实时数据库并发控制协议及其 Petri 网分析

陈 俊,朱艳丽,古乐声
CHEN Jun,ZHU Yan-li,GU Yue-sheng

河南科技学院 信息工程学院,河南 新乡 453003
School of Information Engineering,Henan Institute of Science and Technology,Xinxiang,Henan 453003,China
E-mail:cj626@hist.edu.cn

CHEN Jun,ZHU Yan-li,GU Yue-sheng.New speculative concurrency control protocol and analysis based on Petri net. Computer Engineering and Applications,2009,45(21):121-123.

Abstract: A new Speculative Concurrency Control Protocol(NSCC) for real-time database is proposed.The protocol is based on the traditional SCC.Mang restartings are advoided,and the concurrency of transaction is enhanced.Finally,using theory of Petri net,it proves that the protocol is feasible and effective.

Key words: real-time database;concurrency control;New Speculative Concurrency Control(NSCC);Petri net

摘 要:该文提出一种适用于实时数据库的新可推测并发控制(New Speculative Concurrency Control,NSCC)协议。该协议在传统的 SCC 协议基础之上,进行一系列改进,避免了大量不必要的事务重启,提高了并发度。最后,通过 Petri 网理论验证其可行性和正确性。

关键词:实时数据库;并发控制;新可推测并发控制(NSCC);Petri 网
DOI:10.3778/j.issn.1002-8331.2009.21.036 **文章编号:**1002-8331(2009)21-0121-03 **文献标识码:**A **中图分类号:**TP311.131

对于实时数据库系统来说,事务执行的最大并发度是非常重要的性能指标,特别是在系统资源充足时,高并发度可以提高系统资源的利用率。在截止期内提交的事务所占的比率对实时数据库系统来说是决定性的性能指标。

可推测并发控制[1](Speculative Concurrency Control,SCC)特别适合实时数据库。一方面,SCC 类似于 PCC,能尽可能早地检测到潜在的有害冲突,启动一个替换调度,从而增加事务满足时间限制的机会。另一方面,SCC 类似于 OCC,它允许冲突事务并发执行,因此避免了不必要的可能影响事务及时提交的延迟。这样一来,它就减轻了 PCC 的阻塞问题和 OCC 的重启问题,从而更好地满足事务截止期。

基于此,提出一种适用于实时数据库的新可推测并发控制协议(New Speculative Concurrency Control,NSCC)。该协议在传统的 SCC 协议基础之上,进行一系列改进,避免了大量不必要的事务重启,提高了并发度。最后,通过 Petri 网理论验证其可行性和正确性。

1 NSCC 协议基本执行规则

1.1 初始化规则[2-3]

设 T_r 是就绪事务,当它获得 CPU 调度时,系统为它创建一个乐观影子 T_r^0,并且根据事务 T_r 的优先级、截止期限和系统信息,为其分配一个 n 值,它表示该事务可拥有的影子的最大数目。初始化 SpecNum(T_r)为 0,ReadSet(T_r^0)和 WriteSet(T_r^0)均为空集。

1.2 冲突规则[3]

在 T_r^0 的执行过程中,检测它与其他并发事务的冲突。假定 T_u^0 为与其冲突的事务集中的任意一个。可能的冲突类型以及解决方法如下:

(1)读规则

当 T_r^0 在执行读操作时,检测出它与 T_u^0 的写操作产生冲突,称为冲突事务 T_r^0 与 T_u^0 的读-写冲突,记作 ReadSet(T_r^0)∩WriteSet(T_u^0)≠ϕ。

此时,如果事务 T_r 对应的影子数目未达到 n,并且不存在对应于 T_u^0 冲突的投机影子,则要在冲突检测点开始,为事务 T_r 建立一个投机的影子。该影子可以通过直接复制乐观影子 T_r^0 得到,且该影子被阻塞直到事务 T_u^0 成功提交。

否则,由于缺少负责解决冲突的投机影子,此潜在的冲突不得不被忽略。

(2)写规则

当 T_u^0 在执行写操作时,检测出它与 T_r^0 的读操作产生冲突,称为冲突事务 T_u^0 与 T_r^0 的写-读冲突,记作 WriteSet(T_u^0)∩ReadSet(T_r^0)≠ϕ。

