第35卷 第14期 计 算 机 工 程 2009年7月
Vol.35 No.14 Computer Engineering July 2009
·软件技术与数据库· 文章编号：1000—3428(2009)14—0035—03 文献标识码：A 中图分类号：TP311.52

Native XML 数据库并发控制协议

魏东平，孙华国，宗德君
(中国石油大学计算机科学系，东营 257061)

摘 要：并发控制是改善数据库系统事务性能的最重要机制，也是 Native XML 数据库研究领域的一个难点。通过简化 XPath 数据模型，定义基于该模型的几种 Native XML 数据库操作，提出一种新的基于 XPath 的加锁协议，分析该协议下的调度是可串行化调度，并针对其性能问题进行探讨。

关键词：Native XML 数据库；并发控制；可串行化

Concurrency Control Protocol for Native XML Database

WEI Dong-ping, SUN Hua-guo, ZONG De-jun
(Department of Computer Science, China University of Petroleum, Dongying 257061)

【Abstract】 Concurrency control is the most important mechanism of improving the transaction capability of database system. And it is also a difficulty in the field of Native XML database. Some kinds of Native XML database operations based on the simplified Xpath data model are proposed. On the basis of this, an Xpath-based locking protocol is put forward and the schedule within the locking protocol mentioned above is proved to be serializability. Some detailed discussion is given to analyze the capability of concurrency for the protocol.

【Key words】 Native XML database; concurrency control; serializability

并发控制是改善数据库系统事务性能的重要机制，它允许多个事务并发执行并保证事务的隔离性。但在 Native XML 数据库领域，并发控制机制的研究才刚刚开始，目前还没有成熟的技术以及统一的事务处理模型。

1 XPath 简介

XPath 是一种对 XML 文档的内容进行定位、检索的语言，是后续更强大的数据检索语言如 XQuery 的基础。XPath 将一个 XML 文档视为一棵节点树，在 XPath 树状模型中有 7 类节点，分别是根节点、元素节点、文本节点、属性节点、命名空间节点、处理指令节点和注释节点。由于本文讨论面向数据的 XML 文档，因此对后 3 种节点不作讨论。XPath 对每种节点都定义了计算其字符串值的方法，不同的节点计算方法可能不同。根节点有唯一的子节点，代表 XML 文档元素；元素节点有一个文本节点并可含有一组属性节点；文本节点和属性节点没有子节点。每个节点都有相应的字符值表示这些节点的内容。一个 XML 文档片段如下：

```
<Department>
  <Students>
    <Student student_id="08001">
      <Name>Wang Fang</Name>
      <Sex>Female</Sex>
      <Age>20</Age>
    </Student>
    <Student student_id="08002">
      <Name>Li Ming</Name>
      <Sex>Male</Sex>
      <Age>21</Age>
    </Student>
  </Students>
  <Courses>
    <Course course_id="C9001">
      <Name>Database Technology</Name>
      <Addr>4-4205</Addr>
    </Course>
  </Courses>
</Department>
```

2 Native XML 数据库事务模型

2.1 数据模型

为对单个 XML 文档实施多用户并发控制，将标准的 XPath 数据模型进行简化，引入 1 个 Xp-tree(XPath Tree)数据模型[1]。

定义 1(Xp-tree) 用一个 4 元组形式化描述 Xp-tree，称 *Xp-tree*=(N, B, r, V)。其中，N 表示树中节点集；B：$N\times N$ 是树中有向边的集合，表示元素节点之间的关系；r 表示树的根节点；映射函数 V 的作用是将除根节点外的所有节点映射到元素名(或属性值、文本值)的字符串。

在 Xp-tree 数据模型中，只划分为终节点和非终节点 2 种节点类型，而不对元素节点、属性节点和文本节点加以区分。终节点即为树中的叶子节点，由字符数据组成，且不能包含孩子节点。上文 XML 文档片段的 Xp-tree 数据模型见图 1。该 XML 文档片段的部分节点值如下：

$V(n_1)$=Department
$V(n_2)$=Students
$V(n_3)$=Courses
$V(n_4)$=Student
$V(n_5)$=Student
$V(n_6)$=Course

作者简介：魏东平(1965－)，男，副教授，主研方向：数据库理论及应用；孙华国、宗德君，硕士研究生

收稿日期：2008-12-03 **E-mail：**lzpx_2004@163.com

$V(n_7)$=student_id
$V(n_8)$=Name
$V(n_{18})$=“08001”

