第 42 卷 第 6A 期
2015 年 6 月

计 算 机 科 学
Computer Science

Vol. 42 No. 6A
June 2015

基于动态决策的并发控制算法

陈乙睿 庄 毅

(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 并发控制算法能够保证多个用户同时存取数据库中同一数据时不破坏事务的隔离性和统一性。针对现有并发控制算法适应性较差的问题,提出了自适应并发控制算法。该算法将并发控制过程分为两个阶段:执行授权和策略选择。执行授权阶段根据事务的有效性决定冲突事务执行的顺序;策略选择阶段根据事务的读写状态以及当前冲突率动态地选择乐观/悲观冲突消解策略。设计的策略选择机制使得无论数据库是处于空闲还是繁忙状态,DDCC 算法都具有较高的执行效率。通过对比实验验证了所提出的 DDCC 算法的性能要优于经典的两阶段加锁并发控制算法和 HCC 算法。

关键词 并发控制,动态决策,冲突率预测,冲突消解

中图法分类号 TP311.133.1 文献标识码 A

Concurrency Control Algorithm Based on Dynamic Decision

CHEN Yi-rui ZHUANG Yi

(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract The concurrency control algorithm can guarantee the isolation and consistency of transactions when multiple users access the same date at database simultaneously. This paper proposed an adaptive decision concurrency control algorithm to solve the problem of poor adaptability in existing concurrency control algorithms. This algorithm divides its concurrency control process into two phases: the execution authorizing phase and the strategy selecting phase. In the execution authorizing phase, the algorithm compares effectiveness of transactions to determine the execution order of conflicting transactions. After that, the algorithm selects optimistic/pessimistic conflict resolution strategy dynamically according to the transactions' read/write status and the current conflict rate in the strategy selecting phase. DDCC algorithm has high efficiency no matter whether the system is busy or idle. Performance tests show that DDCC algorithm proposed in this paper is superior to classical strict two phases locking algorithm and hybrid concurrency control algorithm.

Keywords Concurrency control, Dynamic decision, Conflict-rate prediction, Conflict resolution

1 引言

数据库要实现共享必需支持并发执行,但是单纯地并发执行事务,会出现死锁以及数据不一致性等问题。针对这类问题,已有研究者提出了一些并发控制算法[1-8],它们在事务正确执行的同时可保证数据库的一致性以及完整性。

乐观并发控制和悲观并发控制是并发控制算法中最常用的两种方法。在悲观的并发控制中,算法接收到一个操作时,它需要决定是执行、拒绝还是推迟。在乐观的并发控制中,当算法接收到一个操作时便立即执行,在一段时间间隔后(提交过程),乐观的并发控制算法检查事务的执行情况,如果事务执行正常,就继续执行过程;否则算法通过终止重启来撤销事务的某些操作。

悲观并发控制为了保证事务的串行化标准,需要在数据库为事务的所有操作对象维持锁控制。维持锁就意味着有资源的开销,只有发生冲突的可能性较高时(悲观假设)这种开销才有意义,即悲观并发控制适用于冲突可能性较高的情况。另一方面,只有在冲突的可能性较低时乐观并发控制才具有较高的效率。乐观并发控制的前提条件是:当等待队列较少时,事务重启要比队列等待更容易。

为了解决上述问题,本文对混合并发控制(Hybrid Concurrency Control, HCC)算法[7]进行改进,提出基于动态决策的并发控制算法(Dynamic Decision-based Concurrency Control, DDCC),该算法将并发控制过程分为两个阶段:授权执行和策略选择。执行授权阶段根据事务的有效性决定冲突事务执行的顺序;策略选择阶段根据事务的读写状态以及当前冲突率动态地选择乐观/悲观冲突消解策略。DDCC 算法在事务率较高与较低的情况下都具有良好的执行效果。

本文受国家自然科学基金青年科学基金项目(61202351),国家博士后基金项目(一等)(2011M500124),南京航空航天大学基本科研业务费(NS2012133),江苏省普通高校研究生科研创新计划资助项目(CXZZ13-0171)资助。

陈乙睿(1990—),男,硕士生,主要研究方向为分布式数据库,E-mail:chenyirui0607@126.com;庄 毅(1956—),女,教授,主要研究方向为分布计算、计算机网络。

