Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp486-490)

A Transaction Mechanism for Native XML Database

HUI TAN, XINMENG CHEN
College of Computer
Wuhan University
Wuhan 430072, Hubei
CHINA
journal@whu.edu.cn

JINGUANG GU
College of Computer Science and Technology
Wuhan University of Science and Technology
Wuhan 430081, Hubei
CHINA
simon@wust.edu.cn

Abstract: Concurrent accessing of XML data becomes more and more important for XML-based applications, especially for Native XML Database systems (NXD) because of the dramatic growth of number and complexity of such applications. Transaction is a potential solution for the concurent accessing of XML data. However, research on transaction of NXD is still very limited. This paper proposes a transaction model for NXD management systems. It introduces a NXD storage model which is suitable for accessing XML data using XPath and XQuery. A locking protocol, Schema path Lock Protocol, is designed for this transaction model. An implementation of such transaction model is presented last in this paper.

Key–Words: Distributed Transaction Process, Multi-granularity Lock, Native XML Database, XQuery

1 Introduction

Integration on XML-based information is an important topic for web-based information systems. Due to its ability to express semi-structured information, XML is rapidly becoming a language of choice to express and store information on the web. XML documents could be stored in text files, relational or object-oriented database (XML Enabled Database, XED) or native XML database (NXD). Therefore, all features presented in relational and object-oriented databases can be revisited in the context of XML documents. Concurrency control mechanism allows multiple users to query and update a document simultaneously. Concurrency control has been extensively studied in the context of traditional database management system such as RDBMSs and OODBMSs. Some important concepts such as locks, transactions, schedulers are introduces as well as protocols such as two phase locking (2PL) for serializability. Instead, the research on transaction in NXD is still very limited. Transaction mechanisms for rational and object-oriented database could be used for XML documents transaction in XED. One way to provide concurrent access to an XML document is to store the document in relational database and using the existing locking schemes in RDBMS. However, such method often causes over-restrictive locks while guaranteeing serializability. Most NXD systems such as Tamino[9] and Xindice lock the entire XML document for concurrency control, which also cause lots of limitation. In order to ensure the serializability of transaction, locking must prevent the phantom problem. There are two well-known locking mechanisms for solving phantoms problem, physical index based locking and logical predicate based locking. The index based locking, such as key range locking, has been widely adopted for RDBMS. Unfortunately, index based locking is not applicable for XML data management for the absence of an efficient index structure for an XML document model. The predicate based locking is a better choice for native XML database, but it is expensive.

This paper will introduce a new transaction model to improve the concurrently accessing ability of NXD and solve the problem of phantoms. The remainder of this paper is structured as follows. Section 2 gives the general discussion about the storage and accessing properties of native XML database. Section 3 discusses the locking protocol of the transaction, a new fine-granularity locking algorithm named Schema Path Lock (SPLOCK) is introduced in this section. Section 4 presents an implementation of this transaction model. Section 5 summarizes the whole paper.

2 Storage and Accessing Properties of NXD

2.1 The Property of Storage of NXD

Efficient and effective synchronization of concurrent access to an XML document is greatly facilitated if a specialized internal representation is used, which en-

ables fine-granular locking. For this reason, it is suitable to discuss the property of storage model of native XML database firstly, though the storage model is not the key topic of this paper.

An XML document can be described as a tree structure, the root of the tree represents the root of the XML document, and the nodes of the tree represent different kinds of elements of the XML document, which are element nodes, attribute nodes or text nodes. Some storage models such as taDOM[8] use different kinds of shape to denote the different kinds of nodes when they represent the XML document in a graph, in our strategy, it need not to identify the difference of the nodes, because all the discussion of us are based on the precondition that the interface of the native XML database storage is XPath based language such as XQuery or XUpdate. XML document can be represented as a tree with tags on the nodes, name it as $T_d = (V_d, E_d, \delta_d, \Sigma_d, root_d, oid, type_d, oid_d)$, and V_d is a node set of the tree; E_d is an edge set of the tree, edge represents the relationship between element nodes; δ_d is a mapping function of nodes. Every node has a type which corresponds with an unique identifier, and these types and identifiers belong to set Σ_d and oid respectively, and corresponding mapping functions are $type_d$ and oid_d. Each XML data tree has a root node $root_d$[11].