此时,如果事务 T_r 对应的影子数目未达到 n:

作者简介:陈俊(1979-),女,助教,研究领域:实时数据库,Petri 网理论。
收稿日期:2009-05-04 **修回日期:**2009-06-22

若不存在对应于 T_u^0 冲突的投机影子，则需要从先于 T_r^0 读操作的某个点为事务 T_r 建立一个投机的影子，且该影子被阻塞直到事务 T_u^0 成功提交；

若存在对应于 T_u^0 冲突的投机影子 T_r^k，则先令 T_r^k 夭折再从先于 T_r^0 读操作的某个点为事务 T_r 建立一个投机的影子，且该影子被阻塞直到事务 T_u^0 成功提交。

(3)当 T_r^0 在执行写操作时，检测出它与 T_u^0 的写操作产生冲突，称为冲突事务 T_r^0 与 T_u^0 的写-写冲突，记作 $\mathrm{WriteSet}(T_r^0)\cap \mathrm{WriteSet}(T_u^0)\neq\phi$。

根据 Thomas 的写规则，可以通过删除事务发出的过时写操作来保证执行的可串行化。因此，在这种情况下，忽略冲突继续执行。

1.3 阻塞规则

若事务 T_r^i 预读取数据 X，而与事务 T_u^0 的写操作发生冲突，则事务 T_r^i 必须等待 T_u^0 提交后，才能继续执行操作。因此称事务 T_r^i 被阻塞，记作 $(T_u,X)\in \mathrm{WaitFor}(T_r^i)$。

1.4 准提交规则[4-5]

当事务 T_r^0 执行完所有的读、写操作时，便进入准提交阶段。事务 T_r^0 进行准提交操作之后处于已提交状态，不会因为数据访问冲突而被重启或者因为错失截止期而被抛弃，而且该事务所做的修改对数据库是有效可读的，活动事务可以读取该事务的私有缓存来得到该事务所做的修改。

此时，判断与之冲突的事务中截止期早的所占比例，若小于阀值，则进入全提交阶段。当截止期临近时，准提交事务也必须进入全提交阶段。

1.5 提交规则

如果乐观影子 T_r^0 进入全提交阶段，它的执行过程为：

(1)将 T_r 的投机影子全部作废并中断。

(2)对于每一个与 T_r 冲突的事务 T_u，执行以下操作：

①如果 T_u 有一个对应于 T_r 冲突的投机影子 T_u^i，那么，将 T_u 的乐观影子 T_u^0 中断，将投机影子 T_u^i 提升为新乐观影子。同时，对于 T_u 已经执行了冲突中读操作的投机影子也随之中断。

②如果 T_u 不存在对应于 T_r 冲突的投机影子，那么，将 T_u 的乐观影子 T_u^0 中断，在投机影子中选择没有执行冲突中的读操作且最新创建的影子，将其提升为新的乐观影子。

2 NSCC 协议的 Petri 网及其正确性证明

2.1 NSCC 协议的 Petri 网

给定事务间的一个并发执行，如果它们访问同一个客体，T_r 的行为 a_r^i 和事务 T_u 的行为 a_u^j 存在冲突，当 a_r^i 和 a_u^j 都执行写操作，那么使用 Thomas 的 TWR 规则就可以解决它们之间的写—写冲突；a_r^i 执行读操作 a_u^j 执行写操作，或者 a_r^i 执行写操作 a_u^j 执行读操作，则使用 NSCC 协议来解决冲突。

在事务处理过程中的基本执行规则相同，因此两个实时事务的行为特性和多个事务的类似，不失一般性。假定两个实时事务 T_r 和 T_u，分别只完成一个操作并产生冲突，即 $T_r:r(x)$ 和 $T_u:w(x)$。此时两冲突实时事务的 Petri 网模型如图 1 所示，该图描述如下：

