实时事务调度及静态可调度性分析*)

许贵平　刘云生
(华中科技大学计算机科学与技术学院　武汉 430074)

摘　要　在类似闭环控制的硬实时数据库应用环境,实时事务具有一定的静态可预报性,其中实时事务的可调度性分析是维护实时数据库时间正确性的基础。通过利用抢占阈值,提出了一种新的实时事务处理模型,它集成了CPU调度和数据调度,实现离线并发控制,具有单阻塞的特征与好的静态可预测性,并有利于降低事务系统的负载和改善可调度性。进一步由此建立了实时事务的静态可调度性分析模型以及求最优可行调度的整数规划模型,该模型有利于达到实时事务调度的整体优化。

关键词　实时数据库,实时事务调度,可调度性分析

Real-time Transaction Scheduling and Schedulability Analysis

XU Gui-Ping　LIU Yun-Sheng
(Schooll of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　In hard real-time database applications for closed loop control environments, real-time transactions may have some predictabilities, and schedulability analysis is critical to maintaining their time correctness. This paper proposes an integrated real-time transaction-processing model by use of preemption threshold. Its special off-line concurrency control can bound priority-inversion time, and may reduce the run-time overhead and improve schedulability. Based on schedulability analysis, this paper also presents the conditions of feasible transaction scheduling for given real-time transaction set, and an integer programming model for constructing optimal scheduling, which is of benefit to the whole optimization of scheduling.

Keywords　Real-time database, Real-time transaction schedulaling , Schedulability analysis

1　引言

实时数据库事务处理不仅要维护事务的逻辑一致性,而且要保证实时事务的定时正确性,典型地应确保实时事务满足其截止期,即系统应保证实时事务的可调度性。而实现这一任务的根本在于实时事务的可调度性分析,但是在文献中鲜见有这一方面的研究成果,本文针对相对静态的硬实时计算环境对此进行了研究。

实时数据库事务一般执行在应用任务的执行线程内。实时事务处理主要采用基于优先级可抢占的事务调度策略和基于实时锁的并发控制协议[1],事务的抢占最终导致实时任务或者线程的切换,从而增加了系统的开销,影响了实时事务处理的性能,并有可能降低实时事务的可调度性。

为了实现与改善实时事务的静态可调度性,本文利用抢占阈值概念[2~4],提出了一种新的实时事务处理模型,它集成了CPU调度和数据调度,实现了基于冲突避免的离线实时事务并发控制;在此基础上,建立了一个实时事务静态可调度性分析模型,对于给定的实时事务集提出了可行调度应满足的条件,并提出了建立最优可行调度的整数规划模型 。

2　实时事务调度的抢占

为了支持系统的可预测性与实时性,实时数据库系统采用内存数据库管理,从而避免了在事务运行过程中动态请页与I/O所引起的不可预测性。在类似闭环控制的硬实时应用环境,事务对数据库的存取一般以封装好的固定程序(Canned transaction)出现。因此,通过对实时事务集的静态预分析[5,6],能够提取事务的存取数据集、事务间的冲突关系和其它语义知识,利用这些信息,对事务进行无冲突调度,消除了系统运行时的并发控制负载,避免了事务的夭折和重起,保证了实时事务执行的可预报性。

假定实时数据库系统由 N 个固定实时事务构成事务集合 Γ:

$\Gamma=\{t_1,t_2,\cdots,t_N\}$

定义 1　Γ 中任一事务 t_i 定义为一个5元组:

$\langle C_i,\ T_i,\ D_i,read\text{-}set(t_i),write\text{-}set(t_i)\rangle$,

其中,C_i 表示事务 t_i 的最坏执行时间;T_i 表示事务 t_i 的周期,若 t_i 为非周期事务,则 T_i 表示事务 t_i 的两个连续事务实例的最小到达时间间隔;D_i 是 t_i 的相对截止期;$read\text{-}set(t_i)$、$write\text{-}set(t_i)$分别表示 t_i 的读数据集和写数据集。

对 Γ 中任一事务 t_i,为其分配一个数偶$\langle p_i,\pi_i\rangle$,p_i、π_i 分别是 t_i 的优先级和抢占阈值,并且有 $p_i\in[1,2,\cdots,N]$,$\pi_i\in[p_i,\cdots,N]$,$p_i\leqslant\pi_i\leqslant N$,即任一事务的抢占阈值不小于其优先级。约定 t_i 的 p_i 值越大其优先级越高,且没有两个事务具有相同的优先级。

