实时数据库事务的互斥与同步*)

廖国琼　刘云生　王洪庭
(华中科技大学计算机学院　武汉430074)

摘　要　实时数据库事务具有许多不同于传统数据库事务的特性，其互斥和同步问题远比传统数据库事务复杂。该文结合已研制成功的嵌入式实时数据库系统ARTs-EDB，重点讨论实时数据库事务的互斥与同步技术。事务互斥用于实现对数据库共享数据"互斥地"访问。根据数据共享特性的不同，实时事务的互斥可分为两类，一类是通过设置"优先级相关"互斥量而实现的互斥，另一类是通过数据库系统提供的并发控制策略而实现的互斥。事务同步是实时数据库事务的特有要求，主要用于实现嵌套事务内部事务的互相等待与互通消息，以保证嵌套事务的内部一致性。

关键词　实时数据库，事务管理，并发控制，互斥，同步

Mutext and Synchronization of Transactions in Real-Time Databases

LIAO Guo-Qiong　LIU Yun-Sheng　WANG Hong-Ting
(School of Computer Sci. &Tech., Huazhong University of Sci. &Tech., Wuhan 430074)

Abstract　For there are many particular characteristics for transactions in real-time databases, the transaction management mechanisms of real-time databases are more complex than traditional databases. In this paper, the mutex and synchronization thchniques of real-time transactions in an embedded real-time database system—ARTs-EDB are discussed. The mutex techniques are used by real-time transactions to access databases exclusively. Two classes mutex techniques are adopted in the system according to different shared properties of data: one class is the "Priority aware" mutexes in real-time operation systems and another class in the concurrency control mechanisms in database systems. The synchronization thehniques are unique to eal-time transactions, which are used for the internal sub-transactions of nested real-time transactions to wait and communicate with each other to guarantee the internal consistency of nested transactions.

Keywords　Real-time databases, Transaction management, Concurrency control, Mutex, Synchronization

1　引言

实时数据库中的事务(后称实时事务)组合了实时任务和传统数据库事务两者的特性：既需保证数据库一致性，还须满足事务的定时限制。因此，实时事务具有许多不同于传统数据库事务的特性，如结构复杂性、功能替代性、结果补偿性、时间相关性和执行依赖性等[1]。

如同操作系统的进程(或线程)，数据库系统中的事务并非固定处于某个状态，而是随自身的推进和外界条件的变化而发生变化。其原因是数据库系统允许多个事务并发执行，这些事务在执行过程中往往并不完全独立，而是相互制约，如一个事务可能要等待另一事务释放所持有的资源才能继续执行。特别地，对于实时事务，为避免优先级倒置(即低优先级事务阻塞高优先级事务)，允许高优先级事务抢占低优先级事务所占有的资源(如CPU和数据)。而且，实时事务往往是具有复杂嵌套结构的事务，它们之间存在多种依赖关系(如开始依赖、提交依赖、夭折依赖等)[2]。因此，实时事务的互斥和同步问题远比传统数据库事务复杂。

有关实时数据库的研究已近20年，并已取得了大量的研究成果[3]。除对实时数据库的并发控制机制已有大量研究外，未见其它有关实时事务互斥和同步的论述。笔者认为，实时事务的互斥和同步涉及到实时数据库的低层实现技术，而一般的研究和讨论只是基于一定的实验模型进行理论研究和分析，而未考虑具体的实现技术。而且，对于不同的实现环境和所选择的实现策略，实时事务所采用的互斥与同步技术也不尽相同。因此，本文拟结合已研制成功的嵌入式实时数据库系统ARTs-EDB，讨论实时事务的互斥与同步问题及其实现技术。

ARTs-EDB是基于嵌入式应用环境开发的集中式实时数据库管理系统。为更好支持实时要求，该系统基于内存数据库实现，即数据库的工作版本存放在内存，而外存作为数据库后援。内外存数据库均采用"区-段式"结构组织[4]。由于线程具有比进程占用更少资源、线程间通讯与切换的开销不大以及线程调度较为灵活等优点，因此ARTs-EDB采用多线程(Multi-Threads)结构。整个数据库系统只有一个服务进程，每当有数据库请求时，就创建一个线程为之服务，即一个数据库事务对应一个线程。因此，对于操作系统而言，事务是以线程为单位去竞争系统资源(包括CPU和数据)的。

