allows more concurrency than 2PL.

Moreover, a comparison between Tables 2 and 3 shows that XLP may generate more concurrency than 2PL due to fewer cases of lock conflicts. The shared-shared entry in Table 3 corresponds to the P-P, P-R, R-P and R-R entries in Table 2. The two protocols have equal concurrency for these entries. For the shared-exclusive entry in Table 3, the corresponding entries in Table 2 are P-W, P-I, P-D, W-W, W-I, I-I, I-D, D-W, D-I and D-D entries. For these entries, XLP may have higher concurrency than 2PL, since under XLP, transactions can be executed concurrently for the R-I entry and may have a chance to be executed concurrently for the P-W, P-I and P-D entries. Similarly it is also true of the exclusive-exclusive entry in Table 3. For the exclusive-exclusive entry in Table 3, the corresponding entries in Table 2 are W-W, W-I, W-D, I-W, I-I, I-D, D-W, D-I and D-D entries. Among them, transactions scheduled under XLP can be executed concurrently for both of the W-I and I-W entries.

5.3.2. Compared with the tree locking protocol

Two reasons assure that XLP is more suitable for XML documents than the tree locking protocol [21]. First, the ‘navigating-up’ axis (e.g. the ancestor axis) allows revisiting an XML node. But, the tree locking protocol does not support such access behavior. Once the locks on descendant nodes are acquired and those on ancestor nodes are released in the tree locking protocol, it is no longer possible to reacquire locks on the ancestor nodes hereafter. As a result, the locking behavior may make the tree locking protocol inapplicable to XML database systems. In contrast, XLP fully supports most XPath’s axis operations (excluding namespace axes). And second, XLP supports different lock modes (including shared and exclusive locks), while the tree locking protocol supports only the exclusive lock. Therefore, XLP may provide more concurrency than the tree locking protocol.

However, the tree locking protocol may outperform XLP if all of the following three conditions hold. (1) One location path \( L_i \) (executed by some transaction \( T_j \)) is a subset of another location path \( L_j \) (executed by another transaction \( T_j \)). (2) \( T_j \) accesses nodes under \( L_i \)’s destination nodes, and, (3) the locks requested by \( T_i \) and \( T_j \) are incompatible on \( L_i \)’s destination nodes (according to the compatibility matrix in Table 2). In this case, since the locks on the destination nodes of \( L_i \) may block \( T_j \), XLP may restrict concurrency. In contrast, under the tree locking protocol, \( T_j \) may access the destination nodes of \( L_i \) without being blocked because \( T_j \) can release locks on these nodes at any time if they are no longer needed. Even so, the probability of the case is not high because most XML accesses are read operations and the chance that both conditions (1) and (2) hold is low.

5.3.3. Compared with other XPath-based protocols

This subsection compares XLP with the concurrency control protocols in [2,4,5] and the lock procedure in [13]. The comparison is made from various perspectives, including supports of the axes and Predicate selection, consistency, concurrency and performance (cost of conflict check).

XLP supports twelve XPath axes (except the namespace axis) and the Predicate selection in the XPath expression. On the other hand, [2] supports simple path expressions (i.e. path expressions with child or descendant axes) and the Predicate. Protocols in [4,5] also support simple path expressions, but they do not support the Predicate. Based on the DOM model, [13] only supports three XPath’s axis operations, including parent, child and sibling axes, but it does not support the Predicate selection.

As to the consistency issue, [2,4,5] and XLP all ensure serializability, and query access methods in [13] accomplish only the repeatable read property. On the other hand, the phantom phenomenon is avoided in [2,4,5]. Ref. [2] resolves the problem by checking the conflict of a transaction’s state with others’ at the commit time. The PL-PROP and PL-SAT schemes in [4,5] ensure phantom-free schedules by adopting a larger lock granularity in the lock compatibility check. In contrast, although both [13] and XLP can avoid phantoms by enlarging the lock granularity, [13] and XLP leave the problem as an open question for the reason of better concurrency degree.

More lock modes supported in a protocol may result in lower chances of lock conflicts and an increase in concurrency. Ref. [2,4,5] use read and write locks only, while [13] uses ten lock modes, including locks on nodes and locks on edges between nodes. In contrast, XLP provides P-, R-, W-, I- and D-locks. If P- and R-locks are considered as shared locks, and W-, I-, D-locks as exclusive locks, then XLP may generate more concurrency than [13] in both the shared-exclusive and exclusive-shared entries in the compatibility matrix. But Ref. [13] may generate more concurrency than XLP in the exclusive-exclusive entries owing to its richer exclusive lock modes.

As to the lock duration, the protocols in [2,4,5,13] hold locks on the nodes related to a location path until the transaction finishes. In contrast, XLP may release P-locks

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Lock compatibility matrix of 2PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(s) being held</td>
<td>Lock being requested</td>
</tr>
<tr>
<td>Shared (P, R)</td>
<td>Exclusive (W, I, D)</td>
</tr>
<tr>
<td>○</td>
<td>×</td>
</tr>
</tbody>
</table>

on some nodes as early as a location step finishes. Further, since XLP follows 2PL, all locks (including non-released P-locks) are released in the shrinking phase before transaction termination. Therefore, XLP has shorter lock duration than other protocols, leading to lower chances of lock conflicts and more concurrency.

As to the cost of conflict check, the concurrency control schemes in [2,4,5] involve a complex conflict checking process. [2] verifies transactions’ local database states to check conflict between transactions. It is time consuming, even though a sufficient condition for reducing the checking overhead (namely R-W check in [2]) is proposed. The conflict check of the PL-SAT in [4,5] is costly, since the process includes checking the ancestor/descendant relationships between two nodes and checking if the two nodes exist in the same location path. The PL-PROP scheme in [4,5] performs the conflict checking in constant time, but more space is required for storing the propagated read locks. In contrast, XLP has a low cost of conflict check because it explicitly specifies the nodes to be locked and directly checks the node conflict by the lock compatibility matrix. If the hash table is used to implement the lock table, only one look-up operation in the hash table is required. As a result, the cost of conflict check in XLP is constant time, so is the conflict check in [13] (if the hash table is used).

6. Conclusions and future work

Providing a high degree of concurrency in XML database systems is crucial in many applications. In this paper, a new concurrency control protocol XLP is proposed for accessing XML documents. The proposed XLP is based on the XPath model and ensures conflict serializability. This paper has three major contributions. First, the proposed XLP supports most XPath axes, rather than simple path expressions (as in other protocols). Second, we derive conflict conditions for the XPath model, which form a basis of the study and design of (new) protocols for database researchers. Third, the paper proposes a new lock mode, P-lock, which is a lightweighted lock (even less restrictive than a shared lock) and can be released early. Based on P-locks, we also devise a serializable and efficient locking protocol XLP.

Our future work includes developing deadlock-free protocols for the XPath model since deadlocks may happen under XLP. A database recovery mechanism is also one of the next steps in our plan. Finally, we intend to construct a complete transaction system for XLP.

References

[23] http://www.w3.org/DOM/