Concurrency in XML

Daniel Butnaru

20. Juni 2006
1 Introduction

2 XTC
   • System Architecture

3 Storage Model
   • taDOM

4 Concurrency
   • Node Locks
   • Node Lock Conversion
   • Tunable Node Lock Granularity and Lock Escalation
   • Navigational Locks
   • Prevention of Phantoms

5 References
Concurrency occurs when two or more execution flows are able to run simultaneously.

Concurrency control is a method used to ensure that database transactions are executed in a safe manner.

- **Atomicity** refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them are.
- **Consistency** refers to the database being in a legal state when the transaction begins and when it ends.
- **Isolation** refers to the ability of the application to make operations in a transaction which appear isolated from all other operations.
- **Durability** refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone.
The following is a presentation of the work conducted by Michael P. Haustein and Theo Härder on XML concurrency.
Introduction

Classical approach
Classic concurrency control mechanisms developed for relational, hierarchical, and object oriented databases are no longer adequate for semi-structured databases.

Processing XML documents in multi-user database management environments requires
- a suitable storage model of XML data
- support of typical XML document processing interfaces (DOM, SAX)
- concurrency mechanisms specific to XML
XML Transaction Coordinator

- (O)RDBMS-connectable DBMS for XML documents (XDBMS)
- it adheres to the widely used five-layer DBMS architecture
XTC Architecture Overview
System Architecture

- The file-services layer operates on the bit pattern stored on external non-volatile storage devices.
- A buffer-manager handles fixing and unfixes of pages in memory and a replacement algorithm.
- The record manager maintains in a set of pages the tree connected nodes of XML documents.
- Each record is addressed by a unique ID managed within a B-tree by the index manager.
- The catalog manager provides for the database metadata.
- The node manager implementing the navigational access layer transforms the records from their internal physical representation into an external representation.
- XML-services layer contains the XML manager responsible for declarative document access.
- The interface layer makes the functionality of the XML and node services available to common clients.
Efficient and effective synchronization of concurrent XDP can be helped by an adequate internal representation. To this purpose two more node types are introduced:

- **attributeRoot** which separates the attribute nodes from their parents
- **string** attached to a text or attribute node
taDOM Tree

- Document Root
- Element Node
- Attribute Root
- Attribute Node
- Text Node
- String Node

- bib
  - book
    - title
      - year: 1999
      - id: 2
    - editor
      - last: The E...
      - first: Gerbarg
      - affiliation: Citi
    - price: 129.95
  - book
    - title
      - year: 2000
      - id: 1
    - author
      - last: Abiteb...
      - first: Serge
    - author
      - last: Bunem...
      - first: Peter
    - author
      - last: Suciu
      - first: Dan
    - price: 39.95
Node Locks

NR
An NR lock mode (node read) is requested for reading the context node. An NR lock has to be acquired for each node in the ancestor path. It only locks the specified node, but not any descendant nodes.

IX
An IX lock mode (intention exclusive) indicates the intent to perform write operations somewhere in the subtree, but not on a direct-child node of the node being locked.

LR
An LR lock mode (level read) locks the context node together with its direct-child nodes for shared access.
Node Locks

SR

An SR lock mode (subtree read) is requested for the context node c as the root of subtree s to perform read operations on all nodes belonging to s. Hence, the entire subtree is granted for shared access.

CX

A CX lock mode (child exclusive) on context node c indicates the existence of an X lock on some direct-child node and prohibits inconsistent locking states by preventing LR and SR lock modes.

U

A U lock mode (update option) supports a read operation on context node c with the option to convert the mode for subsequent write access.
To modify the context node $c$ (updating its contents or deleting $c$ and its entire subtree), an X lock mode (exclusive) is needed for $c$. It implies a CX lock for its parent node and an IX lock for all other ancestors up to the document root.
Node Locking for a taDOM tree

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>NR</th>
<th>IX</th>
<th>LR</th>
<th>SR</th>
<th>CX</th>
<th>U</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>NR</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IX</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LR</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SR</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CX</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>U</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>X</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Compatibility Matrix

Locking example
Node Locks

Example

- T1 wants to modify the value Darcy
- T2 wants to delete the entire editor node
- T3 is generating a list of all book titles
Node Lock Conversion

- a transaction can hold multiple locks
- a node can be locked by multiple locks
- long list of granted locks
- an algorithm for checking lock compatibility

