面向对象数据库死锁检测方法研究*

郝朝辉 麦中凡

(北京航空航天大学计算机系 北京 100083)

摘要 本文根据面向对象数据库系统中嵌套事务的新的执行特性,提出了一种检测死锁的算法——扩充的边追踪算法,并在一个面向对象数据库原型系统MIDS中予以实现.

关键词 数据库,面向对象数据库,分布式数据库,事务,并发控制.

传统的原子事务(Atomic Transaction)模型可以提供对数据库数据访问的并发控制,即各事务可以按照一定的封锁、时间戳(Time Stamp)、或乐观方法达到可串行化执行的要求.[1]随着用户需求和数据库技术的不断发展,作为对传统事务模型的扩展,嵌套事务模型被越来越多的数据库系统,尤其是面向对象的数据库系统所采用.

嵌套事务(Nested Transaction)模型

如果一个事务可包含若干个子事务,同时每个子事务又可能由一定数目的子事务组成,则这种事务结构就是嵌套事务.[2]不包含在其它事务中的事务称为顶层事务(Top-level Transaction),有子事务的事务称为父事务,子事务是其孩子.事务的子嗣(祖先)关系是事务的孩子(父亲)关系的自反传递闭包,非自反的子孙(祖先),称其为事务的后辈(前辈).

与原子事务相比,嵌套事务不仅可以提高事务内部的并行性,而且在事务内部也能进行恢复控制,从而可大大提高系统的模块化水平[3],这正符合面向对象数据库系统的要求.

按照ODMG-93标准,嵌套事务有以下不同于传统事务的执行特性[4]:子事务的提交与流产不影响父事务的状态.父事务的流产将导致其子事务全部流产.子事务通过abort_to_top_level方法也可以造成全部嵌套事务的流产.子事务可以访问父事务拥有的数据,而不论其是否已经提交.

由于以上要求,嵌套事务模型下的并发控制也较传统事务复杂得多.以封锁方法为例,集中式嵌套事务的封锁规则为一个扩充的两阶段锁协议(2PL)——嵌套事务两阶段锁协议(N2PL).若只考虑读、写两种锁类型,则N2PL协议可概述如下:

事务仅当其持有某对象的读锁或写锁时,才能对它进行读操作.事务仅当其持有其对象的写锁时,才能对它进行写操作.如果某对象的写锁持有者都是事务的前辈事务,则事务可以持有该对象的读锁.如果对象的读、写锁持有者都是事务的前辈事务,则事务可以持有该

---

* 作者郝朝辉,1969年生,硕士,主要研究领域为系统软件,面向对象数据库.麦中凡,1935年生,教授,主要研究领域为软件工程,数据库,程序设计语言,智能工具.

本文通讯联系人:郝朝辉,北京 100036,北京162信箱中国计算机报市场研究部

本文1995-12-13收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

对象的写锁.当事务提交时,其父事务将持有提交事务的全部锁,如果父事务已经持有某些锁,则它应持有具有更高访问权限的锁.当事务流产时,它将丢弃所持有的全部锁.

对于分布式数据库系统来说,除上述协议外,事务在提交时还应遵从分布式事务提交协议.[2]

由于事务在存取数据之前必须先封锁对象,而对象又是一种不可预先抢占的耐用性资源,所以若在资源分配中采用封锁的策略,就有发生死锁的可能.[1]正因如此,死锁处理是每个数据库系统都不能回避的问题.

本文下面的章节将对死锁问题进行具体的讨论.第1节讨论传统死锁处理方法,第2节给出了一个分布式数据库嵌套事务死锁检测算法——扩展的边追踪算法(extended edge-chasing algorithm).接着,我们给出一个使用该算法的实例.

1　死　锁

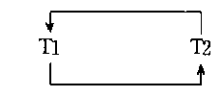
关于死锁,在操作系统中已做过充分研究.一般来说,数据库系统中发生死锁的必要条件是等待条件,事务已经持有分配给它的资源,又去申请并等待另外的资源;非剥夺条件,除非事务释放其占有的资源,否则不强制夺走它所持有的资源;循环等待条件,事务处在互相等待的循环链中.

解决死锁的方法可以分为以下3类:即死锁的预防、避免、死锁检测及恢复.

为预防死锁,只需使存在死锁的必要条件之一不成立即可.文献[1,5]针对传统事务模型提出了比较详尽的解决方法.但是,对于OODBMS来说,由于事务模型和对象模型的复杂化,其中的大部分方法已不再适用,所以OODBMS采取的主要方法是通过准确预测(或限定)事务要求的资源集合来达到预防死锁的目的,这种方法要求事务在开始其操作前必须得到可能会用到的所有资源,同时其它事务又不会请求对这些资源的使用.由于事务可能会用到相当大的数据资源,因此,显然这种作法会极大地降低系统的并发程度.

