维普资讯 http://www.cqvip.com

分布式数据库 混合调度 协议 数据库 ⑨

计算机科学 1994 Vol. 21 №. 6

43-47,51

一种混合调度协议

程 勋 楼荣生 施伯乐

(复旦大学计算机系数据库组 上海 200433)

TP311.13

摘 要 We present a new integrated scheduler which uses combination of non-two-phase locking and SGT to ensure that transactions are processed in a SR manner. It can produce non-two-phase locking history and allows the serialization graph to grow in any direction, so it has more concurrency than pure scheduler whether 2PL or SGT. We first outline 2PL and SGT protocol, then give an initial non-2PL protocol, and at last we give the whole integrated protocol. We also prove the correctness of the new protocol which preserves serializability and illustrate some properties of the new protocol.

关键词 2PL, SGT, serializability, locking, serialization graph, integrated scheduler, history, conflicting operation, transaction.

一、前言

调度器(Scheduler)是分布式数据库中事务并发控制的一部分，它接收事务管理器(TM)发来的操作，实现操作序列的可串行化。根据串行化理论的计算经历论[1]，经历H是可串行化的当且仅当H的串行图SG(H)是无环的。以下给出一个事务系统的模型，本文以下的论述均是在此模型基础之上的。

一个数据库D是数据项的集合$\{X_1, X_2, \cdots, X_m\}$。一个事务系统是操作于D上的事务集合$\{T_1, T_2, \cdots, T_n\}$。一个事务的读集(写集)是指此事务要读(写)的所有数据项的集合。一个事务要读(或写)一个数据项X，则通过事务管理器向调度器发一个读(或写)请求。调度器负责决定是否执行相应操作，判定标准是要生成可串行化的经历。据此，调度器可能允许、拒绝或延迟请求。一个事务的经历是指此事务向调度器发出的成功的读写请求的序列，记作读写操作的序列($R_i(X)$(或$W_i(X)$)表示事务T_i读(或写)数据项X)。假设在一个事务的经历中对每一数据项至多只有一个读或写操作，且假设如$R_i(X)$、$W_i(X)$均属于T_i的经历，则$R_i(X)$总在$W_i(X)$前。这种假设是合理的，因为我们只需第一次读某数据项时将其读入事务的局部缓冲区，以后的相同读操作只需从局部缓冲区中读取，类似地，对同一数据项的写操作，除了最后一个外其余的均可写到局部缓冲区，最后一次才写到数据库中。调度器的一个经历是指由它允许的读写操作的序列，例如，三个事务的经历及它们的一个调度器经历如下：

T_1: $R_1(X)$ $R_1(Y)$ $W_1(Y)$

T_2: $R_2(Y)$ $W_2(Y)$

T_3: $R_3(X)$ $R_3(Y)$ $R_3(Z)$ $W_3(X)$

经历: $R_1(X)$ $R_2(Y)$ $W_2(Y)$ $R_3(X)$ $R_1(Y)$ $R_3(Y)$ $W_1(Y)$ $R_3(Z)$ $W_3(X)$

其中要求如事务经历中$A_n(X)$在$A'_n(X')$之前，则在其参与的调度器经历中也要满足$A_n(X)$在$A'_n(X')$之前。

调度器一般分为两类：纯调度器(Pure Scheduler)和混合调度器(Integrated Scheduler)。纯调度器主要采用以下三种调度机制之一或其变种：2PL、TO或SGT，它们对于经历中的读写冲突与写写冲突采用的是同一种调度机制，故称其为纯调度器；混合调度器对于经历中的读写冲突与写写冲突分别采用不同的调度机制，但分别调度的结果必须保证整个经历的可串行化。本文给出的一种混合调度协议结合了两种调度机制：2PL与SGT，但并没有区分读写冲突和写写冲突，两种冲突都必须通过2PL及SGT的检查。由于二者的协同检查，可在减弱纯