2 相关研究工作

在悲观并发控制算法中，最典型的是强两阶段封锁协议(Strict two-Phase Locking，S2PL)。S2PL 算法在对数据元进行操作之前，需要为事务申请相应的锁(读锁/写锁)。事务执行完后才将锁释放，这样便造成了系统效率较低的问题。为了提高数据库处理事务的效率，Goetz Graefe 等人在自适应索引上实现并发控制算法[1]。算法将自适应索引与锁控制结合在一起，索引一定程度上能够适应并发冲突的数量以及管理控制开销的变化，减少了因工作量变化导致的负载适应需求。

乐观并发控制方面，Nystrom 等人提出一种 CC 算法，算法中软事务使用传统的并发控制技术[2](加锁等方法)，硬事务使用能忽略数据库锁直接进行操作的版本控制方法。为了解决实时分布式系统上使用传统加锁方法反应慢的问题，Mao 等人提出一种具有响应性的细粒度乐观加锁策略的 CC 算法[3]。Makni 等提出一种基于乐观并发控制的算法[4]，这种算法确保了事务时间关系一致，但是算法的执行效率仍然没有得到较好的提高。

文献[5]在两阶段锁(2PL)算法的基础上提出一种基于冲突串行化模型的算法，与一般的锁控制不同，它允许一些存在冲突访问的事务成功提交。算法对于具有长时间事务和高中止率的系统执行效果较好，但是在低中止率的系统中效果并不理想。

Mansour Sheikhan，Mohsen Rohani 等人提出了一种基于神经网络的算法(Neural-based Concurrency Control Algorithm，NCC)[6]，算法采用 ART2 神经网络模型对事务进行分类统计，在冲突时，选择健康程度较好的事务优先解决，能够降低事务的终止率，使得事务更能够顺利执行。NCC 算法与大多数的并发控制算法一样，适用情况单一，不能很好地应对冲突率变化的情况。

文献[7]在 NCC 算法的基础上进行改进，提出 HCC 算法。HCC 算法在决定执行顺序后，通过比较最近 12 分钟的总冲突率的大小来选择不同的并发控制策略。HCC 算法一定程度上解决了上述问题，但是这种方法只能反映最近 12 分钟的平均冲突情况，并不能代表当前时刻冲突发生的概率。

综上所述，虽然在并发控制领域已有较多的研究工作，但是目前的并发控制算法适应性相对较差，而 HCC 算法的敏感性也并不理想，在执行性能上仍然有待提高，所以本文在 HCC 算法的基础上做了进一步研究。

3 基于动态决策的并发控制算法

针对上文中提到的 HCC 算法的不足，本文提出了一种基于动态决策的并发控制算法。该算法根据事务的有效性来决定事务的执行顺序，并预测系统的冲突率，根据预测结果选择乐观/悲观的冲突消解策略。

在阐述 DDCC 算法前先做一些前提假设：

·数据库中的每个对象都有两个版本，即旧版本与新版本。旧版本由已经提交完成的事务写入，新版本则由提交的事务生成。要注意的是，一个对象可能只含有一个旧的版本。

·数据类型 Transaction 的每个对象都是算法处理的最小单元。每个提交的事务请求都有一个 Transaction 对象与之对应，Transaction-ID 是代表相应事务的唯一标识符。

·每个事务都有两个队列即 ListA(先验事务队列)和 ListB(后继事务队列)，它们存储了事务的冲突信息。基于这两个队列，算法可以实现事务的串行化执行和死锁的检测。

3.1 算法模型

图 1 为本文提出的算法模型，数据库接收到请求后将请求存放在事务源中，同时分析该请求，为其分配唯一的结构信息 Transaction。当事务源中事务 T 的先验事务队列为空时，事务 T 转移到事务调度器进行执行，同时 T 由就绪状态转移到执行状态，事务执行完后便进行提交。在此过程中如果发生冲突，则将冲突事务以及冲突信息输入到 DDCC 并发控制模型中进行处理，处理后优胜的事务交由事务调度器继续执行，而状态变更为等待的事务则存放在事务源中，等待其他必要事务的完成。

