

## 基于事务截止期的动态多粒度封锁机制

赵跃华, 韩少聪

(江苏大学 计算机科学与通信工程学院, 江苏 镇江 212013)

(ugene@126.com)

**摘 要:**多粒度封锁机制提高了数据库系统的并发性,但在实时数据库中,由于事务处理的限时特性,对传统的多粒度封锁机制提出了挑战。综合考虑多粒度封锁机制与事务的截止期,提出了一种基于事务截止期的动态多粒度封锁机制,由事务截止期确定事务冲突级别,当事务冲突到达或降到一定级别后,根据当前加锁粒度决定是否对其进行调整。通过仿真实验证明本机制可减少事务冲突,降低事务错失率及事务重启率,提高了事务并发性和实时性。

**关键词:**动态;多粒度;封锁;事务截止期

**中图分类号:** TP311.13 **文献标志码:** A

## Dynamic multi-granularity locking mechanism based on transaction deadline

ZHAO Yue-hua, HAN Shao-cong

(College of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang Jiangsu 212013, China)

**Abstract:** Multi-granularity locking mechanism improves the concurrency of the database system. But in real-time database, the traditional multi-granularity locking mechanism meets serious challenge from the real-time demand of its transaction processing. In this paper, considering multi-granularity locking mechanism and the transaction deadline, a dynamic multi-granularity locking mechanism based on transaction deadline was proposed. The transaction conflict level was determined by transaction deadline, and when the conflicts reached or dropped to a certain level, the locking granularity was adjusted according to current locking granularity. The simulation results show that this mechanism can not only reduce transaction conflicts, but also reduce the miss ratio and restart ratio of transaction, thus improving transaction concurrency and real-time.

**Key words:** dynamic; multi-granularity; lock; transaction deadline

### 0 引言

为了使多个事务能够在共享数据库中并发操作,需要通过并发控制协议协调这些行为,保证数据库的一致性。由于封锁机制在传统数据库系统中得到广泛的应用,经过了很多系统的检验,有很强的健壮性,因此在实时数据库中封锁机制也有很广泛的应用。目前在这方面已经有许多较为成熟的研究<sup>[1-5]</sup>。文献[1]设计了一种多粒度锁机制,引入加锁检测模块,在避免死锁和提高事务并发性上有很大改善;文献[2]采用了一种动态多粒度锁的管理机制,通过在写锁与读锁之间引入一种有序共享关系来消除阻塞,提高事务执行的并发度;文献[4]改进了传统两段锁协议,解决了读写冲突。但是在实时数据库中的事务必须能够“识时”,也就是实时事务有定时性或显示的定时限制,由于并发控制可能使一些实时事务因超过其截止期造成事务提交失败,无论对于读取的事务还是整个系统都产生严重的影响,因此在设计并发控制机制时必须考虑事务的截止期<sup>[6]</sup>。针对这种情况,本文提出了一种基于事务截止期的动态多粒度封锁机制(Deadline-based Dynamic Multi-granularity Locking, DDML),根据事务的截止期动态地调整封锁粒度。

### 1 多粒度封锁机制

#### 1.1 锁模型

封锁方法的基本原则是:事务对任何数据对象操作之前

必须先申请该数据对象的锁,申请成功后才可以对其进行相关操作,操作完成后要释放其拥有的所有锁。通常锁分为读锁和写锁两种,可以有多个事务对数据对象进行读操作,因此读锁也叫共享锁,而事务对数据对象的写操作是冲突的,因此写锁也叫排它锁<sup>[7]</sup>。读锁和写锁的兼容矩阵如表1所示。

表1 读锁和写锁兼容矩阵

操作	读锁	写锁
读锁	兼容	冲突
写锁	冲突	冲突

封锁模式有很多方法,但基本原理都是一样的:当事务之间出现冲突时就通过封锁实现串行化,当锁兼容时,事务并行执行实现并行化。

#### 1.2 锁粒度

锁粒度是指加锁的数据项的大小。在传统数据库系统中,常见的锁粒度有行、表、库等,锁定的对象可以是整个数据库、一张数据表、一个页面或一条记录,甚至是一条记录中的某个字段。封锁粒度与系统的并发性和并发控制的开销有密切的关系。封锁粒度越大,被封锁的数据项就越少,并发性就越小,系统开销也越小;相反,封锁粒度越小,并发度也越高,系统开销也越大<sup>[8]</sup>。因此在同一个系统中同时支持多种粒度的封锁是比较理想的,从而实现多粒度封锁。