*)基金项目:国家自然科学基金资助项目(60073045)。许贵平　博士,副教授,研究方向:现代数据库理论与技术及其集成实现、嵌入式实时系统。刘云生　教授,博士生导师,研究方向: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。

把周期事务在其一个周期内的激活称为一个事务实例，在下文中，除特别指明，对事务与事务实例不加区别。

定义 2 对于实时事务集

$\Gamma=\{t_i=\langle C_i, T_i, D_i, read\text{-}set(t_i), write\text{-}set(t_i)\rangle \mid i=1, \cdots, N\}$

定义 Γ 的调度 S 为如下 Γ 的映射：

$S: \forall t_i \rightarrow \langle p_i, \pi_i\rangle$，

其中 $\langle p_i, \pi_i\rangle \in [1,2,\cdots,N]\times[p_i,\cdots,N]$。

以上实时事务属性是在实时数据库应用设计与事务预分析阶段静态确定的，在系统运行过程中保持不变，事务优先级与抢占阈值的分派应保证事务集合 Γ 的可调度性、正确性及系统的性能需求。系统采用基于抢占阈值的固定优先级可抢占事务调度，事务在执行过程中不挂起自己，除非被其它事务所抢占。系统对并发执行的实时事务存取数据库采用冲突避免的方式，事务遵从可串行化的正确性标准。

定义 3 对 Γ 中两个不同的实时事务 t_i 与 t_j，当且仅当 $p_i>\pi_j$ 时，称 t_j 的执行可被 t_i 抢占，或 t_i 可抢占 t_j。

根据以上定义，一个事务可抢占另一事务，则其优先级一定更高，但反之不一定成立，即事务集合 Γ 具有如下性质。

性质 1 实时事务 t_i 可抢占 t_j 的必要条件是 $p_i>p_j$。

性质 2 如果 Γ 中任一事务 t_i，都有 $\pi_i=p_i$，$i=1,2,\cdots,N$，则事务集 Γ 的调度是基于优先级完全可抢占的；如果 Γ 中任一事务 t_i，都有 $\pi_i=N$，$i=1,2,\cdots,N$，则事务集 Γ 的调度是不可抢占的。

若事务集 Γ 的调度是基于优先级完全可抢占的，那么高优先级事务总是可以抢占低优先级事务的执行而取得 CPU；若事务集 Γ 的调度是不可抢占的，则任一事务的执行都不会因其它事务竞争 CPU 而被抢占。

性质 3 Γ 中两个不同的实时事务 t_i 与 t_j 是互不可抢占的充要条件是

$(p_i\leqslant\pi_j)\wedge(p_j\leqslant\pi_i)$ 或 $\max\{p_i,p_j\}\leqslant\min\{\pi_i,\pi_j\}$。

定义 4 一个活跃事务序列 $(t_{i1}\,t_{i2}\cdots t_{in})(n\geqslant 2)$，如果下列条件成立：

(1)事务 $t_{i1},\cdots,t_{in-1}$ 都已开始执行但都未结束；

(2)事务 t_{ik} 直接抢占了 t_{ik-1} 的执行，$k=2,3,\cdots,n$；

(3)事务 t_{i1} 没有抢占任何事务，而 t_{in} 直至执行结束没有被任何活跃事务所抢占；

则称这个事务序列为一个事务抢占链(Transaction preemption chain)，表示为 TPC。

一个事务抢占链随着时间推移会动态地收缩，事务抢占链具有如下两条性质。

性质 4 如果 $TPC_i=(t_{i1}\,t_{i2}\cdots t_{in})$ 为一个事务抢占链，则该链中事务的优先级与抢占阈值之间的关系为 $P_{i1}\leqslant\pi_{i1}<p_{i2}\leqslant\pi_{i2}<\cdots<p_{in-1}\leqslant\pi_{in-1}<p_{in}\leqslant\pi_{in}$。