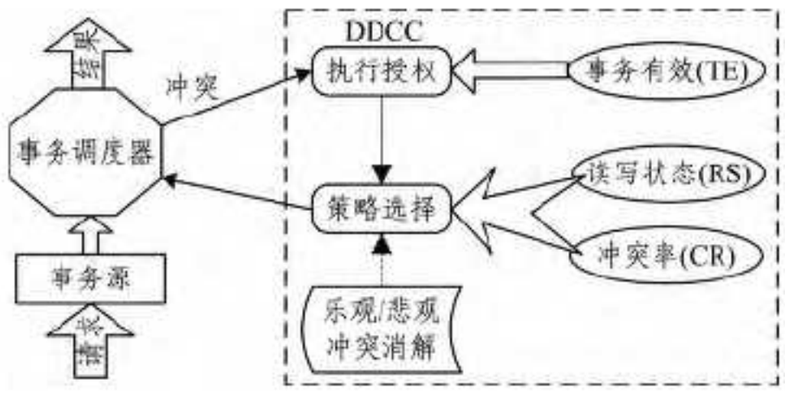

图 1 DDCC 算法模型

DDCC 并发控制模型将并发控制分为执行授权与策略选择两个阶段，执行授权阶段通过对比冲突事务的事务有效性来决定授予两个冲突事务的执行顺序。而策略选择阶段则是根据执行授权的结果来处理，通过事务的串行化执行来消解具体的冲突。在执行中，算法会根据两个事务的读写状态以及当前时刻预测的冲突率来决定具体是采用积极冲突消解策略还是消极冲突消解策略。

图 2 说明了 DDCC 算法进行并发控制的过程。处理冲突时 DDCC 算法将发现的冲突事务 T_1、T_2 交由执行授权过程，执行授权过程根据 TE 信息在两者中选出优先级较高的事务。为了方便说明，这里假设事务 T_2 优胜。授权过程结束后策略选择过程根据两者的读写信息以及冲突率状态决策出具体的冲突消解策略(乐观/悲观)。最后根据决策结果处理响应事务，图示中事务 T_2 将继续执行，而事务 T_1 进入等待状态或者操作其他数据版本。

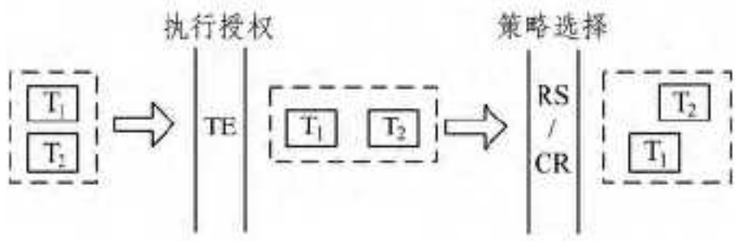

图 2 DDCC 算法执行过程

3.2 事务执行授权

DDCC 算法的第一阶段为执行授权阶段。该阶段根据事务的事务有效性对冲突事务授权，决定冲突事务的执行顺序。

3.2.1 事务有效性

NCC 算法中事务有效性 TE 的计算公式见式(1)[7]，其中 $UserPriority(T)$ 为事务 T 的优先级或者用户分配给它的重要性指标。

$$TE=UserPriority(T)+SUM(TE \text{ of all } W \text{ type transactions in list } B \text{ of}) \quad (1)$$

式(1)计算了事务 T 的 $UserPriority$ 与其后续执行事务

队列中所有写类型事务的 TE 值之和。这种计算方法会造成有些相对并不紧急的事务因为后续事务的重复计算而使其 TE 值反而较高的情况，同时它只考虑写类型的事务的优先性，这会使得读类型事务等待时间相对过长。针对这种缺陷，本文改进了事务有效性的计算方法。

为了方便描述，这里先给出一些简单的定义，使用 $T_i(i=1,2\cdots,n)$ 表示 DDCC 算法正在处理的 n 个事务中的第 i 个事务，其中 n 为事务的总数。