Most native XML databases use DOM or OEM storage model to store the XML documents because these models can improve the processing performance when using DOM API to accessing the XML documents. However, it is a better choice to use XPath/XQuery or XUpdate interface to access the XML documents under the web-based information integration environment. Like the schema of RDBMSs or OODBMSs, the schema of XML documents can improve the performance of querying or updating the data in the NXDs, so storing the schema information with the XML documents is a better choice. DataGuide[5] of Lore[1] is the referenced implementations of the schema based storage model. The data schema of XML document can be defined as a directed graph, name it as $G_t = (V_t, E_t, \delta_t, \Sigma_t, root_t)$. And V_t is set of nodes; E_t is the edges set between nodes, edge denotes the nested inclusion of nodes; function δ_t determines the direction of each edge in E_t; Σ_t is the name set of the nodes, that is type set of elements; the unique node in graph which has no input edge but a output edge is called the root of graph, named it as $root_t$. In this model, children nodes can access corresponding parent nodes by means of function ${\delta_t}^{-1}$[11].

2.2 The Property of Accessing Interface of NXD

XML document can be extracted into a set of several path instances, thus each data node corresponds with a path instance respectively. Path expression query is utilized to match path instance of XML document, so as to acquire sound element instance.

Path query can be represented by $Q =< G_t, T_d, SE, PE, RS >$, G_t and T_d is query document schema and document data respectively, PE is query path expression, SE is the initial elements set of query, RS is the query result set. Since SE is the original point of PE, thus PE is a relative path absolutely, moreover, all path connection identifiers in PE just represent the relationship among elements. The original query form of the user may be acquired by $SE + PE$.

This paper use a concept named XML extension to express how to get record set of the query.

Definition 1 *For a XML document T_d, define XML extension $EXT(Etype)$ as a set of element instances in T_d whose type is Etype.* $EXT(Etype) = \{oid_d(e) \mid e \in V_d \wedge type_d \in \Sigma_d \wedge type_d = Etype\}$

Theorem 2 *If absolute path expression query $Q =< G_t, T_d, root_t, PE, RS >$ is an unique path which can access final goal in G_t, and it use $End(PE)$ to express the final element type of path PE, thus $RS = EXT(End(PE))$.*

PE is XPath-like query expression, which can be described as,

$PE ::= CF$

$F ::= label|text()|@*|*|FCF|F[P]$

$C ::= /|//$

Where P is the set of predicate condition expression to limit the quantity of the nodes affected by the query, it can be described as,

$P := F|F\ O\ const|P\ and\ P$

$O :=< | \leq | > | \geq | = | \neq$

For a specified query expression PE_n, it can be divided into tow parts, one is the pure schema path, it is the path of the schema graph without any predicate condition, the other is a set of predicate expression. The pure schema path is named as SP, it use unique labels to mark every node in the pure schema path, named them as $SL = \{sl_1, sl_2, \ldots, sl_n\}$, SP can be expressed as:

$SP ::= CF$

$F ::= label|text()|@*|*|FCF$

$C ::= /|//$

The set of the predicate condition P can be expressed as $\{(SL = sl_1) \wedge P_{sl_1}, (SL = sl_2) \wedge$

$P_{sl_2}, \ldots, (SL = sl_n) \wedge P_{sl_n}\}$. Where sl_i $(1 \leq i \leq n)$ is the unique label of a node of the pure schema path, P_{sl_i} is the predicate condition of the node.

3 Lock Protocol for Native XML Database

3.1 Insufficiency of existing lock mechanisms

How to add locks to the XML document is the first problem it need to resolve. The following lock mechanisms, which may be applied to XML documents transactions, will be examined.