与预防相比,死锁避免采用了不同的策略.当事务提出对资源的申请时,死锁避免方法令其等待一定长度的时间,如果时间片用完,事务仍然没有得到该资源,系统则使资源的申请方或占有方一方流产.ORION采用了这种死锁处理方法.[6]更复杂一些的算法可能还会包括判定事务优先级的策略.死锁避免法比预防实现简单,也更灵活.但是这种方法可能会造成不必要的事务流产,还有可能会引起事务的反复重启和流产,即活"死锁".

在分布式数据库系统中采用最多的方法是死锁检测法.本质上讲,死锁检测算法就是找出有向图上的循环路径,即发现事务的循环等待链.如图1所示,我们用事务请求资源图表示出一个由2个事务形成的最简单的死锁情况.



说明:T1持有R1,请求R2
T2持有R2,请求R1
图1

在分布式系统中进行死锁检测的方法根据系统资源请求和占有信息的集中程度又可以有集中检测法(Centralized Detection)、分级检测法(Hierarchical Detection)和边追踪方法等等.死锁检测法不会造成很多不必要的事务流产,但是却增加了系统的额外开销和复杂度.

© 中国科学院软件研究所 http://www.jos.org.cn

2 边追踪方法

2.1 传统事务模型下的边追踪方法

所谓边追踪方法，就是沿资源请求图上的等待边方向传递信息，并由各结点根据目前收集到的信息，判断是否出现了死锁. 边追踪方法是一种全分布的检测方法，它比集中检测法和分级检测法具有更高的检测效率，而且由通信延迟引起的"幻象"死锁情况更少.

传统事务模型下的边追踪方法可以描述如下：

第1步，初始化阶段. 当事务开始等待某个锁时即进入初始化阶段. 由于事务和其请求的对象都位于同一结点(只有这样事务才可以申请该锁)，因此该结点上的事务管理系统(Transaction Management System)就可以取得有关事务和所请求锁的信息，于是，TMS将生成一个列出所有阻塞该等待事务的阻塞列表. 在资源请求图上，这表现为从等待事务到列表中各事务间各有一条等待有向边. 接着，TMS将向这些事务所在结点的TMS发送检测消息. 各检测消息除了必要的与通信有关的信息外，还包含了一个路径列表，在该表中列出了本地事务和占有资源的事务的事务号. 我们用2个事务号表示出在这2个事务间有1条等待有向边存在.

第2步，检测阶段. 当TMS收到检测消息时，就会知道另一个结点上的某个事务正在等待某个本地事务的完成. 接着，TMS将检查本地事务是否也处在等待状态. 如果是，TMS将检查本地事务是否已经被列入了检测消息中的路径列表，如果已经有了，则意味着找到了一条事务两两互相等待的循环路径，即发现了死锁；否则，就象第1步所描述的那样，向所有与本地事务相冲突的结点发消息，只不过这时发出的检测消息在路径列表里又增加了本地事务号. 如果本地事务并没有处在等待状态，则认为没有死锁发生，并结束检测. 例如，如图2所示的死锁情况(设各事务分别位于结点A,B,C).

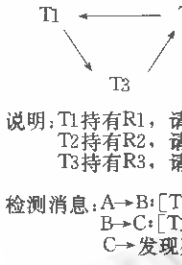


图2

第3步，死锁恢复阶段. 选用某种策略有选择地使某些事务流产. 选择的标准是使损失最小，即使得事务流产的开销及以后重启事务的开销极小. 开销的衡量尺度通常用事务运行CPU时间、执行数据修改的数目和事务已经占有的资源数等等来计算，也可以采用多种策略相结合的方法. 如Objectstore系统的5种流产策略：用户指定、当前事务(Current)、执行时间最短(Youngest)、工作最少(Least Work)和随机(Random)流产.[7]

2.2 扩展的边追踪方法

上节中所描述的算法并不适用于嵌套事务模型下的死锁检测情况，原因是其没有考虑到嵌套事务封锁规则的独特之处，即子事务提交后，所有该子事务的锁都要转给其父事务，而不是释放掉. 这样，所有等待该子事务持有锁的事务都只有等到其父事务，甚至其前辈事务都结束后才有可能得到锁. 实际上，从某种程度上讲，前辈事务也在等待其后辈事务的锁，否则它们就不能提交.

例如，如图3所示，假设有如下事务结构和资源请求情况(我们用树形结构来表示事务之间的关系，如T2和T3是T1的子事务，依此类推).