定义 1　如果事务 T_2 存在于事务 T_1 的后续执行队列中，那么称 T_1 先于 T_2，表示为 $T_1 \preceq T_2$。

定义 2　事务的先于关系满足传递律，即满足式(2)：

$$T_1 \preceq T_2 \wedge T_2 \preceq T_3 \Rightarrow T_1 \preceq T_3 \tag{2}$$

根据定义 2，事务有效性 TE 的计算公式见式(3)。其中 $ItemType(T_i)$ 表示事务 T_i 的操作类型；R、W 分别表示读类型与写类型。

$$TE=\sum_{i=1}^{n}(UserPriority(T_i) \mid T \preceq T_i \wedge ItemType(T_i) \in \{R,W\}) \tag{3}$$

式(3)与式(1)相比有两个方面的优点，一方面所有后续队列中的元素都只计算一次优先级，这样便能够去除间接后续事务的重复计算，得到的事务有效性值更加符合实际情况；另一方面，读等待事务的优先级也能够由先验事务继承，这样能够相对促进那些等待时间较长的读等待事务优先执行。按照式(3)决定的执行顺序能够减少当前时刻事务的平均等待时间，同时也可提高事务的执行效率。

式(3)给出的事务有效性计算方法与文献[9,10]中优先级继承方法具有相类似的功能，能够在有效评估事务重要性的同时反映该事务待执行的紧急程度。如果后续队列中事务较多，那么相应地计算出来的事务有效性程度 TE 也较高，这种计算方法在一定程度上可减少长时间等待事务的持续等待。

3.2.2　事务执行授权算法

基于 DDCC 并发控制模型以及上一小节提出的事务有效性计算方法，设计的授权执行算法思想与步骤如下：

Step1　如果两个事务已存在冲突记录，提取两个事务的后续队列，根据式(3)计算冲突事务的事务有效性；

Step2　比较两个冲突事务的事务有效性，TE 值较高的事务优先执行，剩下的事务进入等待状态；

Step3　进入策略选择过程。

事务执行授权算法根据事务计算出来的 TE 值选择相对明智的执行顺序

DDCC 的执行授权算法伪代码见图 3。

```
Exec_ Authorizing
输入参数：Transaction A,Transaction B;两个冲突事务
算法功能：根据事务有效性决定两个冲突事务执行的优先级，选择后进
入策略选择过程消解冲突。

步骤:
Step1  AfTransA=GetAllTrans(Trans_A);//获取Trans_A的后继事务集合
Step2  AfTransB=GetAllTrans(Trans_B);//获取Trans_B的后继事务集合
Step3   If(TE(Trans_A)>=TE(Trans_B)) //计算比较两个事务的事务有效性
Step4    TransSelected=Trans_A;
Step5   else
Step6    TransSelected=Trans_B;
Step7   Policy_Sel(TransSelected,Trans_A,Trans_B);
```

图 3　执行授权算法伪代码

3.3　策略选择

DDCC 算法的第二阶段为策略选择阶段。该阶段根据读写状态以及当前预测的冲突率选择乐观/悲观冲突消解策略，并针对事务采取相应策略的操作，消解事务之间的冲突。

策略选择阶段的关键在于准确、及时地判断当前系统的状态，本文提出了冲突率预测方法。本小节将重点介绍冲突率预测方法以及策略选择算法。

3.3.1　冲突率预测

HCC 算法使用过去的 12 分钟冲突数的平均值来选择策略[7]，具有较大的滞后性。理想情况是根据当前时刻的冲突状况进行判断。但是如同当前时刻是否发生事故是无法判断的，无法根据某一时刻系统状态判断事务发生冲突的概率，所以本文通过冲突率来预测当前时刻的可能的冲突情况。

我们采用线性回归预测方法预测当前时刻的冲突率 E。时间为 t，冲突率预测的样本为过去的 N 分钟，k 是 E 的回归系数。冲突率 E 满足式(4)：

$$\frac{E-\bar{E}}{t-\bar{t}}=k \tag{4}$$

式中，k 是 E 的回归系数，它由式(5)计算而来，其中 E_i 为第 i 分钟的平均冲突率。

$$k=\frac{\sum_{i=1}^{N}t_iE_i-N\bar{t}\,\bar{E}}{\sum_{i=1}^{N}t_i^2-N\bar{t}^2} \tag{5}$$

式(4)和式(5)推导后可得出式(6)，它给出了当前冲突率 E 的预测方法。

$$E=\bar{E}+\frac{N+1}{2}\,\frac{\sum_{i=1}^{N}iE_i-N\frac{N+1}{2}\bar{E}}{\sum_{i=1}^{N}i^2-N(\frac{N+1}{2})^2} \tag{6}$$