收稿日期:2011-06-20;修回日期:2011-08-02。

作者简介:赵跃华(1958-),男,江苏苏州人,教授,博士,主要研究方向:信息安全、嵌入式系统;韩少聪(1983-),男,山东潍坊人,硕士研究生,主要研究方向:嵌入式系统。

### 1.3 封锁粒度的选择

在嵌入式系统中,系统资源非常珍贵,作为实时数据库,其内存开销也是一个需要考虑的因素。如果加锁粒度为整个数据库,虽然系统开销降低了,但是事务并发性必然会很低,随着嵌入式技术的发展,使得在实时数据库中实施细粒度的封锁成为可能;如果封锁粒度是一条记录或者记录中的字段,事务的并发性会很高,但系统会消耗大量的开销来进行锁的管理和维护,嵌入式系统毕竟不是普通的 PC,受其软硬件资源的限制,太细的封锁粒度对资源本来就很紧张的嵌入式系统来说得不偿失。综合考虑,本文的“多粒度”指的是库粒度和表粒度两种锁粒度。

## 2 实时事务的特征

由于实时任务往往有内部结构和相互之间的联系,实时数据库的事务表现出不同的特征。实时数据库的根本特点在于数据和事务的定时性<sup>[9]</sup>,定时性主要包括两个方面的含义:1)定时限制,即实时事务的执行有显式的时间限制,例如事务的开始时间、截止时间等;2)定时正确性,即事务能按照指定的时间要求正确执行。

实时事务具有不同的定时限制,其中最重要的有:

1)截止时间。指实时事务完成的最后期限,是实时事务最本质的特征<sup>[10]</sup>。具有硬截止期的事务称为硬实时事务,它必须在其截止期内完成,否则将带来灾难性的后果,达到截止期还不能完成的硬实时事务必须夭折;具有软截止期的事务称为软实时事务,应在其截止期内完成,但超过其截止期还有一定的意义,到达截止期还不能完成的软实时事务不必立即夭折。

2)到达时间。指事务在系统中生成的时间,可以是可预报的,也可以是不可预报的。

3)期望执行时间。指估算的最坏执行时间,但由于各种不可预报的因素,它很难做到准确。

对于实时事务的执行应该考虑事务的截止期,尤其是硬实时事务,如果事务在其截止期内不能完成可能造成严重的后果。

## 3 事务的相关描述

对于实时事务  $T$  的属性作如下定义:  $dlt(T)$  为事务在  $t$  时刻访问数据时数据的截止期;  $td(T)$  为事务的截止期;  $ta(T)$  为事务  $T$  的到达时间;  $ts(T)$  为事务  $T$  的开始时间;  $rt(T)$  为事务在  $t$  时刻已执行的时间;  $ct(T)$  为事务  $T$  在  $t$  时刻估计其完成的时间。为描述事务的定时特性,本文采用事务控制块(Transaction Control Block, TCB)来记录与事务有关的一些信息,它随事务的创建而创建,随事务的夭折或提交而撤销。其数据结构如下:

```
typedef struct TCB{
    U_ID;                /* 用户 ID */
    T_ID;                /* 事务 ID */
    TYPE T_type;         /* 事务类型(硬实时,软实时) */
    int dlt;              /* 数据截止期 */
    int td;               /* 事务截止期 */
    int ta;               /* 事务到达时间 */
    int ts;               /* 事务开始时间 */
    int rt;               /* t时刻事务已执行的时间 */
    int ct;               /* t时刻估计事务完成时间 */
    int T_priority;       /* 事务优先级 */
}
```

```
struct TCB * Next;      /* 下一个事务控制块 */
...
} TCB;
```