程 勋 研究生，研究兴趣：数据库、并行处理及人工智能。楼荣生 副教授，研究兴趣：关系数据库、OODB、分布式数据库、MIS。

维普资讯 http://www.cqvip.com

2PL 及纯 SGT 条件的情况下仍保证串行性，从而提高了并行性。

二、2PL 和 SGT 调度器的简介

2PL（二阶段封锁）协议为大多数封锁调度协议的基础。该协议规定一个事务内所有加锁操作在所在撤锁操作之前，它的主要内容有以下三条：

①对所有事务 T_i 和数据对象 X，$L_i(X)<O_i(X)<U_i(X)$，其中 O_i 表示 R_i 或 W_i，L 表示加锁，U 表示撤锁。

②若 $O_i(X)$ 和 $O_j(X)$ 是冲突操作，则 $U_i(X)<L_j(X)$ 或 $U_j(X)<L_i(X)$。

③对任何提交事务 T_i，$L_i(X)<U_i(X)$。

以上三条决定了集中式数据库中调度的可串行化。对于分布式数据库，为了保证在全局范围内同时撤锁，又加了第四条，成为严格的 2PL 协议。

④对任一事务 T_i 和数据项 X 有 $A_i<U_i(X)$ 或 $C_i<U_i(X)$，其中 A 为中止（abort），C 为提交（commit）。

严格的 2PL 协议可以保证调度序列的可串行化、可恢复性、无链锁中止性，其产生的经历是严格经历[1]。

2PL 协议最大的优点在于其简洁，但条件很强，并行性差。当一个事务 T_i 有很多操作与前面的事务中的操作发生冲突时，T_i 的这些操作必须等到前面的事务提交后方可执行。2PL 协议的另一缺点是死锁，死锁防止了不可串行化的经历，但其严重影响了系统的性能。

SGT（串行图测试）协议在调度器中生成并维护一个存储串行图（SSG[1]），调度器每收到一个操作就修改 SSG 并测试无环性。具体做法如下：

插入：当接收到操作 $O_i(X)$ 时进行。若 SSG 中不存在事务 T_i，则增加结点 T_i；若 SSG 中存在结点 T_j 且有一操作 $O_j(X)$ 与 $O_i(X)$ 冲突，则增加弧 $T_j \rightarrow T_i$。

测试：若新加的弧造成环，则拒绝操作 $O_i(X)$ 并令事务 T_i 撤回。

经上两类动作已能使调度器产生串行调度。为了不使存放在 SSG 中的结点越来越多，应把不再使用的结点删去。删除规则如下：每当一个事务已结束且其在 SSG 中是源结点（即入度为 0），则删去该事务的结点及与其相关的弧。

SGT 协议中 SSG 的生长方式有一特点，即一事务加入 SSG 时总是位于另一事务的后面（位于第一条弧的顶端），我们把这种现象称为 SSG 的单向生长。

三、初步协议

由于 2PL 协议的上述缺点，人们在其基础上改进而衍生了许多封锁调度。一类封锁协议称为二版本协议，这类协议本质上仍是 2PL 的，它们通过数据的“前值”（旧值）、“后值”（新值）来增加 2PL 协议中读写的并行性。这类协议的特点是对事务的读操作不延迟，马上允许执行，真正的写操作（写入数据库）则延迟到事务提交时，所以读操作能读出另一事务将要写的数据的“前值”，相对地，写事务生成数据的“后值”。此类协议典型的例子是 2V2PL 协议（two-version two-phase locking protocol）。这类协议之所以本质上仍是 2PL 的，是因为读锁要延迟到事务提交时才释放，写操作要等到被写数据上的读锁均释放后才能提交。这样，写事务仍要等待读事务结束，从而降低了并行性，并可能造成写事务的“饥饿”现象，即无限延迟。