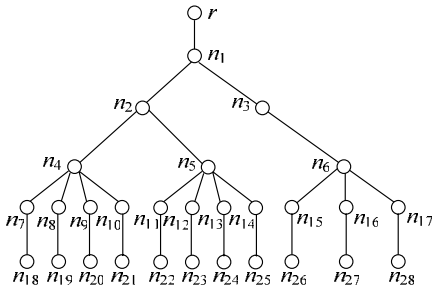

图 1　XML 文档片段的 Xp-tree 数据模型

2.2　操作模型

本文假定事务可以对 XML 文档实施 4 种基本数据操作：

(1)查询 $Q(p)$：通过路径表达式 p 对相应的 XML 文档进行查询，查询结果是满足条件 p 的所有节点集，并读取各节点的内容。

(2)插入 $I(n, v)$：在节点 n 下插入一个新的节点 m，其值为 $V(m)=v$，并在 Xp-tree 数据模型中增加一条边(n, m)。

(3)删除 $D(n)$：删除以节点 n 为根的子树，并删除从 n 的父节点到 n 的边。

(4)更新 $U(n, v)$：改变节点 n 的值为 v，对一个节点的更新只是对其内容(CDATA 和属性值)的更新，以 v 的值来代替先前的值。例如 $U(n_{18}, \text{“08003”})$表示将节点 n_{18} 的值改为“08003”。

3　Native XML 数据库并发控制协议

3.1　锁协议模型

在 Native XML 数据库并发控制中，针对上面所提出的 *Xp-tree* 数据模型以及操作模型定义 6 种锁类型：

(1)IR_C-lock 锁：当一个事务 T 执行查询操作 $Q(p)$时，首先需要对路径表达式 p 上的节点申请 IR_C-lock 锁。若当前节点没有与 IR_C-lock 锁相冲突的其他锁，则调度器允许对该节点加锁，加锁成功后事务 T 可以继续执行；否则，事务 T 将处于等待队列。IR_C-lock 锁适用于非终节点，其中 C 为下一个将要进行加锁的节点(一般为当前节点的子节点)。

(2)R-lock 锁：用于对查询操作 $Q(p)$结果的节点集进行加锁，其加锁粒度为子树级。只有当前活动的事务 T 已经获得对所要读取的节点的 R-lock 锁时，事务 T 才可以进行读取操作；否则，事务 T 将处于等待队列。例如当事务 T 执行查询操作 Q(/Department/Students/Student)时，需执行以下加锁序列：$IR_{C1}(r)IR_{C2}(n_1)IR_{C3}(n_2)R(n_4)R(n_5)$，其中，$C1$, $C2$, $C3$ 分别为：“Department”, “Students”, “Student”。

(3)IC-lock 锁：当一个事务 T 执行更新操作(如插入 I、删除 D、更新 U)时，首先须对从根节点到目标节点之间的节点申请 IC-lock 锁。若当前节点没有与 IC-lock 锁相冲突的其他锁，则调度器允许对该节点加锁，加锁成功后事务 T 可以继续执行；否则，事务 T 将处于等待队列。IC-lock 锁适用于非终节点，加锁粒度为节点级。

(4)A_C-lock 锁：当一个事务 T 执行插入操作 $I(n, v)$时，首先须对目标节点 n 申请 A_C-lock 锁。若当前节点 n 上没有与 A_C-lock 锁相冲突的其他锁，则调度器允许对该节点 n 加锁，加锁成功后事务 T 可以继续执行；否则，事务 T 将处于等待队列。A_C-lock 锁适用于非终节点，且加锁粒度为节点级，C 表示新插入的节点。例如当事务 T 须在<Student student_id=“08001”>节点下增加一个元素“Addr”时，须执行加锁序列：$IC(r)IC(n_1)IC(n_2)A_C(n_4)$，其中，$C$=“Addr”。

(5)D-lock 锁：当一个事务 T 执行删除操作 $D(n)$时，首先须对目标节点 n 申请 D-lock 锁。若当前节点 n 上没有与 D-lock 锁相冲突的其他锁，则调度器允许对该节点 n 加锁，加锁成功后事务 T 可以继续执行；否则，事务 T 将处于等待队列。D-lock 锁可用于终节点和非终节点，加锁粒度为子树级。例如当事务 T 执行删除节点<Student student_id=“08002”>时，须执行加锁序列：$IC(r)IC(n_1)IC(n_2)D(n_5)$。