性质 5 一个事务实例不会出现在两个不同的事务抢占链中。

定义 5 在系统运行时，如果在时刻 s 发生事务抢占而引起事务切换，则称时刻 s 为一个抢占点。

当被抢占事务重新被调度而执行结束时，对应的抢占点死亡，关于该抢占点的有关信息可以从系统中删除。

定义 6 把系统中当前时刻之前的最后一个存活的抢占点所对应的被抢占事务的抢占阈值定义为系统抢占阈值，表示为 π_{sys}。系统初启时，设置 $\pi_{sys}=0$。

随着系统中当前最近一个抢占点的产生与死亡，系统抢占阈值动态地发生相应的变化。

在这里的事务处理模型下，事务切换只能在两种情形下发生：①高优先级事务抢占了正在执行的低优先级事务；②一个事务执行完毕而转事务调度，选择一个适当的事务开始执行。根据以上定义与分析，下面给出基于抢占阈值的固定优先级可抢占事务调度的调度规则：

(1)抢占规则 事务 t_j 正在执行，此时中断发生引起高优先级事务 t_i 到达并放行，当 $p_i>\pi_j$ 时 t_i 抢占 t_j，并产生新的抢占点与新的系统抢占阈值；

(2)重调度规则 若某事务执行结束，设此时处于事务就绪队列头的事务为 t_h(t_h 是最高优先级就绪事务)，最后一个存活的抢占点对应的被抢占事务为 t_p，当 t_h 与 t_p 是两个不同的事务且 $p_h>\pi_{sys}$ 时，调度 t_h 执行；否则，重新调度 t_p 执行。

3 基于抢占阈值的并发控制协议

基于事务预分析获得 Γ 中每个实时事务的读数据集和写数据集，识别出事务间的冲突关系，再通过控制冲突事务间的相互抢占，实现冲突避免的事务调度，从而达到离线并发控制的目的。适当分配事务的抢占阈值，可以直接禁止冲突事务的相互抢占，所以能够利用事务的抢占阈值，有机地将 CPU 调度和数据调度结合起来，使实时事务的调度更灵活，具有更好的静态可预报性。由于没有使用锁式并发控制，消除了锁管理的系统开销和死锁的产生。下面讨论基于抢占阈值的并发控制协议。

定义 7 称 $access\text{-}set(t_i)=read\text{-}set(t_i)\cup write\text{-}set(t_i)$ 为事务 t_i 的存取集。

定义 8 对 Γ 中两个不同的实时事务 t_i 与 t_j，如果

$read\text{-}set(t_i)\cap write\text{-}set(t_j)\neq\varnothing$

或 $write\text{-}set(t_i)\cap access\text{-}set(t_j)\neq\varnothing$

则称 t_i 与 t_j 是冲突事务。

定义 9 定义集合 $conflict\text{-}set(t_i)=\{t \mid t\in\Gamma$，并且 t_i 与 t 是冲突事务$\}$ 为事务 t_i 的冲突事务集。

下面给出冲突避免的事务调度条件。

条件 1 Γ 中任意两个冲突事务 t_i 与 t_j 互不可抢占，即有

$(p_i\leqslant\pi_j)\wedge(p_j\leqslant\pi_i)$ 或 $\max\{p_i,p_j\}\leqslant\min\{\pi_i,\pi_j\}$。

定理 1 如果实时事务集 Γ 的一个调度 S 满足条件 1，则调度 S 是冲突避免的。

证明：假定在调度 S 的一个具体调度运行中，存在两个冲突的事务 t_i 与 t_j 在某一时间段内都始终处于活跃状态，即它们都已开始执行但还都未结束，不妨设事务 t_i 先于 t_j 开始执行，则只有下列四种可能情形：

case1 t_i 与 t_j 处在同一个事务抢占链中，由性质 4，必有 $p_j>\pi_i$，但由条件 1，这是不可能的；

case2 t_i 与 t_j 分别处在两个不同的事务抢占链 $TPC_j=(t_{j1}\,t_{j2}\cdots t_{jm})$ 与 $TPC_i=(t_{i1}\,t_{i2}\cdots t_{in-1}\,t_{in})$ 中，那么在时序上，TPC_i 先于 TPC_j 执行，由重调度规则可推得 $p_{j1}>\pi_{in-1}$，又由性质 4，$p_j\geqslant p_{j1}$ 且 $\pi_{in-1}\geqslant\pi_i$，从而 $p_j>\pi_i$，这与条件 1 矛盾。

case3 t_i 在事务抢占链 $TPC_i=(t_{i1}\,t_{i2}\cdots t_{in-1}\,t_{in})$ 中，而 t_j 不在任何事务抢占链中，那么在时序上，TPC 先于 it_j 执行，同 case2 可推得与条件 1 矛盾的结果；

case4 t_i 不在任何事务抢占链中，则 t_i 没有被任何事务抢占，并且 t_i 也没有抢占任何其它事务，即 t_i 执行完毕之后 t_j 才开始执行，显然这不符合假定的条件：t_i 与 t_j 在某一时间