Table Locking or Document Level Locking. In XED systems, the XML documents may be stored in a parent-child relationship table, In this approach the entire table representing the hierarchy will be locked in case of query that traverses the hierarchy. If this were not the case, a phantom could occur. A lock on the entire parent-child relationship table makes it impossible, however, to add an element in a subtree that has not been read by any user. In NXD systems such as Tamino, Xindice, the entire document affected by the query will be locked, it is insufficient to the concurrency of the XML documents.

Predicate Locking. Predicate locks[4] has been introduced to fix the problem mentioned above. There is no longer any need to lock an entire table; phantoms cannot occur because predicates are given that describe the tuples that have been selected in an INSERT, UPDATE or DELETE query. however. they are rarely implemented in commercial relational databases systems because they are prohibitively expensive. It's a better choice to combine the predicate lock and other kind of locks to get a fine-granularity lock mechanism and a high performance to process the XML documents.

Tree Locking. Tree locking [2] protocols are used in multi-user operating systems to allow concurrent access to the directory structure like XML documents. In these protocols locks do not hold for entire granules but only for nodes, i.e., when a node is locked its descendants are not also locked. However, the operation to add and remove a lock on a node of the XML document is expensive too.

Path Locking. Path lock scheme[3] such as Path Lock Satisfiability (PLS) and Path Lock Propagation (PLP) means to add read locks or write locks on the paths affected by the query. Current path lock scheme is to add read or write lock to the instance graph of the XML document. Though the path lock can get a multi-granularity lock mechanism for XML documents, but too many locks have to be added to the path affected by the query, especially by the XPath or XQuery based query.

Table 1: Lock compatibility. Compatibilities marked as '+', the uncertain marked as 'P', which will be tested by the predicate condition.

	Granted					
Requested	IS	IX	S	SIX	U	X
IS	+	+	+	+	P	P
IX	+	+	P	P	P	P
S	+	P	+	P	P	P
SIX	+	P	P	P	P	P
U	P	P	+	P	P	P
X	P	P	P	P	P	P

3.2 A multi-granularity lock for the transaction of NXD

Path Lock mechanism is a better choice to get a multi-granularity lock mechanism for Native XML Documents, especially for the situation that the interface of Native XML database system is XPath or XQuery. However, to reduce the lock quantity, it needs not to add the lock to the instant graph of the XML documents, add lock to the schema graph instead. This kind of lock is named as Schema Path Lock (SPLOCK), it can be defined as:

Definition 3 *Define Schema Path Lock as* $L_n = (type_n, T_n, G_n, PE_n)$*,* $type_n$ *is the type of the lock, it can be NONE lock, INTENT SHARE lock (IS), SHARE lock (S), EXCLUSIVE lock (X), UPDATE lock (U), INTENT EXCLUSIVE lock (IX) and SHARE INTENT EXCLUSIVE lock (SIX), the compatible matrix of these locks discussed in table 1 and paper [6];* T_n *is the identifier of the transaction who owns the lock,* G_n *is the identifier of the node in the XML schema graph which the lock holds;* PE_n *is the path expression of the transaction.*

SPLOCK is the mix of path lock and predicate lock, it adds path lock to the XML schema graph to reduce the quantity of lock of the system and it uses the predicate condition to restrict the scale of the lock will be affected.

The schema of the path lock can be one of the following:

- The schema information stored in the NXDs or dynamic generated from the XML documents.
- The background knowledge. The knowledge represents in Ontology language such as DAML+OIL or OWL[10], and maps it to XML DTD or Schema using suitable algorithm.

When a new lock request to be added to a node of G_t, how to identify the new lock is compatible with the locks which have been granted to the node. A new function named $Conf(T_1, T_2)$ is used to do this task, T_1 is the lock which requests to be added and T_2 is the lock has been granted, if $Conf(T_1, T_2) = `+'$ means lock $T_1$ is compatible with lock $T_2$, otherwise $Conf(T_1, T_2) = `-'$ and lock T_1 conflicts with lock T_2.