*)项目资助：国家自然科学基金项目(编号60073045)，国防武器装备预先研究项目(编号413150403)，中国博士后科学基金项目(编号2003034482)，华中科技大学博士后科学基金项目(编号2003021)。**廖国琼**　博士，主要研究方向为现代(实时、主动、内存、移动、工程等非传统)数据库理论与技术及其集成实现、数据库与信息系统开发。**刘云生**　教授，博士生导师，主要研究方向为现代(实时、主动、内存、移动等非传统)数据库理论与技术及其集成实现、数据库与信息系统开发、实时数据工程、软件方法学与工程技术。

2 实时事务的互斥

实时事务互斥的主要任务是解决并发执行事务对共享数据"互斥地"访问，并避免优先级颠倒。在实时数据库系统中，事务所访问的共享数据可分为两类。一类是访问完后可立即释放的数据。这类数据主要是用于事务管理的数据，如事务表、锁表及各种事务管理队列(如接纳、就绪、运行和等待队列)等。而另一类是待事务执行结束(提交或夭折)后才能释放的数据(事务 ACID 特性要求)。这类数据是指数据库中的数据和数据字典中的数据(在 ARTs-EDB 中，DML 和 DDL 语句都当作事务执行)。对于前者可采用操作系统的互斥量方式实现，而后者则必须采用数据库管理系统本身所提供的并发控制机制实现，以下将对这两种方式分别论述。

2.1 互斥量方式

由于事务是以线程身份去竞争资源，因此，对于事务管理共享数据的互斥可通过设置操作系统线程级互斥量实现，即在访问共享数据前加锁互斥量，访问完后再对互斥量解锁。

在实时数据库中，通常最高优先级事务应优先执行，而对于具有相同优先级的事务，可选择先进先出(SCHED-FIFO)策略或时间片轮转(SCHED-RR)策略进行调度。这种考虑优先级的实时调度最终要通过实时操作系统提供 CPU 可抢占机制和实时调度策略实现。

但是，若事务在执行过程中优先级不允许发生改变，就可能由于在互斥量上的等待而导致优先级倒置。图1是一个优先级倒置的例子。事务到达的顺序是 T_1 最先，T_2 次之，T_3 最迟。事务优先级高低是 T_1 最低，T_2 最高，T_3 居中。在 t_1 时刻 T_1 对某互斥量上锁；在 t_2 时刻 T_2 来到，抢占 T_1 占有的 CPU；在 t_3 时刻 T_2 由于等待 T_1 拥有的互斥量而阻塞，T_1 恢复执行(以原来的优先级)；t_4 时刻 T_3 来到，抢占 T_1 占有的 CPU 直到 t_5 时刻运行结束。这样，在[t_4，t_5]时间区间，由于在互斥量上的等待而出现低优先级事务 T_3 阻塞高优先级事务 T_2 的现象。