如图3所示，在资源请求图上并没有出现明显的回路. 但是，根据N2PL规则，T4却不

© 中国科学院软件研究所 http://www.jos.org.cn

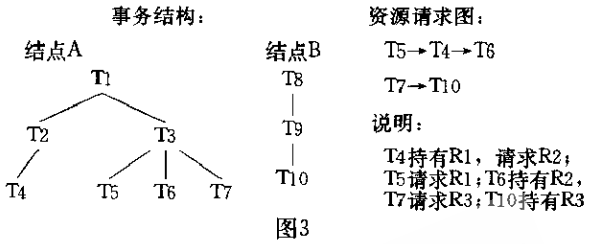


图3

可能得到对 R2 的锁. 这是因为,T6 在提交后,其对 R2 的锁要转给 T3,而 T3 只有在 T5 提交或流产后才能提交或流产,并释放对 R2 的锁. 但同时,由于 T5 正在等待 T4 释放 R1,而 T4 又在等待 T6,实际上,也就是在等待 T5,T6 的父事务 T3,这样,实际上就造成了 T4 与 T3 之间的死锁. 而 T7,T10 之间的资源请求情况则不会产生死锁.

基于这种情况,传统事务模型下的边追踪算法应该做以下修改:

定义. 最久等待事务是指满足下列 2 个条件的事务:①不是请求锁事务的前辈事务. ②是持有冲突锁事务的最小前辈事务. 最久等待事务其实是请求事务所要等待的最后释放冲突锁的事务,如前例中的 T3.

对算法的修改描述如下:

第 1 步,初始化阶段. 当事务开始等待时,不再向阻塞列表中的事务发检测消息,而向相应的最久等待事务发检测消息. 例如,前例中原算法将向 T6 发消息,现在则向 T3 发消息. 然而,检测消息中的内容仍然同原算法一样,如前例中应为[T4,T6].

第 2 步,检测阶段. 当事务 T 收到检测消息时,比如前例中 T3,TMS 就会检测 T 是否也在等待. 如果是,T 和锁的持有者 T′也应被加到检测消息中的路径列表里,并向另一个最久等待事务(T′的前辈)发送检测消息. 同样,如果 TMS 发现 T 正在等待的事务是检测消息路径列表中的某个事务或其前辈,则可判定出现了死锁.

除了向 T 等待的每个最久等待事务发出新的检测消息以外,原来的检测消息也要由 TMS 发送到 T 的每个子事务,以便判断 T 是否在等待其后辈事务. 如在前例中,把从 T4 接收来的检测消息再依次发送给 T5,T6,T7. 因为 T6 未处在等待状态,所以与死锁无关. 而 T5 则在等待 T4 持有的锁,这样就发现了在上例中讨论的死锁情况. 同样,T7 也接收了检测消息,并向 T10 发出消息[T4,T6,T7,T10],但是这里并没有出现死锁.

第 3 步,死锁恢复仍可按原算法执行.

2.3　算法的进一步完善

在以上的算法中,我们并没有考虑效率问题. 而实际上,上述算法至少还应在以下 2 个方面进行改善:①发送的消息太多. 处在死锁回路上的每个结点都要发送检测消息,而且这些消息还要分别被其它结点转发,所以,发送消息的复杂度为 $O(n^2)$,开销太大. ②如果同时在不同地点发现死锁,则检测算法就可能会使多个事务流产,这不符合尽量减少流产事务的原则.

基于以上原因,我们引入了事务优先级的概念. 每个事务都按照一定的原则(父事务优先级高于子事务等等)赋以固定的优先级,在发送消息和选择流产事务时将参照事务之间优先级的比较结果采取不同的动作. 如:只有当高优先级事务在等待低优先级事务时,才开始死锁检测. 当事务优先级低于引起追踪的事务的优先级时,则停止追踪. 发现死锁时,使具有最低优先级的事务流产,以解除死锁.

另外,还要考虑到系统的异常情况,如结点崩溃、消息丢失等等. 一般来说,结点崩溃只

© 中国科学院软件研究所 http://www.jos.org.cn

会引起该结点上所有事务流产，不会造成新的死锁，所以不必考虑. 而消息丢失，则有可能检测不到死锁的存在. 要解决这个问题，最简单的方法是让处在等待状态中的事务周期性地重发检测消息，直到该事务不再处于等待状态，即流产或已经得到所请求的锁为止. 采用重发的方法并不需要来自接收方的确认，因而开销不大.

3 在MIDS中的实现

MIDS是我们正在SUN工作站用SUN C++语言开发的一个OODBMS原型系统. 在文献[9,10]中有该系统的总体情况介绍.

MIDS的客户/服务器体系结构具有如下特点：MIDS服务器是一个页式服务器. 提供一个单一的全局服务器进程. 多个客户进程可在网络中随意分布.