To make the transaction serializable, the following rule is true.

Theorem 4 *To a node of XML schema graph, it has n transactions named $T_1, T_2, \ldots, T_n$, and corresponding n locks named $L_1, L_2, \ldots, L_n$. All the locks are compatible with each other, or $T_1, T_2, \ldots, T_n$ are serializable. Now, suppose we want to add a new lock T_{n+1} to this node, if $T_1, T_2, \ldots, T_n, T_{n+1}$ are serializable, the following expression must be satisfied.*

1. $\forall i \forall j((1 \leq i \leq n \wedge 1 \leq j \leq n \wedge i \neq j \wedge (i < j)) \rightarrow (Conf(T_j, T_i) = `+') \vee RS_{PE_i} \cap RS_{PE_j} = \emptyset)$.
2. $\forall i(1 \leq i \leq n \rightarrow (Conf(T_{n+1}, T_i) = `+') \vee RS_{PE_i} \cap RS_{PE_{n+1}} = \emptyset)$.

Proof: first to prove item 2. If $Conf(T_{n+1}, T_n) = `+'$, this means the requesting lock $T_{n+1}$ is compatible with the granted lock $T_i$, the transaction is serializable obviously. If $Conf(T_{n+1}, T_n) = `-'$ and $RS_{PE_i} \cap RS_{PE_{n+1}} \neq \emptyset$, this means lock T_{n+1} and T_i will try to add lock to the same nodes of the XML documents, these will make the transaction unserializable.

Suppose adding lock in the sequence of $T_1, T_2, \ldots, T_n$, it can prove that item 1 is also satisfied. □

It can use theorem 4 to get the value of $Conf(T_i, T_j)$ when the compatibility of lock T_i and lock T_j is marked with 'P'. If $RS_{PE_i} \cap RS_{PE_j} = \emptyset$, $Conf(T_i, T_j) = `+'$, else $Conf(T_i, T_j) = `-'$.

From section 2.2 we know that a query expression PE can be divided into two parts, one is the pure schema path SP, the other is the predicate condition P, the query result RS will be chosen by the predicate condition P. In fact, the lock is added to G_t from the root to node (see section 3.3), we only need to process the predicate condition P_{sl_m}, while sl_m is the label of the node.

3.3 Scheduling Algorithm for Schema Path Lock

How to add locks and add what kind of locks to the Schema tree of the XML document is the problem must be discussed in this subsection. We fist introduce the concept used in the algorithm.

The algorithms to add locks to the schema tree G_t can be described as following:

- Adding locks from the root to the nodes of the XML schema.
- Freeing locks from the nodes to root of the XML schema.
- If add S or IS lock to a non-root node, a IS lock or higher level lock (IS, IX, S, SIX, U, X) must be added to its parent or key ancestor path.
- If add X, U, SIX or IX lock to a non-root node, a IX lock or higher level lock (IX, SIX, U, X) must be added to its parent or key ancestor path.
- when adding lock to a node fails, it means the new lock can not obey the theorem 4, the scheduler tries to add locks to the parent node or the key ancestor node based on the path expression of the query request.
- A lock conversion operation must be performed when the scheduler try to add one or more new kind lock to a schema node which has owns one lock. The scheduler will try to convert current lock to the highest level lock. described as figure 1.

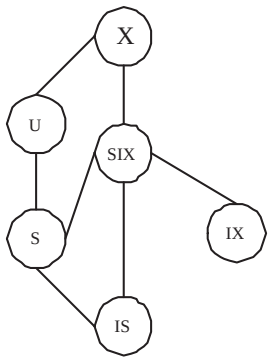

Figure 1: Lock Conversion Graph.