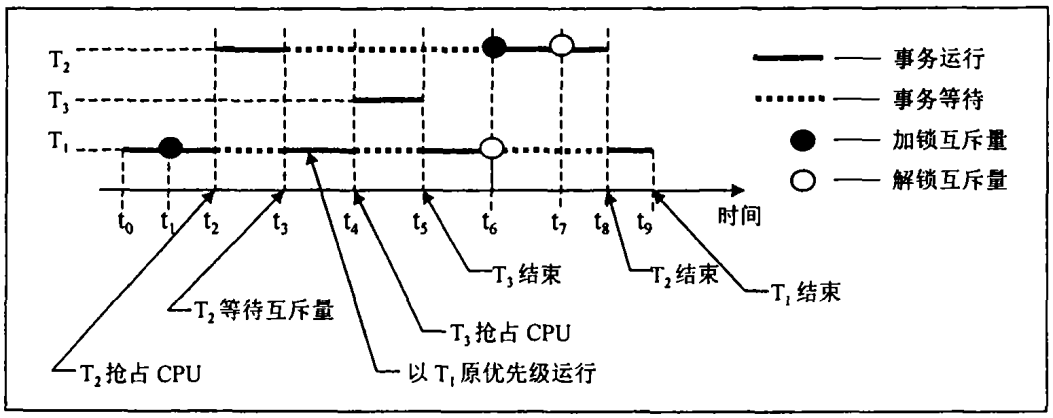


图1 优先级倒置

为解决上述问题，实时操作系统提供了优先级相关(priority aware)互斥量，通过设置互斥量的属性来改变拥有互斥量线程的优先级，以避免优先级倒置的发生。

POSIX 标准规定了两种优先级相关互斥量：优先级天花板(priority ceiling)互斥量和优先级继承互斥量(priority inherit)[5]。优先级天花板协议是指当一个线程拥有一个互斥量时，它将以互斥量指定的优先级进行(该互斥量的优先级可事先设置)。这样，任何锁住该互斥量的线程将自动将它的优先级提高到互斥量的优先级运行。而优先级继承协议是指当一个线程在由另一个低优先级线程拥有的互斥量上等待时，拥有互斥量的线程将被提升到被阻塞线程的优先级。这样，拥有互斥量的线程代表高优先级线程工作。当解锁互斥量时，其优先级将自动降到原来的优先级，并唤醒等待该互斥量的高优先级线程。

图2为采用优先级继承协议(ARTs-EDB 所采用)解决优先级倒置的例子。t_3 时刻 T_2 等待 T_1 拥有的互斥量，T_1 将提高到 T_2 的优先级执行，即使 T_3 在 t_4 时刻来到，也必须等待。当 T_1 在 t_5 时刻解锁互斥量时，恢复原来优先级，此时唤醒最高优先级事务 T_2 执行。

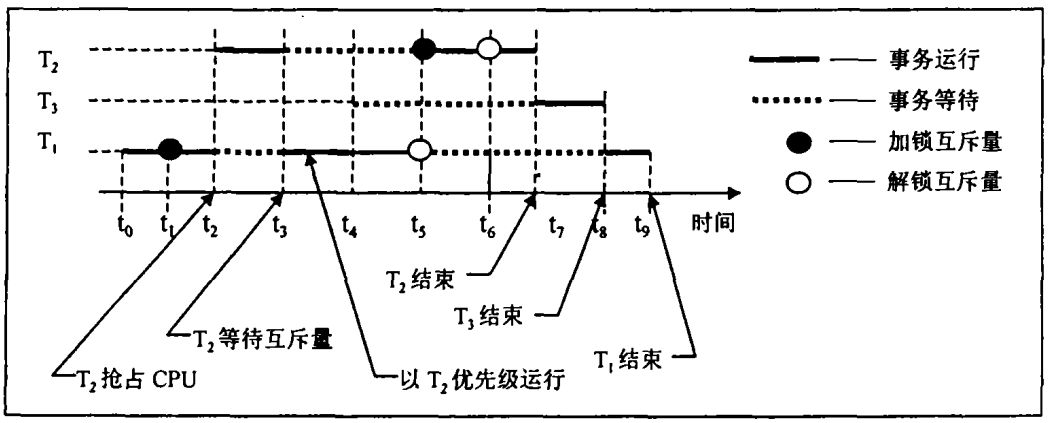


图2 采用优先级继承避免优先级倒置

2.2 并发控制机制

与一般的共享数据不同，数据库中的数据(包括数据字典)的存取是通过事务进行的。对于事务而言，必须保证事务的原子性和隔离性，即一个事务的所有操作要么全部完成，要么什么也没做；且在提交前其执行结果对其他事务是不可见的。因此，数据库中的数据不能简单通过互斥量加锁机制来完