本文利用过去 N 分钟统计的冲突数构成线性回归方程，并根据方程推导出当前冲突率的计算公式。相比于 NCC 计算平均冲突率的方法，本文的方法将冲突率变化的趋势考虑在内，对当前时刻的冲突率做出相对准确的预测。它解决了 NCC 算法在策略选择时存在的滞后性问题，使用式(6)预测的当前冲突率更加符合实际情况，消解冲突时效率也更高。

利用预测的冲突率 E 决策时，需要为 E 指定阈值 Eh。当 $E<Eh$，即冲突率较低时，采用乐观冲突消解策略；反之，则采用悲观冲突消解策略。要注意的是，Eh 要设置得当，如果 Eh 太大或者太小，决策便会接近纯乐观或者纯悲观状态。

3.3.2　策略选择算法

基于 DDCC 并发控制模型设计的策略选择算法思想与步骤如下：

Step1　判断优先执行事务的先验队列与等待事务的后继队列是否存在交集。如果没有，转 Step3，进入冲突消解过程。

Step2　交换优先执行事务与等待事务，同样判断是否存在交集，如果没有，转 Step3，进入冲突消解过程；如果有交集，说明造成死锁，返回失败并重启两个事务。

Step3　在冲突事务的读写集合已知的情况下，如果事务的读写集合满足式(7)，转 Step5；如果不满足式，转 Step6。

Step4　冲突事务的读写集合未知时，如果当前冲突率 $E<Eh$，转 Step5；否则转 Step6。

Step5 采用乐观冲突消解策略，同时执行冲突事务，如果发生死锁，重启两个事务即可。

Step6 用悲观冲突消解策略，执行优先事务，另一个事务进入等待状态。

本文设计的 DDCC 的策略选择算法的伪代码见图 4。

```
Strategy_Select
输入参数：TransSelected，Trans_A,Trans_B
算法功能：在优先保证TransSelected优先执行的情况下消解Trans_A与Trans_B的冲突
步骤：
Step1 优先执行事务TransExec=TransSelected;
Step2 等待事务TransWait=TransSelected == Trans_A ? Trans_A : Trans_B;
区分优先执行事务及等待事务。
Step3 If(ListA(TransExec)∩ListB(TransWait)=null)
Step4   转Step11
Step5 Else
Step6   If(ListA(TransWait)∩ListB(TransWait)=null)
Step7     交换TransExec与TransSelected
Step8   Else
Step9     已经造成死锁，重启两个事务
Step10 End; End;
Step11 If（E<Eh）
Step12   if（WS(TransExec)-X(TransExec)∩RS(TransWait)=null）
Step13     添加TransExec到TransWait的ListA中，
       添加TransWait到TransExec的ListB中
Step14   else
Step15     TransWait.State=WAIT,
       添加TransWait到TransExec的ListB(SOFT_WAIT)
Step16 else
Step17   将TransExec添加到TransWait的ListA中，
       将TransWait添加到TransExec的ListB中
```

图 4 策略选择算法伪代码

在进行决策之前，首先对执行授权结果进行验证，如果会造成死锁，则采取相应的处理措施，中断重启或者变更执行的顺序；然后根据读写状态选择相应的冲突消解策略，如果满足式(7)，便采用乐观的冲突消解策略，反之则选择悲观的冲突消解策略。

$$(WS(TransExec)-X(TransExec))\cap RS(TransWait)=\varnothing \tag{7}$$

式中，$WS(T)$ 表示事务 T 的写操作对象集合，$X(T)$ 表示事务已完成的操作对象集合，而 $RS(T)$ 则表示事务的读操作对象集合。

而在事务读写状态未知的情况下，根据当前冲突情况选择冲突消解方案，如果 $E<Eh$，选择乐观的冲突消解策略；否则，选择悲观冲突消解策略。

策略选择阶段的死锁判断能够提前处理已经死锁的事务，提高 DDCC 算法的执行效率。而在冲突率较高或者两个冲突事务的后续读写对象存在交集时，事务发生死锁或者再次冲突的概率较高，此时采用消极冲突消解，能够保证事务的成功执行；反之则说明发生死锁或者再次冲突的概率较低，便采用乐观冲突消解以提高事务的执行效率，此时即使导致中断或再次冲突，也是在系统的可容忍范围内的。