对于就绪的事务按照优先级从高到低进行组织,其组织结构如图 1 所示。

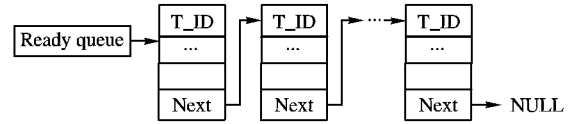


图 1 事务队列

## 4 基于事务截止期的调度

目前针对事务截止期调度的研究主要集中在三种:截止期最早最优先,可达截止期最早最优先及剩余时间最短最优先<sup>[11]</sup>。

### 4.1 截止期最早最优先

该策略是使“截止期”最早的事务具有最高的优先级,它使得最需要处理(截止期最短)的事务优先获得系统资源,但是没有考虑事务处理本身所花费的时间,因此该策略可能将高优先级分配给一个已经过时的事务,从而使那些有机会在截止期内完成的事务推迟。对每个事务  $T$ ,其截止期定义如下:

$$td(T) = (ct(T) - rt(T)) \times SF + ta(T) \quad (1)$$

其中  $SF$  是松弛因子,  $ct(t)$ 、 $rt(T)$  和  $ta(T)$  的定义如前。

### 4.2 可达截止期最早最优先

定义一个事务  $T$  当前是“可到达”的,其计算表达式如下:

$$t + (ct(T) - rt(T)) \leq td(T) \quad (2)$$

其中  $t$  是当前时间,  $ct(T)$ 、 $rt(T)$  及  $td(T)$  的定义如前。该策略通过对事务已运行时间来判断其是否是“可达”的,该策略综合考虑了事务截止期和事务运行的时间,是对第一种策略的改进。

### 4.3 剩余时间最短最优先

对于一个事务  $T$ ,其剩余时间  $st$  计算表达式如下:

$$st(T) = td(T) - (t + ct(T) - rt(T)) \quad (3)$$

该策略考虑的是事务截止期及系统的运行时间,对正在执行的事务,如果  $st \geq 0$ ,说明事务可在截止期前完成,那么  $st$  不变,优先级不变;如果  $st < 0$  说明事务处理已经或将要超过截止期。

因此,实时数据库的并发控制机制中必须考虑实时事务的截止期,针对传统多粒度封锁机制中没有考虑事务的“识时”特性,本文提出一种基于事务截止期的动态多粒度封锁机制。

## 5 基于事务截止期的动态多粒度封锁机制

事务进入队列后需要填充事务控制块 TCB 的相关信息。基于事务截止期的动态多粒度封锁机制根据应用环境设置一个阈值  $c$ ,根据事务的剩余时间  $st(T)$  与  $c$  的比较结果来提升或降低锁粒度的大小。在该机制中,为每个数据库维护一个库锁信息表,为每个表维护一个表锁信息表,分别记录数据库及表当前持有的锁信息。系统初始化时默认的锁粒度为库级,事务申请锁时,首先查看锁是否冲突,然后查看锁锁粒度状态,再根据事务剩余时间  $st(T)$  与阈值  $c$  的比较结果(事务冲突是否严重),确定是否改变锁粒度,并比较事务的剩余时间(即事务的优先级)确定是否抢占事务。其动态多粒度锁处理流

程如图2所示。

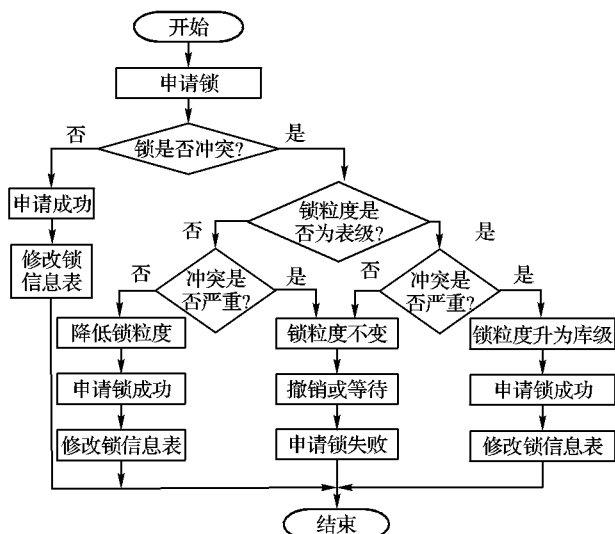


图2 动态多粒度锁处理流程

设  $T = \{t_1, t_2, t_3, \dots\}$  是事务集合,  $R(t_i, x)$  表示事务  $t_i$  对对象  $x$  加读锁,  $W(t_i, x)$  表示事务  $t_i$  对对象  $x$  加写锁,  $st(t_i)$  表示事务  $t_i$  的剩余时间。该机制具体描述如下。