请求 \ 持有	I	S	X
I	Y	Y	N
S	Y	Y	N
X	N	N	N

"I"—意向锁 "S"—共享锁 "X"—排它锁
"Y"—相容 "N"—不相容

图3 锁相容矩阵

操作 \ 对象	关系	数据段
查询	I	S
更新	I	X
插入	X	/
删除	X	/

"/"——不上锁

图4 上锁方式

成,而必须采用并发控制来实现互斥访问。对于实时数据库而言,并发控制策略除要求保证数据库一致性以外,还应避免优先级倒置的发生。典型的并发控制策略有优先级继承、优先级夭折和优先级天花板等。ARTs-EDB 是采用高优先级夭折策略来解决数据库中数据存取冲突和避免优先级倒置。

2.2.1 封锁粒度和锁相容矩阵 ARTs-EDB 是基于内存数据库实现的。由于内存数据库避免了事务执行过程中的I/O,因此大大减少了执行时间,故有人建议内存数据库采用关系级的封锁机制。其优点是可节约锁表空间(相对于数据段或元组粒度),但缺点是减低了事务并发度。ARTs-EDB 则根据内存数据库的区-段表组织方式和为避免产生幻象[6],采用两级封锁粒度的意向锁机制[7]:对于插入和删除操作,封锁粒度为关系;而对于查询和更新操作,封锁粒度为数据段(一个数据段中可能包含一个或多个元组),但还需在关系级加意向锁。图3为相容性矩阵,图4为不同数据库操作的上锁方式。

2.2.2 上锁原语和解锁原语 图5是 ARTs-EDB 所采用的锁表。为提高锁表的查找效率,分别在关系和事务上建立索引。关系上的索引用于查找属于同一关系的所有对象(Object)(包括关系本身及该关系的数据段);而事务上的索引用于查找属于同一事务的锁(Lock)。

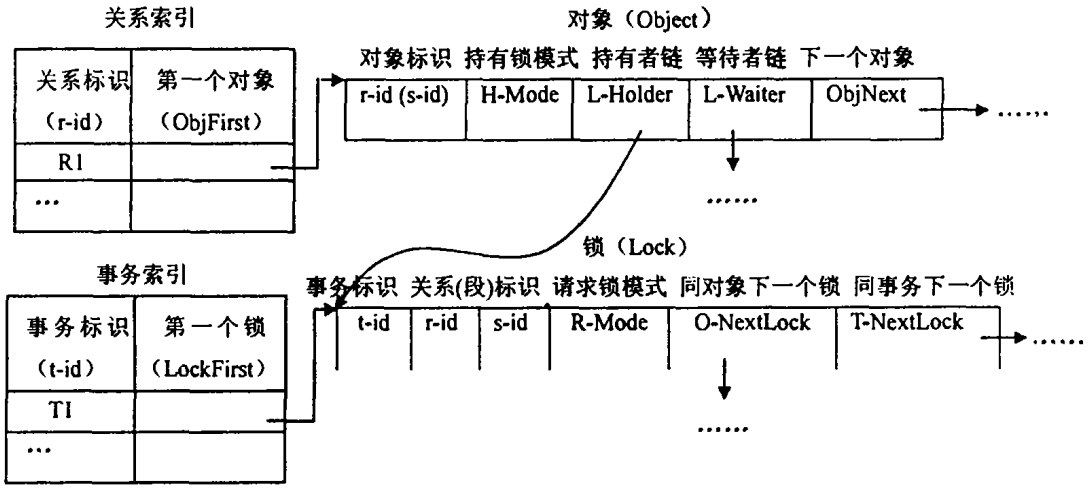


图5 锁表结构

ARTs-EDB 采用严格两阶段封锁机制,即在事务结束(提交或夭折)时释放持有的全部锁。事务的锁操作和解锁操作是通过锁管理原语 Lock 和 Unlock 实现的。Lock 是在事务存取数据时调用,而 Unlock 是在事务提交或夭折时调用(一次释放事务占有的全部锁)。另外,Lock 和 Unlock 都要求对锁表"互斥地"操作,因此,在进行操作前,需对锁表上优先级相关互斥量(lock_table_mutex),待操作完成后解锁。