4 仿真实验与结果分析

为了对 DDCC 算法进行有效评估，进行 3 个 Matlab 仿真实验，实验环境为 3.27GHz 下 32 位 Windows 7 系统。实验的前提假设是：工程软件数据库每秒收到 n 个事务请求，每个请求包含 1—6 条命令，命令会对 1000 个数据对象产生随机的读写操作。模拟出的 80% 的指令针对 20% 的数据对象进行操作。DDCC 算法中预测统计时间 $N=12$，冲突率阈值 $Eh=5$。

实验时，事务率随着时间的增长而不断增加，以提高事务冲突率。为了方便对比分析，实验记录下每分钟事务中断的个数和该分钟内事务的平均执行时间。本文算法的对比对象采用前文提到的经典 S2PL 算法和该论文主要参考的 HCC 算法。

图 5 为 3 种算法在事务率上升情况下中断个数曲线的对比。图中横坐标为时间，该实验中随着时间的增长，事务率也在线性增加，而纵坐标记录的是每分钟事务中断的个数。从曲线中可以看出，3 种算法随着事务率增长，中断数也在增加。相比而言，HCC 算法和 DDCC 算法总体上都要比 S2PL 中断率要低。值得注意的是，DDCC 在某段时间内比 HCC 算法要好，这是因为 DDCC 算法预测冲突率更为准确，能够及时变更冲突消解策略，降低了事务中断概率，更保证了算法的顺序执行。所以相比而言，DDCC 算法优于 HCC 算法和 S2PL 算法。

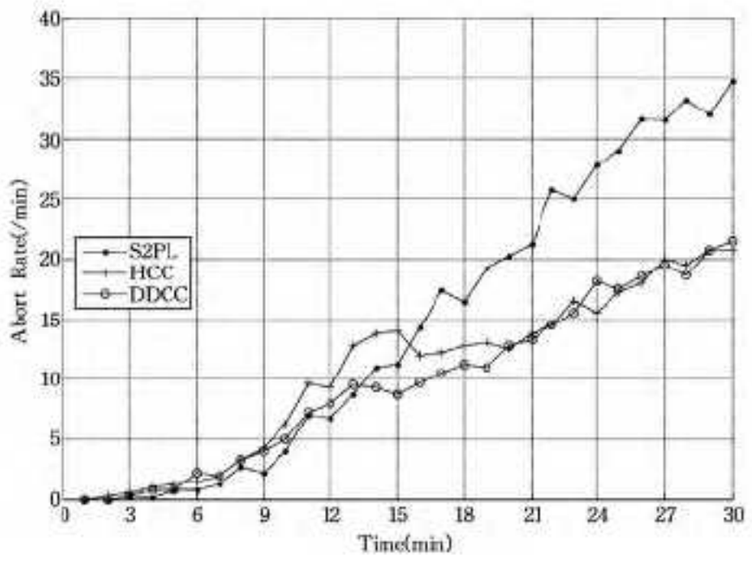

图 5 算法中断个数曲线对比

图 6 对比了 3 种算法在不同事务率情况下事务平均执行时间。从图中可以看出，HCC 算法和 DDCC 算法由于在事务率低时采用乐观冲突消解，节省了锁申请的过程，因此平均执行时间较短，而随着事务率的增加，3 种算法的中断个数也有所增长，这导致平均执行时间不断提升。因为 S2PL 算法单纯进行锁申请与锁释放，所以在事务率较高的情况下中断个数也多，这就意味着其平均时间比 HCC 算法和 DDCC 算法的要长。因为 HCC 算法在决定执行顺序时需要使用 ART-2 对事务进行分类，相比于 DDCC 算法更为耗时，所以实验结果要劣于 DDCC 算法。