段内都始终处于活跃状态。因此在假定的条件下 case4 是不可能的。

以上说明 Γ 的一个满足条件 1 调度 S 一定是冲突避免的。

定理 2 实时事务集 Γ 满足条件 1 的调度 S 是逻辑正确的。

证明:如果调度 S 满足条件 1,由定理 1,则 S 是冲突避免的,即 Γ 中任何两个冲突事务的执行是串行的;而所有非冲突事务间的执行是可串行化的,所以调度 S 是可串行化的,因此是逻辑正确的。

4 静态可调度性分析

对于实时事务集 Γ 的一个调度 S,除了考察 S 的逻辑正确性之外,还要保证 S 的定时正确性,因此必须进行系统的可调度性分析,这在此种实时数据库应用中尤为重要。为了简化讨论,约定 Γ 中任一事务 t_i,都有 $D_i \leqslant T_i$;具备抢占条件的高优先级事务在任意时刻都可以抢占正在执行中的低优先级事务而获得 CPU。应用事务的响应时间进行可调度性分析,事务 t_i 的响应时间是指 t_i 从放行到执行完成的时间间隔,记为 R_i,它包括三个部分:(1)事务 t_i 的最坏计算时间 C_i,(2)高优先级事务或 t_i 的以前放行的事务实例(instances)对 t_i 干扰时间 I_i,(3)由于抢占阈值的作用,低先级事务对 t_i 造成的阻塞时间 B_i,即有

$$R_i = C_i + I_i + B_i \tag{1}$$

在低优先级事务 t_j 的执行过程中,事务 t_i 到达并立即放行,若 $p_j < p_i \leqslant \pi_j$,则高优先级事务 t_i 被阻塞,从而发生了优先级颠倒。用 A_i、S_i、F_i 分别表示所讨论的 t_i 的事务实例的到达时间、首次获得 CPU 开始执行的时间和完成时间。

定理 3 在实时事务集 Γ 的调度 S 的一个调度执行中,事务 t_i 只能被至多一个低优先级事务 t_j 阻塞一次,且有 $p_j < p_i \leqslant \pi_j$。

证明:如果事务 t_i 被低优先级事务 t_j 所阻塞,则 t_j 一定在 t_i 到达之前已经开始执行但还未结束,并且在时间段$[A_i, S_i]$的某个子区间 Δ_1 内 t_j 占有 CPU 处于执行态,根据抢占规则和重调度规则,一定有 $p_j < p_i \leqslant \pi_j$。若 t_j 在$[A_i, F_i]$内执行结束,显然 t_i 不会被事务 t_j 的后继事务实例所阻塞,即高优先级事务不会被一个低优先级事务的两个或两个以上的事务实例所阻塞。

如果事务 t_i 同时被另一个低优先级事务 t_k 所阻塞,则 $p_k < p_i \leqslant \pi_k$,并且在时间段$[A_i, S_i]$的某个子区间 Δ_2 内 t_k 占有 CPU 处于执行态,不妨设 Δ_1 先于 Δ_2。由于 $p_k < p_i \leqslant \pi_j$,并且 $p_j < p_i \leqslant \pi_k$,因此 t_j 与 t_k 是互不可抢占的,根据定理 1,t_j 与 t_k 的执行是串行的,因而 t_j 的完成时间 F_j 一定先于 t_k 的开始时间 S_k,根据抢占规则和重调度规则,考虑到 $p_j < p_i$ 与 A_i 先于 F_j,所以 S_i 先于 S_k,即可推得 $\Delta_2 \not\subset [A_i, S_i]$,与假设矛盾。

以上证明了定理 3 是正确的。

根据定理 3,能够给出事务 t_i 可能经历的最坏阻塞时间为

$$B_i = \max\{C_j \mid P_j < P_i \leqslant \pi_j, \forall t_j \in \Gamma\} \tag{2}$$