1) 事务  $t_i$  执行时申请对对象  $x$  加锁, 事务终止 (提交或夭折) 后释放其所有的锁。

2) 如果事务  $t_j$  对对象  $x$  拥有读锁, 事务  $t_i$  申请对  $x$  的读锁, 则申请通过。

3) 如果事务  $t_j$  拥有对对象  $x$  上的写锁, 事务  $t_i$  申请对  $x$  上的读锁, 则按如下规则进行。

a) 判断对象  $x$  的加锁粒度是库还是表。

b) 如果加锁粒度是库, 那么比较  $st(t_i)$  与阈值  $c$  的大小, 如果  $st(t_i) > c$ , 说明事务  $t_i$  能在其剩余时间内完成, 事务冲突不严重, 则降低锁粒度为表, 同时修改库锁信息表及表锁信息表; 相反, 如果  $st(t_i) < c$ , 事务冲突严重, 则不能降低锁的粒度, 比较  $st(t_i)$  与  $st(t_j)$ , 如果  $st(t_i) > st(t_j)$ , 则  $st(t_i)$  抢占事务  $t_j$ , 反之,  $t_i$  等待  $t_j$ 。

c) 如果加锁粒度是表, 那么比较  $st(t_i)$  与阈值  $c$  的大小, 如果  $st(t_i) > c$ , 说明  $t_i$  能在剩余时间内完成, 事务冲突不严重, 无需更改锁粒度,  $t_i$  等待  $t_j$  完成; 如果  $st(t_i) < c$ , 事务冲突严重, 需将加锁粒度升级为库级, 同时修改库锁信息表及表锁信息表, 然后比较  $st(t_i)$  与  $st(t_j)$ , 如果  $st(t_i) > st(t_j)$ , 则  $st(t_i)$  抢占事务  $t_j$ , 反之,  $t_i$  等待  $t_j$ 。

4) 如果事务  $t_j$  拥有对对象  $x$  上的读锁, 事务  $t_i$  申请对  $x$  上的写锁, 其进行规则与 3) 类似。

5) 如果事务  $t_j$  拥有对对象  $x$  上的写锁, 事务  $t_i$  申请对  $x$  上的写锁, 其进行规则与 3) 类似。

## 6 性能分析

为了评估 DDML 的性能, 实验选用事务错失率 (Miss Rate, MR) 和事务重启率 (Restart Rate, RR) 作为评估 DDML 的性能指标。其中  $MR = N_{\text{missed}} / N_{\text{total}} \times 100\%$ ,  $N_{\text{missed}}$  表示超截止期的事务数量,  $N_{\text{total}}$  表示事务总数;  $RR = N_{\text{restarted}} / N_{\text{total}} \times 100\%$ ,  $N_{\text{restarted}}$  表示重启的事务数量,  $N_{\text{total}}$  表示事务总数。在不同事务时间间隔情况下分别对高优先级两阶段锁 HP-2PL (High Priority Two-Phase Locking)、传统动态多粒度封锁 DML (Dynamic Multi-granularity Locking) 及 DDML 的事务错失率进行了测试, 以及在不同事务到达率情况下对 HP-2PL、DML 及

DDML 的事务重启率进行了测试。因为在实时数据库中, 必须确保硬实时事务在其截止期内完成, 否则可能对系统造成严重的影响甚至是灾难性后果<sup>[12]</sup>。而软实时事务和非实时事务超过期截止期对系统影响不大, 因此本实验的事务类型为硬实时事务。

### 6.1 实验环境及参数

实验基于 SIMPACK 工具<sup>[13]</sup>, 通过 C++ 编程实现, 具体实验参数如表 2。

表2 实验参数

参数	值
事务到达率	每秒 30 ~ 330 个事务, 间隔为每秒 30 个事务
阈值 $c$ 的取值	0.1
松弛因子 $SF$	3
到达时间间隔	5 ~ 10 ms, 间隔 1 ms
每次测试时间	1000 s
测试次数	50