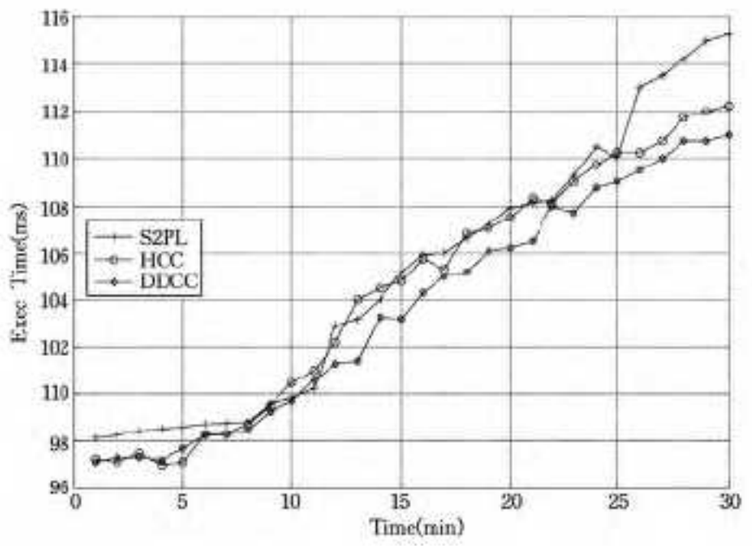

图 6 事务平均执行时间曲线对比

结束语 本文研究了已有的 HCC 并发控制算法，并针对 HCC 算法的不足，提出一种并发控制算法：DDCC 算法。该算法将并发控制过程分为执行授权和策略选择两个阶段。执行授权阶段根据事务的有效性进行判断，选择优先级较高的事务优先执行。策略选择阶段能够根据事务的读写状态以

（下转第 28 页）

$$=\frac{\frac{B^*(y_0)-D(x_0)}{1-\lambda+\lambda D(x_0)}+1+(1-\lambda)(1-R(A(x_0),B(y_0)))}{1-\lambda+\lambda R(A(x_0),B(y_0)}$$

$$<\frac{\alpha R(A(x_0),B(y_0))-(1-\alpha)(1-\lambda)(1-R(A(x_0),B(y_0)))-1+1+(1-\lambda)(1-R(A(x_0),B(y_0)))}{1-\lambda+\lambda R(A(x_0),B(y_0))}$$

$$=\frac{\alpha R(A(x_0),B(y_0))+\alpha(1-\lambda)(1-R(A(x_0),B(y_0)))}{1-\lambda+\lambda R(A(x_0),B(y_0))}=\alpha$$

所以 $(A(x_0)\to B(y_0))\to(D(x_0)\to B^*(y_0))<\alpha$，即 $D(x_0)$ 不能使式(1)成立。

综上所述可知，定理4是恒成立的。

结束语　本文提出了新的蕴涵算子族 L-λ-Ⅱ，并基于该蕴涵算子族讨论了 FMP 模型和 FMT 模型的三Ⅰ支持算法、α-三Ⅰ支持算法。所得到的结论进一步丰富和发展了全蕴涵三Ⅰ算法的相关理论。如何给出常用的 R_0、Gödel、Łukasiewicz、Goguen 等蕴涵算子三Ⅰ算法的更一般形式，笔者将另文讨论。

参 考 文 献

[1] Zadeh L A. Outline of new approach to the analysis of complex systems and decision processes [J]. IEEE Trans on Systems, Man and Cybernetics, 1973, 3(1): 28-33