(6)U-lock 锁：当一个事务 T 执行更新操作 $U(n, v)$时，首先须对目标节点 n 申请 U-lock 锁。若当前节点 n 上没有与 U-lock 锁相冲突的其他锁，则调度器允许对该节点 n 加锁，加锁成功后事务 T 可以继续执行；否则，事务 T 将处于等待队列。U-lock 锁仅用于终节点，其加锁粒度为节点级。例如当事务 T 执行更新操作 $U(n_{19}, \text{“Zhang Juan”})$时，须执行加锁序列：$IC(r)IC(n_1)IC(n_2)IC(n_4)IC(n_8)U(n_{19})$

表 1 给出了上述 6 种锁之间的兼容关系。

表 1　锁兼容性矩阵

锁请求	锁持有					
	IR_C	IC	R	A_C	U	D
IR_C	+	+	+	→	×	−
IC	+	+	−	+	×	−
R	+	−	+	−	−	−
A_C	→	+	−	+	×	−
U	×	×	−	×	−	−
D	−	−	−	−	−	+

其中，＋表示兼容；－表示冲突；→表示条件兼容；×表示不可能存在冲突。可见，D-lock 锁与其他锁之间是不兼容的。IR_C-lock、IC-lock、A_C-lock 与 U-lock 锁不可能存在冲突，由于 IR_C-lock、IC-lock 和 A_C-lock 锁适用于非终节点，而 U-lock 锁则适用于终节点。理论上，2 个不同的事务 T_1 和 T_2 可以在同一个节点 n 下执行插入操作[3]，因此 A_C-lock 与 A_C-lock 锁之间是相互兼容的。假定 2 个事务 T_1 和 T_2 分别对节点 n 申请 A_{C1}-lock 和 IR_{C2}-lock 锁，则当 $C1=C2$ 时，两者相冲突，因为事务 T_1 是要增加一个节点 $C1$，而事务 T_2 是要读取节点 $C1$ 的内容，这样会导致幻象问题。所以 A_C-lock 与 IR_C-lock 之间是条件兼容的。

3.2　锁协议规则

上述定义的 6 种锁，须满足下列规则：

(1)申请锁规则：事务在执行任何一种操作之前，必须申请合适的锁类型，并且在对当前节点的子节点申请加锁时，必须综合考虑事务的操作类型以及子节点的类型来选择合适的锁类型。

(2)冲突检测规则：一个事务在获得相应的锁之前，必须根据锁的兼容性矩阵对该锁与当前节点所持有的锁的兼容性进行检测。

(3)2 段锁协议规则：每一个事务都必须遵守 2 段锁(2PL)协议。

例如，在上文所示的 XML 文档片段上，假设事务 T_1 执行查询操作 Q(/Department/Students/student)，读取节点 student 的内容，事务 T_2 执行更新操作 $U(n_{25}, \text{“23”})$，将节点 n_{25} 的值由 21 改为 23。图 2 是这 2 个事务的一个并发调度。可以看出事务 T_2 的 $IC(n_5)$操作与事务 T_1 的 $R(n_5)$操作相互冲突，因此 $IC(n_5)$必须在 $Unlock(n_5)$操作完成之后执行。

T_1	T_2
$IR_{C1}(r)$	$IC(r)$
$IR_{C2}(n_1)$	$IC(n_1)$
$IR_{C3}(n_2)$	$IC(n_2)$
$R(n_4)$，$R(n_5)$	
Read	
$Unlock(n_4)$，$Unlock(n_5)$	
$Unlock(n_2)$	$IC(n_5)$
$Unlock(n_1)$	$IC(n_{14})$
$Unlock(r)$	$U(n_{25})$
	Update
	$Unlock(n_{25})$
	$Unlock(n_{14})$
	$Unlock(n_2)$
	$Unlock(n_1)$
	$Unlock(r)$

图 2　事务 T_1 和 T_2 的一个并发调度

3.3　可串行化分析

可以证明，若并发执行的所有事务均遵守 2 段锁协议，则对这些事务的任何并发调度策略都是可串行化的。结合上文，可证明本文所提出的锁模型是正确的，而且遵守 2 段锁协议，因此，本文提出的锁模型是可串行化的。

3.4　性能分析

目前，Native XML 数据库正处于发展初期，在事务及并发控制方面的研究尚不系统，虽然已提出一些并发控制协议[1-4]，但都存在一些缺陷。而本文所提出的并发控制协议有效地解决了这一问题，主要表现在以下几个方面：

(1)幻象问题。如文献[2]所提出的 XLP 协议，它假定 *R*-lock 锁和 *I*-lock 锁是相互兼容的，但当一个事务 T_1 在当前节点 *n* 下执行插入操作，而另一个事务 T_2 需要读取当前节点 *n* 的内容时就会导致幻象问题。而本文所提出的并发控制协议通过 A_C-lock 与 IR_C-lock 锁的条件兼容性有效地解决了这一问题。