事务 t_i 可能受到高优先级事务的干扰时间 I_i 可以分为两部分:即 t_i 在时间段$[A_i, S_i]$与$[S_i, F_i]$内被高优先级事务的干扰时间,其中在时间段$[A_i, S_i]$内,系统中活跃的高优先级事务都会延迟 t_i 的开始执行时间 S_i;在时间段$[S_i, F_i]$内,根据抢占规则和重调度规则,能够对 t_i 造成干扰的事务 t_j 必须满足:$p_j > \pi_i$。另外,低先级事务可能对 t_i 造成的阻塞只能发生在时间段$[A_i, S_i]$内,即它们只能延迟 t_i 的开始执行时间 S_i。

为了更好地表示事务 t_i 的最坏响应时间,这里给出事务 t_i 的最坏开始时间和完成时间,分别仍用 S_i,F_i 表示,不过此时它们分别表示时间区间$[A_i, S_i]$和$[A_i, F_i]$的长度而不是时刻。

$$S_i = B_i + \sum_{t_j \in hp(t_i)} (1 + \lfloor \frac{S_i}{T_j} \rfloor) C_j \tag{3}$$

其中 $hp(t_i)$表示 Γ 中所有优先级高于 p_i 的事务组成的集合。

$$F_i = S_i + C_i + \sum_{\forall t_j \in \Gamma, p_j > \pi_i} (\lceil \frac{F_i}{T_j} \rceil - (1 + \lfloor \frac{S_i}{T_j} \rfloor)) C_j \tag{4}$$

事务 t_i 的最坏响应时间为 $R_i = F_i$。为了求 R_i,可以用不动点迭代法解方程(3)和(4),例如解方程(4)的迭代公式为:

$$w_i^{n+1} = S_i + C_i + \sum_{\forall t_j \in \Gamma, p_j > \pi_i} (\lceil \frac{w_i^n}{T_j} \rceil - (1 + \lfloor \frac{S_i}{T_j} \rfloor)) C_j \tag{5}$$

取迭代初值 $W_i^0 = S_i$,当 $W_i^{n+1} = W_i^n$ 或 $W_i^{n+1} > D_i$ 迭代终止,前者表明迭代收敛,取 $R_i = W_i^n$;后者表明事务 t_i 是不可调度的。

基于事务的最坏响应时间,可以给出事务集 Γ 的一个调度 S 的可调度性条件。

条件 2 $R_i \leqslant D_i, i = 1, 2, \cdots, N$。

5 事务优先级与抢占阈值的分配

定义 9 如果实时事务集 $\Gamma = \{t_i = (C_i, T_i, D_i, read\text{-}set(t_i), write\text{-}set(t_i)) \mid i = 1, \cdots, N\}$ 的调度 S 满足条件 1 和条件 2,则称 Γ 的调度 S 是可行的。

下面把基于抢占阈值的事务调度方法应用到航空电子学的一个典型实时事务系统 Avionics Example[3,7](表 1 所给出的事务集 Γ)中,该系统有 18 个周期事务,通过分析事务的数据需求可以得到事务之间的冲突关系信息。易验证表 1 所给出 Γ 的调度 S 是满足条件 1 的,从而能够保证事务系统执行的逻辑正确性和数据一致性。通过上一节的可调度性分析,计算每一个事务的最坏响应时间 R_i,由于 $R_i \leqslant D_i, i = 1, 2, \cdots, 18$,所以该调度 S 是可行的。如何求得一个事务集的调度的可行解(如果可行解是存在的),在所有的可行解中,如何寻找其最优解?下面建立相应的最优解模型与算法。

表 1 Avionics Example 事务集 Γ(单位 ms)