4 Implementation

As shown in figure 2, we implement a native XML transaction management system in our semi-structured information integration system named OBSA[7]. It can be divided into four layers logically,

- XML Data Storage Layer;
- XML Data Buffer Layer;
- XML Transaction Processing Layer;
- XML Transaction Management Layer.

The XML transaction processing layer implements the SPLOCK introduced in this paper, but not implements the algorithm of finding key ancestor. All the operations are processed in the memory before the transactions commit, the buffer manager update the data stored in physical storage layer when all the transactions commit.

5 Conclusion

A schema path locking protocol is implemented to support ACID[6] and concurrency accessing properties of native XML database, it adds locks on schema graph of the XML documents to reduce the quantity of locks and use predicate locking to prevent the phantom problem of transaction. With the analysis of the query expression, a more flexible schedule algorithm is introduced to support the conversion of the transaction locks.

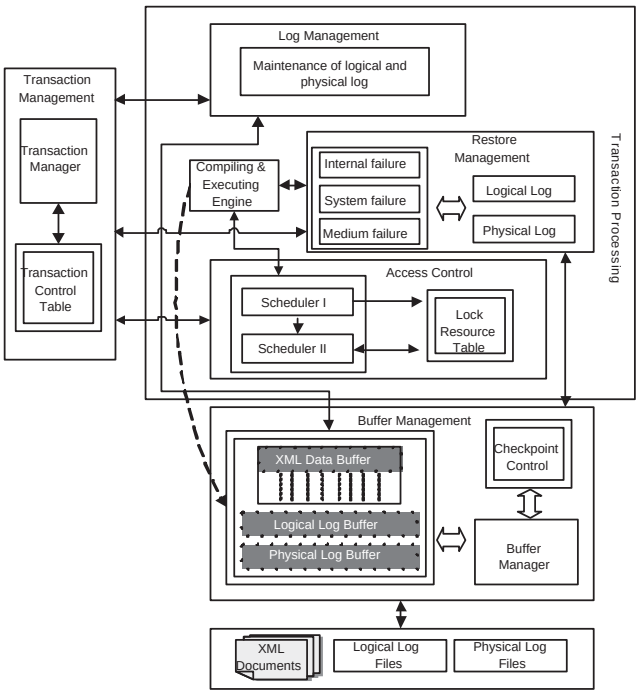

Figure 2: The Implementation of Transaction.

References:

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured data. *Journal on Digital Libraries*, 1(1):68–88.

[2] N. S. Barghouti and G. E. Kaiser. Concurrency control in advanced database applications. *ACM Computing Surveys*, 23(3):269–317, 1991.

[3] S. Dekeyser, J. Hidders, and J. Paredaens. A Transaction Model for XML databases. *World Wide Web: Internet and Web Information System*, 7(1):29–57, 2004.

[4] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, 1976.

[5] R. Goldman and J. Widom. DataGuides:Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 436–445. Morgan Kaufmann Publishers Inc, 1997.

[6] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[7] J. Gu, H. Chen, L. Yang, and L. Zhang. OBSA:Ontology-based Semantic Information Processing Architecture. In J. Liu and N. Cercone, editors, *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence 2004*, pages 607–610, Beijing, 2004. IEEE Computer Society Press.

[8] M. P. Haustein and T. Härder. taDOM:a Tailored Synchronization Concept with Tunable Lock Granularity for the DOM API. In L. A. Kalinichenko, R. Manthey, B. Thalheim, and U. Wloka, editors, *Proceedings of 7th East European Conference on Advances in Databases and Information Systems*, volume 2798 of *Lecture Notes in Computer Science*. Springer, 2003.

[9] H. Schöning. Tamino - a DBMS designed for XML. In *Proceedings of 7th International Conference on Data Engineering*. IEEE Computer Society Press, 2001.

[10] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[11] G. Wang and M. Liu. Query Processing and Optimization for Regular Path Expressions. In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003)*, Lecture Notes in Computer Science 2681, pages 30–45, Klagenfurt, Austria, 2003. Springer.