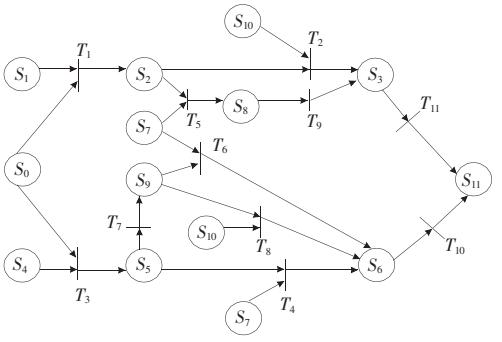

图 1 NSCC 协议的 Petri 网 NSCC-PN

(1)NSCC 协议解决冲突的方法是建立投机影子而非加锁，因此不存在死锁问题。如果实时事务 T_r 和 T_u 发生读-写冲突，两事务的乐观影子 T_r^0 和 T_u^0 继续并发执行，只需要读事务 T_r 建立与冲突对应的投机影子 T_r^1 并阻塞。

(2)如果写事务 T_u^0 先提交，读事务乐观影子 T_r^0 夭折，同时投机影子 T_r^1 变成新的乐观影子开始执行。

(3)如果读事务 T_r^0 先提交，读事务投机影子 T_r^1 夭折，同时写事务乐观影子 T_u^0 夭折并产生新的乐观影子开始执行。

NSCC 协议的 Petri 网 NSCC-PN=(P,T,F,W,M_0)，其中：

$P=\{S_0,S_1,\cdots,S_{11}\}$ $T=\{T_1,T_2,\cdots,T_{11}\}$(详情见表 1)

$F=\{(S_0,T_1),(S_0,T_3),(S_1,T_1),(S_2,T_2),(S_2,T_5),(S_3,T_{11}),(S_4,T_3),(S_5,T_4),(S_5,T_7),(S_6,T_{10}),(S_7,T_4),(S_7,T_5),(S_7,T_6),(S_8,T_9),(S_9,T_6),(S_9,T_8),(S_{10},T_2),(S_{10},T_8),(T_1,S_2),(T_2,S_3),(T_3,S_5),(T_4,S_6),(T_5,S_8),(T_6,S_6),(T_7,S_9),(T_8,S_6),(T_9,S_3),(T_{10},S_{11}),(T_{11},S_{11})\}$

$W=\{1,1,\cdots,1\}$

初始标记 M_0 有两个，分别对应以下两种情况：

写事务先提交 $M_{0w}=(2,1,0,0,1,0,0,0,0,0,2,0)$

读事务先提交 $M_{0r}=(2,1,0,0,1,0,0,2,0,0,0,0)$

表 1 NSCC-PN 中各库所和变迁

库所	意义	变迁	意义
S_0	事务冲突产生状态	T_1	事务 T_u 预写操作
S_1	事务 T_u 处于开始状态	T_2	乐观影子 T_u^0 进行写操作
S_2	乐观影子 T_u^0 处于执行状态	T_3	事务 T_r 预读操作
S_3	事务 T_u 处于写操作完成状态	T_4	乐观影子 T_r^0 进行读操作
S_4	事务 T_r 处于开始状态	T_5	乐观影子 T_u^0 夭折并产生新乐观影子
S_5	乐观影子 T_r^0 处于执行状态	T_6	投机影子 T_r^1 夭折
S_6	事务 T_r 处于读操作完成状态	T_7	产生投机影子 T_r^1 并阻塞
S_7	乐观影子 T_r^0 处于提交状态	T_8	乐观影子 T_u^0 夭折投机影子 T_r^1 执行
S_8	新乐观影子 T_u^0 处于执行状态	T_9	新乐观影子 T_u^0 执行
S_9	投机影子 T_r^1 处于阻塞状态	T_{10}	事务 T_r 提交
S_{10}	乐观影子 T_u^0 处于提交状态	T_{11}	事务 T_u 提交
S_{11}	事务处于结束状态		