**Idea**

Replace all locks per node with a single lock in a mode giving sufficient isolation.
### Lock Conversion Matrix

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>NR</th>
<th>IX</th>
<th>LR</th>
<th>SR</th>
<th>CX</th>
<th>U</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>NR</td>
<td>NR</td>
<td>-</td>
<td>IX</td>
<td>LR</td>
<td>SR</td>
<td>CX</td>
<td>NR</td>
<td>X</td>
</tr>
<tr>
<td>IX</td>
<td>IX</td>
<td>IX</td>
<td>-</td>
<td>IXNR</td>
<td>IXSR</td>
<td>CX</td>
<td>IX</td>
<td>X</td>
</tr>
<tr>
<td>LR</td>
<td>LR</td>
<td>LR</td>
<td>IXNR</td>
<td>-</td>
<td>SR</td>
<td>CXSR</td>
<td>LR</td>
<td>X</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
<td>SR</td>
<td>IXSR</td>
<td>SR</td>
<td>-</td>
<td>CXSR</td>
<td>SR</td>
<td>X</td>
</tr>
<tr>
<td>CX</td>
<td>CX</td>
<td>CX</td>
<td>CX</td>
<td>CXNR</td>
<td>CXSR</td>
<td>-</td>
<td>CX</td>
<td>X</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

A lock \( l_1 \) with a subscript \( l_2 \) means that \( l_1 \) has to be acquired on \( c \) and \( l_2 \) on all its direct children.
Tunable Node Lock Granularity and Lock Escalation

Flexibility

Improvements in efficiency and potential parallelism can be achieved by enabling tunable lock granularity and lock escalation:

- on one side we want to reduce the number of locks for less complexity and higher throughput
- on the other side we want higher concurrency through fine-tuned lock granularity

Granularity

The parameter lock depth \( ld \geq 0 \) describes the lock granularity by means of the number of node leaves (starting at root) on which nodes are to be held. If a lock is requested for context node \( c \) whose path length to the document root element is greater than \( ld \), only an SR lock for the ancestor node belonging to the lock-depth level is requested.
Tunable Node Lock Granularity and Lock Escalation

Escalation

Escalation can be achieved by two new parameters:

- **escalation threshold (et)**. If the number of locked nodes of a transaction on a subtree exceeds et, the held locks are replaced by an adequate lock at the subtree root.

- **escalation depth (ed)**. Defines the maximal subtree depth starting from the leaves of a taDOM tree up to the scanned subtree root.
Coarse-grained node locks with depth 2

Before

After
Navigational Locks

- so far nodes accessed by unique ID
- DOM provides for (20) methods for traversing XML trees using navigational paths.

**PathSynchronization**

Synchronizing a navigation path means that a sequence of navigational method calls or modification operations starting at a known node within the taDOM tree must always yield the same sequence of result nodes within a transaction.
Locking navigational operations in a taDOM tree

**Compatibility Matrix**

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>ER</th>
<th>EU</th>
<th>EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EU</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EX</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Virtual navigation edges on an element-node

prevSiblingEdge

nextSiblingEdge

firstChildEdge

lastChildEdge
Path Locks

ER
An ER lock mode (edge read) is needed for an edge traversal in read mode, e.g., by calling the getNextSibling() or getFirstChild() DOM method for the nextSiblingEdge or firstChildEdge, respectively.

EX
An EX lock mode (edge exclusive) enables an edge to be modified which may be needed when nodes are deleted or inserted. For all edges, affected by the modification operation, EX locks are acquired, before the navigation edges are redirected to their new target nodes.

EU
The EU lock mode (edge update) eases the starvation problem of write transactions.
Use of navigation locks
Prevention of Phantoms

Cause

Phantom reads occur when an insert or delete action is performed on a node that belongs to a range of nodes being read by another transaction. The transaction’s first read of the range of nodes shows a node that no longer exists in the second or succeeding read, as a result of a deletion by a different transaction.
Prevention of Phantoms

Problem
- so far locking works for record oriented direct or navigational access
- what about set oriented access (DOM: `getNodesByName()`)?

Possible solution
We acquire an exclusive lock one level above the working node, that is, on its direct ancestor, and prevent the transaction from being confused by phantom inserts.

Trade-off
Increases blocking and deadlock probability. Left for future research.
References

Questions?