	Tran. C_i	T_i	D_i	*conflict-set*	p_i	π_i	R_i
t_1	0.051	1.00	1.00		18	18	0.051
t_2	3.01	200	5.0		17	17	3.214
t_3	2.03	25	25		16	16	10.631
t_4	5.03	25	25	$\{t_{10}\}$	15	16	20.191
t_5	1.00	40	40		14	15	21.242
t_6	3.02	50	50		13	15	24.415
t_7	5.03	50	50	$\{t_{11}\}$	12	15	31.832
t_8	8.05	59	59		11	15	45.626
t_9	9.05	80	80		8	15	59.480
t_{10}	2.03	80	100	$\{t_4, t_{16}\}$	10	15	48.809
t_{11}	5.05	100	115	$\{t_7\}$	9	15	56.297
t_{12}	1.03	200	200	$\{t_{16}\}$	7	15	142.232
t_{13}	3.05	200	200		6	6	144.435
t_{14}	1.03	200	200		5	6	145.516
t_{15}	1.03	200	200		4	6	146.597
t_{16}	3.03	200	200	$\{t_{10}, t_{12}\}$	3	15	147.648
t_{17}	1.00	1000	1000		2	6	148.699
t_{18}	1.00	1000	1000		1	6	148.699

5.1 求最优可行解的整数规划模型

对于实时事务集

$\Gamma=\{t_i=\langle C_i, T_i, D_i, read\text{-}set(t_i), write\text{-}set(t_i)\rangle \mid i=1,2,\cdots,N\}$,

为了求其调度的最优可行解,下面建立相应的整数规划模型。

为此,定义目标函数为

$$G=\sum_{i=1}^{N}\frac{R_i-D_i}{C_i} \tag{6}$$

最优可行解 $S:\langle p_i,\pi_i\rangle\in[1,2,\cdots,N]\times[p_i,\cdots,N], i=1,\cdots,N$ 使目标函数 G 取最小值。整数规划模型的约束必须保证调度 S 的可行性,即要使之满足条件 1 和条件 2。

为了描述 Γ 中任意两个事务 t_i 与 t_j 的优先级、抢占阈值之间的关系,引入 $2N^2$ 个 0/1 变量 p_{ij}、q_{ij},其中 $i,j=1,\cdots,N$ 且

$p_{ij}=1$,iff $p_i<p_j$;else $p_{ij}=0$,而 $q_{ij}=1$,iff $p_i\leqslant\pi_j$;else $q_{ij}=0$。

从而事务 t_i 的最坏阻塞时间可以表示为

$$B_i=\sum_{j=1}^{N}p_{ji}q_{ij}C_j$$

因此(3)式可以写为

$$S_i=\sum_{j=1}^{N}p_{ji}q_{ij}C_j+\sum_{j=1}^{N}(1+\lfloor\frac{S_i}{T_i}\rfloor)C_j, i=1,2,\cdots,N \tag{7}$$

即

$$S_i=\sum_{j=1}^{N}(p_{ij}q_{ij}+p_{ij}(1++\lfloor\frac{S_i}{T_i}\rfloor))C_j, i=1,2,\cdots,N \tag{8}$$

(4)式可以写为

$$R_i=C_i+\sum_{j=1}^{N}(p_{ji}q_{ij}+p_{ij}(1+\lfloor\frac{S_i}{T_j}\rfloor))C_j+\sum_{j=1}^{N}(1-q_{ji})(\lceil\frac{R_i}{T_j}\rceil-(1+\lfloor\frac{S_i}{T_i}\rfloor))C_j, i=1,2,\cdots,N \tag{9}$$

进一步,令

$$\lfloor\frac{S_i}{T_i}\rfloor=e_{ij},\lceil\frac{R_i}{T_j}\rceil=f_{ij}, i,j=1,2,\cdots,N \tag{10}$$

其中 e_{ij}, f_{ij} 为非负整数,因此

$$S_i=\sum_{j=1}^{N}(p_{ji}q_{ij}+p_{ij}(1+e_{ij}))C_j, i=1,2,\cdots,N \tag{11}$$

$$R_i=C_i+\sum_{j=1}^{N}(p_{ji}q_{ij}+p_{ij}(1+e_{ji})+(1-q_{ji})(f_{ij}-(1+e_{ij}))C_j, i=1,2,\cdots,N \tag{12}$$

并且具有如下约束:

$$0\leqslant\sum_{j=1}^{N}(P_{ji}q_{ij}+p_{ij}(1+e_{ij}))C_j-e_{ij}T_j<T_j, i,j=1,2,\cdots,N \tag{13}$$

$$0\leqslant f_{ij}T_j-C_j-\sum_{j=1}^{N}(P_{ji}q_{ij}+p_{ij}(1+e_{ij})+(1-q_{ji})(f_{ij}-(1+e_{ij}))C_j<T_j, i,j=1,2,\cdots,N \tag{14}$$