2.2 NSCC 协议的正确性证明

NSCC 协议的 Petri 网 NSCC-PN 有两个初始标记 M_{0w}

和 M_{0r} 分别对应两个不同的可达树 NSCC-PNWT(如图 2)和 NSCC-PNRT(如图 3),其中的所有标记如图 2 所示。

表 2　NSCC-PN 中的标记

M_{0u}=(2,1,0,0,1,0,0,0,0,0,2,0)	M_1=(1,0,1,0,1,0,0,0,0,0,2,0)
M_2=(1,1,0,0,0,1,0,0,0,0,2,0)	M_3=(1,0,0,1,1,0,0,0,0,0,1,0)
M_4=(0,0,1,0,0,1,0,0,0,0,2,0)	M_5=(1,1,0,0,0,0,0,0,0,1,2,0)
M_6=(1,0,0,0,1,0,0,0,0,0,1,1)	M_7=(0,0,0,1,0,1,0,0,0,0,1,0)
M_8=(0,0,1,0,0,0,0,0,0,1,2,0)	M_9=(1,1,0,0,0,1,0,0,0,0,1,0)
M_{10}=(0,0,0,0,0,1,0,0,0,0,1,1)	M_{11}=(0,0,0,1,0,0,0,0,0,1,1,0)
M_{12}=(0,0,1,0,,0,0,1,0,0,0,0,1,0)	M_{13}=(1,1,0,0,0,0,0,0,0,0,1,1)
M_{14}=(0,0,0,0,0,0,0,0,0,1,1,1)	M_{15}=(0,0,0,1,0,0,1,0,0,0,0,0)
M_{16}=(0,0,1,0,0,0,0,0,0,0,1,1)	M_{17}=(0,0,0,0,0,0,1,0,0,0,0,1)
M_{18}=(0,0,0,1,0,0,0,0,0,0,0,1)	M_{19}=(0,0,0,0,0,0,0,0,0,0,0,2)
M_0=(2,1,0,0,1,0,0,2,0,0,0,0)	M_{20}=(1,0,1,0,1,0,0,2,0,0,0,0)
M_{21}=(1,1,0,0,0,1,0,2,0,0,0,0)	M_{22}=(1,0,0,0,1,0,0,1,1,0,0,0)
M_{23}=(0,0,1,0,0,1,0,2,0,0,0,0)	M_{24}=(1,1,0,0,0,0,1,1,0,0,0,0)
M_{25}=(1,1,0,0,0,0,0,2,0,1,0,0)	M_{26}=(1,0,0,1,1,0,0,1,0,0,0,0)
M_{27}=(0,0,0,0,0,1,0,1,1,0,0,0)	M_{28}=(0,0,1,0,0,0,1,1,0,0,0,0)
$M29$=(0,0,1,0,0,0,0,2,0,1,0,0)	M_{30}=(1,0,0,0,1,0,0,1,0,0,0,1)
M_{31}=(0,0,0,1,0,1,0,1,0,0,0,0)	M_{32}=(0,0,0,0,0,0,0,1,1,1,0,0)
M_{33}=(0,0,0,0,0,0,1,0,1,0,0,0)	M_{34}=(0,0,1,0,0,0,0,1,0,0,0,1)
M_{35}=(0,0,0,0,0,1,0,1,0,0,0,1)	M_{36}=(0,0,0,1,0,0,0,1,0,1,0,0)
M_{37}=(0,0,0,0,0,0,0,0,1,0,0,1)	M_{38}=(0,0,0,0,0,0,0,0,1,0,1,0,1)

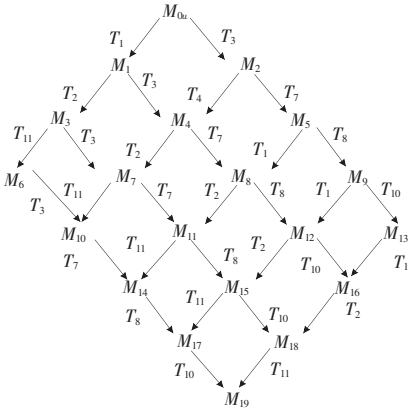

图 2　NSCC-PN 在写事务先提交时的可达树 NSCC-PNWT

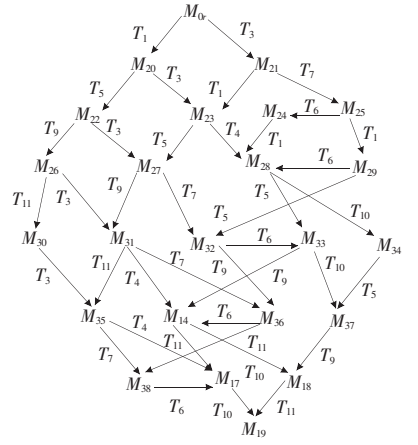

图 3　NSCC-PN 在读事务先提交时的可达树 NSCC-PNRT

定理　NSCC 协议是正确的。

证明　利用可达树分析技术，可知 NSCC 协议的 Petri 网 NSCC-PN 具有下列动态特性[6]:

(1)NSCC-PN 中的任一状态都是初始可达的

NSCC-PN 的初始标识 M_0={M_{0u},M_{0r}} 从上述可达树中可以看出,由 M_{0u} 开始的状态可达集 $R(M_{0u})$={M_{0u},M_1,M_2,…,M_{19}};M_{0r} 开始的状态可达集 $R(M_{0r})$={M_{0r},M_{14},M_{16},M_{17},…,M_{38}}。因此,$R(M_0)$=$R(M_{0u})$∪$R(M_{0r})$={M_{0u},M_{0r},M_1,M_2,…,M_{38}}。SCC-PN 的状态集 MS={M_{0u},M_{0r},M_1,M_2,…,M_{38}}。因此,对任一标记 M_i∈MS,均有 M_i∈$R(M_0)$,即 NSCC-PN 的任一状态都是从 M_0 可达的。

也就是说,该 Petri 网中没有孤立的死标识或无用的标识,这意味着 NSCC 协议在执行过程中不会出现无法跳出的状态或无用的状态。

(2)NSCC-PN 是有界的

在 NSCC-PN 的可达树 NSCC-PNWT 和 NSCC-PNRT 中,任何位置上的令牌数从未超过 2。因此,NSCC-PN 是有界的并且其上界为 2。Petri 网有界则表明 NSCC 协议在任何情况下所处的状态是确定的。

(3)NSCC-PN 是 Ll 活性的(Ll-live)

有界 NSCC-PN 的可达树 NSCC-PNWT 和 NSCC-PNRT 包含了所有可能的标记。从可达树中不难看出，从 M_0 开始，∀Vt_i∈T(i=1,2,…,11)都可以被 $L(M_0)$中的某个点火序列至少点火一次，即所有的变迁都是 Ll 活性的，因此,Petri 网 NSCC-PN 是 Ll 活性的。

从 Petri 网某初态开始一定能够到达相应的终态[6],这意味着 NSCC 协议从初始状态 M_0 开始后无论出现什么情况都不会死锁,即它们可以正确地结束事务。

(4)NSCC-PN 具有完整性

NSCC-PN 是有界的,其可达树就是其可达集。从其可达树可以看到，当写事务先提交时,NSCC-PN 的所有状态都是 M_{0u} 可达的并且可以点火转移到终止状态 M_{19}=(0,0,0,0,0,0,0,0,0,0,0,2);当读事务先提交时,NSCC-PN 的所有状态都从 M_{0r} 可达并且也可以点火转移到终止状态 M_{19}。因此,无论是写事务先提交还是读事务先提交,NSCC 协议都能正确地维护系统一致性。

(5)NSCC-PN 具有前进性

无论从可达树 NSCC-PNWT 还是 NSCC-PNRT 都可以看出,任意状态之间没有出现循环,因而任一点火都将系统从初态逐步地推向相应的终态。相应地,算法在执行过程中不会出现无意义的动作。

因此，同时具备上述五个性质的 Petri 网所描述的 NSCC 协议是正确的。

3　结束语

根据实时数据库系统的实时特性,提出了一种新可推测并发控制协议。该协议有助于提高系统资源利用率,增强事务实时性。对 NSCC 协议进行了模拟实验,通过实验证明,NSCC 协议有助于提高实时数据库系统事务处理效能。

参考文献:

[1] Bestavros A,Braoudakis S.A family of speculative concurrency control algorithms.Technical Report TR-92-017[R].Computer Science Department,Boston University,Boston,MA,1992-07.

[2] Taniar D,Goel S.Concurrency control issues in grid databases[J]. Future Generation Computure Systems,2007,23:154-162.

(下转 241 页)