下面先给出一个初步的非 2PL 的封锁协议，它利用了上述二版本策略来提高并行性，且其封锁策略是非二阶段的。

把一个事务 T 要操作的数据项集分成读集（Rd）和写集（Wr），读集中是 T 要读的数据项，写集中是 T 要写的数据项，Rd(T) 与 Wr(T) 可能相交。每个事务在发出第一个操作请求前，必须告诉调度器其读集与写集。这一信息可通过查询编译器或前端处理器获得。当调度器得到事务的读集与写集后，它首先请求在读集中所有数据的读锁，并将这些数据从数据库读入局部缓冲区。此时按照 2PL 协议，这些读锁一定得保持到事务提交，但这样显然阻塞了其它写事务，既然我们已将数据读完了，不妨立刻释放这些读锁（导致了非 2PL）。同时，我们将对写集中的数据项请求写前锁，此锁的设置是为了防止其它事务也同时修改此数据而造成修改丢失。不过读锁与写前锁是不冲突的，即允许其它事务能在写操作提交之前读这些数据的“前值”。在提交时，写前锁将被升级为独享的写锁，数据将被写入数据库（“后值”取代“前值”），以上三种锁的关系表示在表 I 中。

表 I　锁的冲突表

请求锁	已有锁		
	读　锁	写前锁	写　锁
读　锁	不冲突	不冲突	冲　突
写前锁	冲　突	冲　突	冲　突
写　锁	冲　突	冲　突	冲　突

我们将上述的初步协议表示如下：（LOCK（读，X）表

维普资讯 http://www.cqvip.com

示请求在数据项 X 上加读锁，如冲突则等待直到获得。其余符号类似）

(1)获得写前锁。

```
for all X∈Wr(T) do
begin
    LOCK(写前,X)
end
```

(2)获得读锁。

```
for all X∈(Rd(T)-Wr(T)) do
begin
    LOCK(读,X)
end
```

(3)将读集中的数据项读入局部缓冲区，然后释放读锁。

(4)事务 T 开始处理。

if T 发出 read(X)，则返回局部缓冲区中的 X 值

if T 发出 write(X)，则写 X 入局部缓冲区

(5)事务 T 处理结束。

将事务 T 所有的写前锁升级为写锁

将局部缓冲区中的数据写入数据库

释放事务 T 拥有的所有锁

(6)事务 T 终止。

上述协议的优点在于读锁的早释放及写锁的晚申请，是非二阶段的封锁协议，因为它在获得写锁之前释放了读锁。但正因为如此，此初步的协议还不能被接受，它可能产生不可串行化的调度经历。如假设 T_1 正在运行，它已释放了所有读锁。现在另一事务 T_2 可以修改一个已被 T_1 读过了的数据项，从而使 T_2 在等价串行经历中位于 T_1 之后。假设 T_1 又对某一数据项 Y 设置了写前锁，同时 T_2 也可读 Y（设置读锁），从而使 T_2 在逻辑上又位于 T_1 之前。由此产生了一个不可串行化的经历。

在诸如 2V2PL 这类协议中上述非串行化经历的消除（产生死锁）是通过 2PL 实现的，但现在这协议是非 2PL 的，所以我们要依靠已知事务的读集和写集提供的信息来排除类似的不可串行化经历而保证算法的正确性，即保证生成可串行化的经历。方法是融合 SGT 协议的方法，即让调度器同时也生成并维护一个存储串行图（SSG），并增加一个验证阶段。

四、完整协议

完善后的整个混合调度协议如下：（LOCK（读，X）表示请求在数据项 X 上加读锁，如冲突则等待直到获得。写前（T_i）表示已被事务 T_i 加了写前锁的数据项集合。其余符号类似。调度器为 SSG 中的每个结点维护两个数据项集：读集（T）和写集（T）。）

(1)Arrival Point（事务 T 到达）

①获得写前锁

```
for all X∈Wr(T) do
begin
  LOCK(写前,X);
  若SSG 中无结点 T,则将结点 T 加入
    SSG,令读集(T)=写集(T)=φ;
  若X∈读集(Ti),则在 SSG 中加弧 Ti
    →T;
  若X∈写集(Ti),则在 SSG 中加弧 Ti
    →T;
end;
```

②获得读锁

```
for all X∈(Rd(T)-Wr(T)) do
begin
  LOCK(读,X);
  若SSG 中无结点 T,则将结点 T 加入
    SSG,令读集(T)=写集(T)=φ;
  若X∈写集(Ti),则在 SSG 中加弧 Ti
    →T;
  若X∈写前(Ti),则在 SSG 中加弧 T
    →Ti;
end;
```