### 6.2 实验结果

通过多次实验对 HP-2PL、DML 及 DDML 的事务错失率和事务重启率性能进行测试比较, 平均后的实验测试结果如图 3 和图 4 所示。

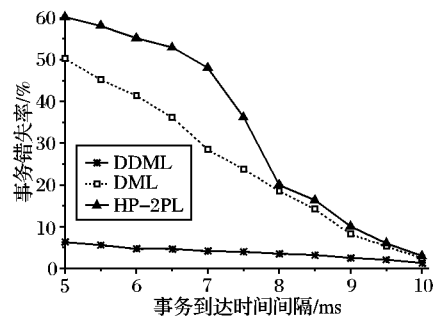


图3 事务错失率 MR

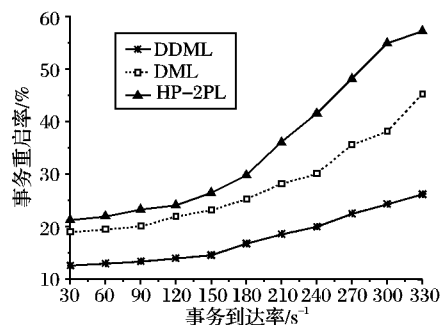


图4 事务重启率 RR

图 3 描述了事务不同到达时间间隔下的事务错失率。随着事务到达时间间隔增大, 事务错失率不断降低, 当间隔时间小于 8 ms 时, HP-2PL 与 DML 事务错失率出现明显差别, 这是因为 DML 动态调整加锁粒度, 提高事务并发度, 但因为没有考虑事务截止期, 因此仍有较高的事务错失率。但是 DDML 一直保持较低的事务错失率, 这是因为 DDML 在动态调整加锁粒度的基础上考虑了事务截止期, 有较低的事务错失率。

图 4 描述了不同事务到达率情况下的事务重启率。结果显示 DDML 的表现优于 DML 和 HP-2PL。这是因为当事务发生冲突时 DDML 能动态调整加锁粒度, 减少事务重启, 而不像 DML 采用固定加锁粒度, 也不像 HP-2PL 简单地采用高优先级重启低优先级事务。

(下转第 3284 页)

2 的删除是发生在上次同步后,服务端组件中的同步适配器会以取增量删除的方式下载该删除记录至客户端,然后在客户端删除 ID = 2 的记录。

3) 服务端更新数据: update Product set Price = 650 where ID = 3, 在客户端删除数据: delete Product where ID = 3, 导致 ClientDeleteServerUpdate 冲突。若以服务端优先处理冲突,则服务端和客户端均为服务端的更新结果(3, Phone, 650), 结果见表 5。若以客户端优先,则 ID = 3 的记录在服务端及客户端均被删除,结果见表 6。

4) 服务端更新数据: update Product set Price = 2100 Where ID = 1, 在客户端更新数据: update Product set Price = 2200 where ID = 1, 导致 ClientUpdateServerUpdate 冲突, 以 ClientInsertServerInsert 相同方式处理冲突。

表4 初始状态

服务端			客户端		
ID	Name	Price	ID	Name	Price
1	PC	2000	1	PC	2000
2	Laptop	4000	2	Laptop	4000
3	Phone	400	3	Phone	400

表5 服务端优先处理结果

服务端			客户端		
ID	Name	Price	ID	Name	Price
1	PC	2100	1	PC	2100
3	Phone	650	3	Phone	650
4	Icebox	1500	4	Icebox	1500

表6 客户端优先处理结果

服务端			客户端		
ID	Name	Price	ID	Name	Price
1	PC	2200	1	PC	2200
2	Laptop	4500	—	—	—
4	Washer	1600	4	Washer	1600

实际应用中,可根据业务逻辑及具体应用场景选择冲突处理方式,解决数据同步过程中产生的冲突。

### 3 结语

随着移动设备的硬件性能的不不断提高,基于桌面系统的数据库应用程序正越来越多地移植到移动设备中,移动办公在给人们带来方便的同时,必须要解决的问题就是如何实现移动设备客户端与服务端的数据同步。本文结合 WCF、同步服务等相关技术,给出了一个基于 WCF 的分布式移动数据同步解决方案。在银行基金代销系统中,基金代销人员需要经常上门拜访客户,因此在必要时需将服务器中的数据下载到移动设备,以便向客户介绍基金的情况,若基金销售成功,又需要将记录在本地数据库的基金销售信息在网络可用时同步到服务器。应用本方案成功解决了基金代销系统的数据同步问题,并取得了良好的效果。