MIDS存储管理系统(Storage Management System)采用虚拟内存映射技术(Virtual Memory Mapping Technology)实现，其基本原理是利用UNIX操作系统的页保护功能，当事务试图访问内存中还未调入的页时，操作系统将发出页错误信号，存储管理系统截获此信号，并完成相应页的调度. SMS对读页和写页操作也用类似的方法处理.

MIDS事务管理系统的设计要点可描述如下：

TMS分为2部分，一部分位于服务器，负责维护与数据占有和请求情况有关的信息. 另一部分位于各个客户机，负责与服务器进行通信及本机的事务管理工作.

重要的数据结构有：封锁表、申请表和事务表，所有重要数据都采用哈希表结构进行组织，以提高访问速度.

提供对深度可达7层的嵌套事务支持，同时，还可以支持某些特定事件所触发的规则，所有事务都划分为8个优先级别，即系统级、级别1到级别7级.

根据MIDS的体系结构和事务管理系统的设计实现，在死锁检测时，不再需要进行客户机和服务器关于资源占有、请求情况的通信. 另外，由于在嵌套事务内部采用了串行执行方式，死锁可能会出现的形式也有所减少，这就使得我们可对上节中讲到的算法作一简化.

基于上述内容，对于MIDS事务管理系统的封锁管理器来说，当某事务提出对数据资源的请求时，先由存储系统计算这一请求所对应的虚存页号，然后再由封锁管理器进行死锁检测.

MIDS死锁检测算法是一个递归算法，描述如下：

0. 死锁标志置假.

1. 生成阻塞列表，即把阻塞当前事务的各事务的事务号加入进去. 生成路径列表.

2. 对应于每个阻塞事务T，执行下列动作：

找到其最久等待事务(如前所述，由于MIDS采用串行子事务执行方式，所以其最久等待事务即为当前进程中的顶层事务. 同时又因为，在MIDS事务模型下，占有数据事务的前辈事务不可能正在等待其它事务，所以它们可以不必向其它事务发消息)，现在，从顶层事务开始，沿事务结构树向下，一直到达最底层事务T′，查看是否有某个事务已经在路径列表中出现过，若有，则置死锁标志，从本次递归中返回. 否则把这些事务的事务号加入路径列表，继续.

死锁标志已置？ 若已置，则从本次递归中返回.

若没有，则继续.

T′在等待吗？ 若没有，则从本次递归中返回.

若也在等待，则生成T′自己的阻塞列表，转2，进入下一级递归.

3. 若有死锁，则找出优先级最低的事务，将其流产，解除死锁. 否则返回.

© 中国科学院软件研究所 http://www.jos.org.cn

4 总 结

在本文中，我们讨论了一种检测死锁的方法——边追踪算法，并给出了一个实例. 在设计算法时，我们采用了静态确定事务优先级的作法，实际上，有时这种作法并不令人满意. 比如，它可能会使某些低优先级事务长期得不到执行，与如何安全地退出一个正在运行的事务一样，这将是我们下一步工作的方向.

参考文献

1 周龙骧. 数据库管理系统实现技术. 武汉：中国地质大学出版社，1990.

2 Moss J E B. Nested transactions：an approach to reliable distributed computing [Ph. D. thesis]. MIT.，1981.

3 Beeri C，Bernstein P A，Goodman N. A model for concurrency in nested transaction system. Computer Science Press，1986.

4 Cattell R G G. The object database standard：ODMG-93；release 1.1. MorganKauf-Man Publishers，Aug. 1993.

5 邓胜辉，楼荣生. 死锁预防方法研究. 第 11 届全国数据库学术会议论文集，1993. 144～148.

6 Won Kim. Architecture of the ORION next-generation database system. IEEE Transactions on Knowledge and Data Engineering，1990，2(1).

7 Ojbect Design Inc. Objectstore Release 2.0 User's Guide. Objectstore Release 2.0 Reference Manual，1993.

8 Moss J E B. Transaction management for object—oriented systems. OODBMS'1986，1986.

9 车敦仁，麦中凡. 多媒体智能数据库系统 MIDS/BUAA 的总体设计. 计算机科学，1994，(2).

10 车敦仁，麦中凡. MIDS/BUAA 的对象数据模型及其特征. 计算机科学，1994，(2).

RESEARCH OF AN ALGORITHM FOR DETECTING DEADLOCK IN OODBMS BASED ON NESTED TRANSACTION MODEL

Hao Zhaohui　Mai Zhongfan

(*Department of Computer Science　University of Aeronautics and Astronautics　Beijing*　100083)

Abstract　Based on the new characteristics of nested transactions of OODBMS, this paper proposes an algorithm for detecting deadlock——the extended edge-chasing algorithm. Furthermore, an example which the authors have fulfilled in MIDS prototype system is also provided.

Key words　Database, object-oriented database, distributed database, transaction, concurrency control.

© 中国科学院软件研究所 http://www.jos.org.cn