加锁算法 Lock()

```
输入:t-id(事务 T 标识),object(对象),R-Mode(请求锁模式)
输出:0(失败),1(成功)
算法:
步骤1:creating a lock for T; /* 创建锁
步骤2:pthread_lock(&lock_table_mutex); /* 锁表互斥量加锁
步骤3:if R-Mode is compatible with the H-Mode of the object;
        /* 封锁对象请求锁与持有锁相容
        adding a lock in the L-Holder list of the object; /* 锁加入
        持有锁队列
      else /* 锁冲突
        if p(T) > max(p (holder)) /* T 优先级大于所有持有锁
          事务优先级
          aborting all holder transactions in L-Holder list; /* 夭折
          所有持有锁事务
          adding a lock in the L-Holder list of the object;
        else
          if p(T) < max(p (holder)) /* T 优先级小于持有锁事
            务最高优先级
            adding a lock in the L-Waiter list of the object; /* 锁
            加入等待锁队列
          else /* T 优先级与所有持有锁事务优先级相同
            if no deadlock occurs /* 无死锁发生
              adding a lock in the L-Waiter list of the object;
            else /* 发生死锁
              aborting T /* 夭折 T
步骤4:pthread_unlock(&lock_table_mutex); /* 锁表互斥量解锁
```

解锁算法 Unlock()

```
输入:t-id
输出:0(失败),1(成功)
算法:
步骤1:pthread_lock(&lock_table_mutex); /* 锁表互斥量加锁
步骤2:for every lock of T /* 对 T 的每一个锁
        removing it from the object list where it resides; /* 从所
        在的队列中删除
        if the L-Holder list is NULL /* 对象的持有锁队列为空
          if the L-Waiter list is not NULL /* 对象的等待锁队列
            不为空
            unsuspending all transactions in the L-Waiter list
            /* 解挂所有等待锁队列中的事务
          else
            removing the object/* 删除对象
        else /*对象的持有锁队列不为空
          if MAX(p(holder) < MAX(p (waiter)) /* 持有锁最高
            优先级小于等待锁最高优先级
            aborting all transactions in the L-Holder list; /* 夭折
            持有锁队列中事务
            unsuspending all transactions in the L-Waiter list;
            /* 解挂所有等待锁队列中的事务
步骤3:pthread_unlock(&lock_table_mutex); /* 锁表互斥量解锁
```

3 实时事务的同步

传统数据库事务具有隔离性，即事务提交前其执行结果对其它事务是不可见的。而且，传统数据库事务执行成功与否与其它事务无关，在执行过程中也无须与其它事务通信。因此，也就不存在所谓的事务同步。然而，实时事务往往是嵌套结构事务，嵌套事务内部事务在执行过程中要求互相等待与互通消息，从而需提供同步机制来保证嵌套事务内部一致性。

嵌套事务内部事务的同步来源于下列要求：

3.1 父子事务之间的执行依赖性

嵌套事务内部事务最基本的依赖关系有提交依赖和夭折依赖[8]：

定义1(提交依赖) 若只有当 T_i 提交后 T_j 才能提交，则称事务 T_j 存在对 T_i 的提交依赖，记为 $T_j\ CD\ T_i$。

定义2(夭折依赖) 若 T_i 夭折，T_j 必须夭折，则称事务 T_j 存在对 T_i 的夭折依赖，记为 $T_j\ AD\ T_i$。

不考虑子事务的可补偿性和关键性，一般嵌套结构事务存在下列依赖关系：

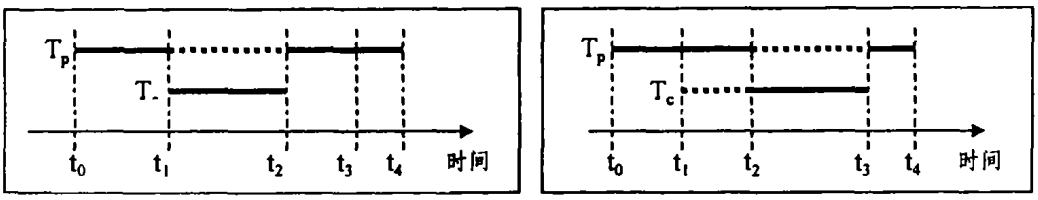
(a)父事务提交依赖于所有子事务

(b)父子彼此之间存在夭折依赖

这就要求父事务须等待其所有子事务提交后才能提交；且父子事务夭折都要通知对方。

3.2 子事务的执行模式(立即执行/推迟执行)