[2] 王国俊. 模糊推理的全蕴涵三Ⅰ算法[J]. 中国科学(E辑), 1999, 29(1): 43-53

[3] 王国俊. 模糊推理的一个新方法[J]. 模糊系统与数学, 1999, 13(3): 1-9

[4] 徐蔚鸿, 谢中科, 杨静宇, 等. 两类模糊推理算法的连续性和逼近性[J]. 软件学报, 2004, 15(10): 1485-1492

[5] 王国俊, 宋庆燕. 一种新型的三Ⅰ算法及其逻辑基础[J]. 自然科学进展, 2003, 13(6): 575-581

[6] 裴道武. FMT 问题的两种三Ⅰ算法及其还原性[J]. 模糊系统与数学, 2001, 15(4): 1-6

[7] 王国俊. 数理逻辑引论与归结原理[M]. 北京: 科学出版社, 2006

[8] 王国俊. 非经典数理逻辑与近似推理[M]. 北京: 科学出版社, 2008

[9] 裴道武. 模糊推理全蕴涵算法及其还原性[J]. 数学研究与评论, 2004, 24(2): 359-368

[10] 宋士吉, 吴澄. 模糊推理的反向三Ⅰ算法[J]. 中国科学(E辑), 2002, 32(2): 230-246

[11] 宋士吉, 吴澄. 模糊推理的反向三Ⅰ约束算法[J]. 自然科学进展, 2002, 12(1): 95-100

[12] 彭家寅, 侯建, 李洪兴. 基于某些常见蕴涵算子的反向三Ⅰ算法[J]. 自然科学进展, 2005, 15(4): 404-410

[13] 彭家寅. 基于某些常见蕴涵算子的模糊推理全蕴涵三Ⅰ约束算法[J]. 自然科学进展, 2005, 15(5): 539-546

[14] 张兴芳, 孟广武. 蕴涵算子族及其应用[J]. 计算机学报, 2007, 30(3): 448-453

[15] 张森, 李成允, 张兴芳. 正则蕴涵算子族 $G-\lambda-R_0$ 及其三Ⅰ支持算法[J]. 计算机工程与应用, 2009, 45(22): 29-31

[16] 支晓斌, 范九伦. 一个新的蕴涵算子族[J]. 模糊系统与数学, 2010, 24(4): 12-18

[17] 张森, 张兴芳. 一类蕴涵算子下的支持度及 α-三Ⅰ算法[J]. 计算机工程与应用, 2011, 47(1): 43-45

[18] 彭家寅. 基于蕴涵算子族 $H_{(p,\lambda)}$ 的三Ⅰ算法模糊系统[J]. 计算机工程与应用, 2013, 49(1): 53-58

(上接第4页)

及预测的当前冲突率选择乐观/悲观冲突消解策略。DDCC 算法使得数据库在不同事务率的情况下并发控制效果都较好。通过实验对比发现，DDCC 算法的性能要优于经典两阶加锁并发控制 S2PL 算法和 HCC 算法。DDCC 算法能够在系统较闲、事务较少时快速响应，迅速处理完事务；而在系统较忙、冲突较高时，在确保事务等待时间较短的情况下保证事务正确有序地执行，尽量避免了事务的中断重启。

参 考 文 献

[1] Graefe G, Halim F, Idreos S, et al. Concurrency control for adaptive indexing[J]. Proceedings of the VLDB Endowment, 2012, 5(7): 656-667

[2] Nystrom D, Nolin M, Tesanovic A, et al. Pessimistic concurrency control and versioning to support database pointers in real-time databases[C]// 16th Euromicro Conference on Real-Time Systems, 2004(ECRTS 2004). IEEE, 2004: 261-270

[3] Mao Q, Wang J, Zhan Y. The optimistic locking concurrency controlling algorithm based on relative position and its application in real-time collaborative editing system[C]// The 8th International Conference on Computer Supported Cooperative Work in Design, 2004. IEEE, 2004, 1: 99-105

[4] Makni A, Bouaziz R, Gargouri F. Formal verification of an optimistic concurrency control algorithm using SPIN[C]// Thirteenth International Symposium on Temporal Representation and Reasoning, 2006(TIME 2006). IEEE, 2006: 160-167

[5] Aydonat U, Abdelrahman T S. Relaxed concurrency control in software transactional memory[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(7): 1312-1325

[6] Sheikhan M, Rohani M, Ahmadluei S. A neural-based concurrency control algorithm for database systems[J]. Neural Computing and Applications, 2013, 22(1): 161-174

[7] Sheikhan M, Ahmadluei S. An intelligent hybrid optimistic/pessimistic concurrency control algorithm for centralized database systems using modified GSA-optimized ART neural model[J]. Neural Computing andApplications, 2013, 23(6): 1815-1829

[8] 彭林, 谢伦国, 张小强. 采用向量时钟的软件事务存储算法[J]. 计算机科学, 2010, 37(5): 282-286

[9] Jea K F, Chen S Y. A high concurrency XPath-based locking protocol for XML databases[J]. Information and Software Technology, 2006, 48(8): 708-716

[10] Pleshachkov P O, Kuznetsov S D. Transaction management in RDBMSs with XML support[J]. Programming and Computer Software, 2006, 32(5): 243-254