另外表达优先级、抢占阈值之间的关系的 0/1 变量 p_{ij}、q_{ij} 具有如下约束条件($i,j,k=1,\cdots,N$,且 i,j,k 两两互异):

$$p_{ii}=0, q_{ii}=1 \tag{15}$$

$$p_{ij}+p_{ji}=1 \tag{16}$$

$$1-p_{ij}\leqslant(1-p_{ik})+(1-p_{kj}) \tag{17}$$

$$p_{ij}\leqslant q_{ij} \tag{18}$$

$$1-q_{ij}\leqslant(1-p_{ik})+(1-q_{kj}) \tag{19}$$

最后,可行解还必须满足条件 1 和条件 2,其中条件 2 可以转化为如下约束:

$$C_i+\sum_{j=1}^{N}(P_{ji}q_{ij}+p_{ij}(1+e_{ij})+(1-q_{ji})(f_{ij}-(1+e_{ij}))C_j\leqslant D_i, i,j=1,2,\cdots,N \tag{20}$$

而条件 1 由实时事务集 Γ 所确定,例如对 Avionics Example 事务集 Γ,条件 1 转化为约束:

$$q_{4,10}=1, q_{10,4}=1, q_{7,11}=1, q_{11,7}=1, q_{10,16}=1, q_{16,10}=1, q_{12,16}=1, q_{16,12}=1 \tag{21}$$

根据以上分析,求实时事务集 Γ 的调度的最优可行解等价于解如下约束整数规划问题:

$$\min_{p_{ij},q_{ij},e_{ij},f_{ij}} G=\sum_{j=1}^{N}(C_i+\sum_{j=1}^{N}(p_{ji}q_{ij}+p_{ij}(1+e_{ij})+(1-q_{ji})(f_{ij}-(1+e_{ij}))C_j-D_i)/C_i \tag{22}$$

其等式或不等式约束条件分别为(13)~(21),这是一个双线性约束整数规划问题,存在解这类问题的算法和商业软件。

结论 针对某种硬实时计算环境,本文深入研究了实时事务的调度与实时事务的静态可调度性分析,对于在维护数据一致性的同时保证实时事务的可调度性问题进行了一定的探索,其主要贡献有:

(1)提出了基于抢占阈值的实时事务调度与并发控制策略,建立了一种新的把 CPU 调度和数据调度高度集成的实时事务处理机制。由于采用了离线并发控制和有效地控制了事务间的抢占,该机制具有改善实时事务可调度性与降低系统负载的可能性。

(2)基于实时事务的最坏响应时间,针对本文的事务处理机制,建立了实时事务的静态可调度性分析模型。其中由于界定了实时事务优先级颠倒的时间,该模型具有一定的实用性。

(3)建立了基于整数规划的实时事务最优可行调度的计算模型。该模型使事务优先级与抢占阈值的分配更灵活,能够实现调度的整体优化。

需要进行的后续研究工作有:当不存在可行调度解时,对非关键的实时事务应如何调度而可以保证其他实时事务的静态可调度性;将本文的理论实际应用于我们所开发的实时数据库中,进行实际检测与性能验证。

参 考 文 献

1 Lui sha, et al. A real-time locking protocol. IEEE Transactions on Computers, 1991,40 (7):793~800

2 Wang Y, Saksena M. Scheduling fixed priority tasks with preemption threshold. In: Proc 6th IEEE Real-time Computing Systems and Applications Symposium, Hong Kong, China, 1999. 328~335

3 Saksena M, Wang Y. Scalable multi-tasking using preemption thresholds. In: Proc. 6th IEEE Real-Time Technology and Applications Symposium, Work-In-Progress Session, Washington D. C., 2000

4 Gai P, et al. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-chip. In: Proc 22th Real-Time Systems Symposium, London, England, 2001. 73~83

5 Liu Yunsheng, Hu Guolin, Shu Liangcai. Transaction pre-processing in real-time main memory database. Journal of software, 1997, 8 (3):204~209 (in Chinese)

6 Ulusoy O, Buchmann A. A real-time concurrency control protocol for main-memory database systems. Information Systems, 1998, 23(2):109~125

7 Kuo T W, Liang M C, Shu L C. Abort-oriented concurrency control for real-time databases. IEEE Transactions on Computers, 2001,50 (7):660~673