(2)根据锁的类型采用合适的加锁粒度，能有效减少加锁的数量。先前提出的部分并发控制协议并没有考虑到这一事实。而本文所提出的锁模型既有加在节点级上的锁，也有加在子树级上的锁，因此提高了并发度。

(3)高效性。一方面，利用本文所提出的并发控制协议，可以允许 2 个不同的事务 T_1 和 T_2 在同一节点下同时进行更新操作(如插入、删除、更新)；另一方面，通过 A_C-lock 与 IR_C-lock 加锁模式允许一个事务读取节点 *n* 的一棵子树集(或一个子节点)的内容，而另一个事务则在当前节点 *n* 下插入一个不同类型的子节点，二者可以并发执行。例如，在上文列出的 XML 文档片段中，假设事务 T_1 执行查询操作，读取 student_id 为 08002 的学生的姓名(即 *Name* 的值)，事务 T_2 执行插入操作，在当前节点下插入该生的籍贯信息 Addr，事务 T_3 执行更新操作，更新该生的年龄信息 Age。理论上，事务 T_1, T_2 和 T_3 是可以并发执行的。但在先前提出的部分并发控制协议中，三者是相互冲突的。利用本文所提出的并发控制协议，事务 T_1, T_2 和 T_3 可以并发执行，因为事务 T_1 的加锁序列为：$IR_{C1}(r)IR_{C2}(n_1)IR_{C3}(n_2)IR_{C4}(n_5)IR_{C5}(n_{12})R(n_{23})$，事务 T_2 的加锁序列为：$IC(r)IC(n_1)IC(n_2)A_C(n_5)$，事务 T_3 的加锁序列为：$IC(r)IC(n_1)IC(n_2)IC(n_5)IC(n_{14})U(n_{25})$, $IR_{C4}(n_5)$, $A_C(n_5)$ 与 $IC(n_5)$是相互兼容的($C4 \neq C$)。

4　结束语

并发控制是 Native XML 数据库研究领域的一个难点。本文从 XPath 角度出发，提出一种新的基于 XPath 的并发加锁协议，根据 XML 数据库的操作类型，提出一种简化的数据模型，在此模型上实现了数据库的多用户事务的并发控制。通过分析可以看出，采用此协议提高了数据库的并发度，为 Native XML 数据库完整事务机制的实现提供了一定的理论支持。下一步的工作是将此协议应用到 Native XML 数据库系统中，进行实验验证。

参考文献

[1] Dekeyser S, Hidders J, Paredaens J. A Transaction Model for XML Databases[J]. World Wide Web: Internet and Web Information Systems, 2004, 7(l): 29-57.

[2] Jea K F J, Chen S Y, Wang S H. Concurrency Control in XML Document Databases: XPath Locking Protocol[C]//Proc. of the 9th International Conference on Parallel and Distributed Systems. Taiwan, China: [s. n.], 2002.

[3] Choi E H, Kanai T. XPath-based Concurrency Control for XML Data[C]//Proc. of the 14th Data Engineering Workshop. Kyoto, Japan: [s. n.], 2003.

[4] Pleshachkov P, Novak L. Transaction Isolation in the Sedna Native XML DBMS[C]//Proc. of the Spring Young Researcher's Colloquium on Database and Information Systems. Siberian, Russia: [s. n.], 2004.

编辑　金胡考

(上接第 29 页)

由表 1~表 3 可以看出，改进的多通道水印模型在较好保持了水印图像隐形性的前提下，有效提高了水印提取的准确度。由于该算法模型没有实际嵌入参考水印，因此其水印图像的质量高于多通道水印模型。基于该算法模型的鲁棒水印检测正确率与多通道水印的相当，明显优于单通道水印算法。

参考文献

[1] 谢荣生, 徐耀群, 赵建东, 等. 小波域多通道图像数字水印嵌入方法[J]. 哈尔滨商业大学学报: 自然科学版, 2002, 18(5): 508-511.

[2] 谢荣生, 郝燕玲, 杨树国. 混沌二维置换网络的设计及其在图像数字水印隐藏中的应用[C]//全国第三届信息隐藏学术研讨会议论文集. 西安: 西安电子科技大学出版社, 2001: 88-92.

[3] Deepa K, Dimttrios H. Digital Watermarking Using Multiresolution Wavelet Decomposition[C]//Proc. of IEEE International Conf. on Acoustics Speech and Signal Processing. Seattle, USA: [s. n.], 1998: 2969-2972.

编辑　陈　晖