#### 参考文献:

- [1] 王文琴, 费贤举, 鞠时光. 基于数据复制技术实现移动数据同步[J]. 计算机应用, 2006, 26(7): 1676-1678.
- [2] 索红光, 王雷全. 智能客户端系统中数据同步策略的研究与实现[J]. 计算机工程与设计, 2007, 28(2): 351-353.
- [3] 李艳, 蓝雯飞. 移动数据库中数据同步技术[J]. 重庆科技学院学报: 自然科学版, 2008, 10(6): 112-114.
- [4] 高艳宏, 马光恩. 基于 Sync Services for ADO.NET 的智能客户端应用研究[J]. 计算机技术与发展, 2010, 20(10): 224-227.
- [5] 严商, 黄樟灿. WCF: Windows 平台新一代通信基础研究与分析[J]. 计算机与数字工程, 2008, 36(4): 86-89.
- [6] 韩旭, 王海波, 柳克俊. 基于 .NET Framework WCF 的面向服务 SOA 中间件设计[J]. 小型微型计算机系统, 2010, 31(12): 2359-2363.
- [7] 吴莉. 基于 .NET 框架的 N 层分布式应用程序研究[J]. 贵州工业大学学报: 自然科学版, 2008, 37(4): 207-210.
- [8] 杨志和, 胡虚怀. 移动环境下的数据同步模型研究[J]. 计算机工程与应用, 2007, 43(13): 191-193.

(上接第 3280 页)

### 7 结语

传统的多粒度封锁机制能提高系统的并发性,但是在实时数据库中的事务具有“识时”的特性,如果直接将多粒度锁机制用在实时数据库中而不考虑事物的“识时”特性,那么必然会降低系统的实时性。本文结合实时数据库中事务的特性,提出了基于事务截止期的动态多粒度封锁机制,通过实验说明能降低事务的错失率及事务重启率。

#### 参考文献:

- [1] 王劲波, 薛永生, 徐助民. 一种基于加锁粒度的分布式高优先级两段锁的并发控制模型[J]. 计算机应用与软件, 2003, 20(6): 16-18.
- [2] 张永华, 张芳, 宋菁. 一种基于内存数据库的并发控制协议研究[J]. 微计算机信息, 2007, 23(3): 170-172.
- [3] 廖国琼, 刘云生, 王洪庭. 嵌入式实时数据库事务的并发控制[J]. 计算机工程, 2005, 31(9): 27-28, 150.
- [4] 熊燕群, 李进京. 适用于实时数据库系统的并发控制协议[J]. 计算机工程与设计, 2009, 30(3): 631-633.
- [5] LAM K Y, KUO TEI-WEI, TSANG W H, et al. Concurrency control in mobile distributed real-time database systems[J]. Information

Systems, 2000, 25(4): 261-286.

- [6] 韩启龙, 郝忠孝. 基于数据截止期的并发控制策略[J]. 哈尔滨工业大学学报, 2008, 40(2): 259-262.
- [7] 马淑娇, 李晓, 周俊. 2PL 并发控制的研究与实现探析[J]. 计算机应用研究, 2003, 20(1): 38-40.
- [8] 华中科技大学. 一种数据库中动态多粒度锁的事务冲突判决方法: 中国, 200910305845[P]. 2010-03-10.
- [9] 陈传波, 王桦. 实时数据库的事务调度研究[J]. 计算机应用, 2005, 25(9): 2004-2006.
- [10] 刘云生, 李国徽, 卢炎生. 实时数据库的事务处理[EB/OL]. [2010-05-19]. <http://www.cnblogs.com/jin20000/archive/2011/05/19/2051490.html>.
- [11] 蔡振, 蔡世洪, 申志强. 实时数据库的事务调度策略分析[J]. 计算机与数字工程, 2007, 35(12): 65-67.
- [12] ZHAO YUEHUA, QIU JING. A new multi-dynamic priority real-time database scheduling algorithm[C]// Proceedings of the 2010 ICCET international conference on Computer Engineering and Technology. New York: IEEE, 2010: 177-180.
- [13] FISHWICK A. SIMPACK: Getting started with simulation programming in C and C++, TR92-022[R]. Gainesville: University of Florida, Department of Computer and Information Sciences, 1992.