立即执行是指子事务被触发后立即执行，而推迟执行是指子事务被触发后等待父事务完成后再开始执行。图6为两种模式下事务的同步要求，其中 T_p 为父事务，T_c 为子事务，时刻 t_0 为 T_p 的开始点，t_1 为 T_c 的被触发时间，t_3 为 T_p 的开始提交点。



(a) 立即执行的子事务　　(b) 推迟执行的子事务

图6

在图6(a)中，T_c 在 t_1 时刻触发后立即执行，T_p 等待 T_c 执行完成后(t_2 时刻)再开始执行。而在图6(b)中 T_c 虽然在 t_1 时刻触发，但仍然要等到 T_p 完成后(t_2 时刻)再开始执行。

对于推迟执行子事务的同步，是通过在父事务中增加"END"管理原语实现(图7)。"END"是指父事务已完成所有数据库操作，但由于需等待子事务的执行结束才能提交，因此，当父事务执行到 END 原语时，检查其是否有推迟执行的子事务，若有，则通知执行，并等待；若没有，则执行提交原语。

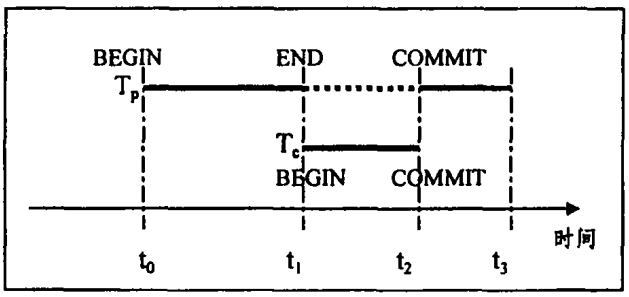


图7 利用 END 原语实现子事务的推迟执行

而对于父子事务之间依赖关系以及立即执行子事务的同步可通过设置条件变量、信号量以及线程间的通信机制(如管道)等技术实现，在此就不再赘述。

结论 与传统数据库不同，实时数据库中事务的互斥除保证共享数据的一致性外，还需考虑事务的优先级要求，即避免优先级倒置。因此，其互斥机制比传统数据库互斥机制复杂。对于采用互斥量实现的互斥方式需要实时操作系统的支持，即提供"优先级相关"互斥量。而对于采用并发控制实现的互斥，也必须是"识时"的，本文是采用高优先级夭折策略解决数据库数据的存取冲突。实时事务的同步是为满足嵌套事务的特殊要求而设计的，这是传统平坦事务模型所不需要的，从而提高了实时事务执行的灵活性。

参 考 文 献

1 刘云生著. 现代数据库技术. 北京：国防工业出版社，2001,3

2 刘云生，Ramamritham K，Stankovic J. 关于实时数据库事务. 软件学报，1995，6(10)：614～621

3 Haritsa J R，Ramamritham K. Real-time database systems in the new millennium. Real-time Systems，2000,19(3)：205～208

4 刘云生，胡国玲. ARTs-I：一个主动实时内存数据库系统. 华中理工大学学报，1996,24(3)：31～34

5 Butenhof (著)D R. 丁磊，曾刚译. POSIX 多线程程序设计. 北京：中国电力出版社，2003,4

6 Hector G M，Jeffrey D U，Jennifer W. Database system implement. 北京：机械工业出版社，2002,2

7 Bernstein P A，Hadzilacos V，Goodman N. Concurrency control and recovery in database systems. Addison Wesley，Reading，Massachusetts，1987

8 Chrysanthis P K，Ramamritham K. Synthesis of Extended Transaction Models Using ACTA. ACM Transactions on Database Systems，1994,19(3)：450～491