③验证

```
if(SSG 有环)
then
    begin
    释放 T 的所有锁;
    将结点 T 及相关的弧从 SSG 中删
      除;
    重新调度 T;
end
```

(2)Initial Locked Point(ILP)

①将 Rd(T)中的数据读入局部缓冲区

②令读集(T)=Rd(T)-Wr(T)

③释放事务 T 拥有的读锁

(3)Start Point

①事务 T 开始处理

if T 发出 read(X)，则返回局部缓冲区中的 X 值

if T 发出 write(X)，则写 X 入局部缓冲区

②事务 T 准备提交。

维普资讯 http://www.cqvip.com

③将事务 T 所有的写前锁升级为写锁

(4)Final Locked Point(FLP)

①令写集(T)=Wr(T)

②将局部缓冲区中的数据写入数据库

③释放事务 T 拥有的所有锁

④对每一个已结束的事务(包括 T),若其在 SSG 中是源结点(即入度为0),则删去该事务的结点及与其相关的弧。

⑤重复步骤①,直至没有满足上述条件的事务

(5)Termination Point(事务 T 结束)

下面我们来看完善后的混合调度协议是如何防止初始协议生成的那个不可串行化经历的。一个事务 T_1 已将其只读集数据项读入局部缓冲区并释放了读锁,同时在其即将写的数据项上具有写前锁。假设此时 T_2 到达并要在 T_1 的只读数据项上执行写操作,即要设置写前锁,由协议,SSG 中应添加弧 $T \rightarrow T_1$;如同时,T_2 又要读入一个被 T_1 设了写前锁的数据项,由协议 SSG 中又应添加弧 $T_1 \rightarrow T$,但这样在 SSG 中出现了环 $T \rightarrow T_1 \rightarrow T$,从而 T_2 将通不过验证而被延迟或中止退回。这样就避免了此非串行化经历的产生。

下面我们再举一较复杂的例子。三个事务 T_1、T_2、T_3 的各自经历如下:

T_1: $R_1(X)$ $W_1(Y)$

T_2: $W_2(X)$ $W_2(Z)$

T_3: $R_3(Y)W_3(Z)$

[time→]

在混合调度协议下的一个执行顺序如表 I 所示。

表 I 混合调度协议下的一个执行过程

T_1	T_2	T_3	SSG
到达, 写前封锁 Y 读封锁 X 读 X 释放 X 上的读锁。			T_1
	到达, 写前封锁 X 写前封锁 Z 修改 X 修改 Z 结束		$T_1 \rightarrow T_2$
正在执行 …		到达, 写前封锁 Z 读封锁 Y	$T_1 \rightarrow T_2 \rightarrow T_3$ $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ (验证失败。)
		撤回	$T_1 \rightarrow T_2$

五、协议的正确性及特性

本文给出的混合调度协议具有很好的事务调度特性,其生成的经历是严格经历,具有可恢复性、无链锁中止性,并可避免死锁。下面在证明了协议的正确性(即达到了串行化)后,将简要说明这些特性。

为了证明协议保证事务调度的可串行化,我们在读写冲突、写读冲突、写写冲突的基础上定义一个优先关系→。串行图是此优先关系对应的有向图的子集,因此此优先关系的无环性意味着串行图的无环,从而保证了可串行性[1]。我们需要证明上述优先关系均能正确反映在协议维护的 SSG 上,这样 SSG 的无环性即保证了优先关系的无环性。所有在→中的事务将假定是那些已通过了验证阶段的事务,因为那些还没通过验证阶段的事务对数据库并不产生任何影响,我们在正确性证明中将不考虑这些事务。定义上述事务集上的优先关系如下:两个事务 T_i 与 T_j 有优先关系 $T_i \rightarrow T_j$,当且仅当①T_i 读 X,后来 T_j 写 X(读写冲突),或②T_i 写 X 提交后,T_j 读 X(写读冲突),或③T_i 写 X 提交后,T_j 也写 X(写写冲突)。

以下先给两个引理,然后给出正确性定理。

引理1 关系 $T_i \rightarrow T_j$ 发生 iff 以下情况之一发生:

(1)T_i 在 X 上获得一个读锁,然后当 T_i 释放读锁后,T_j 在 X 上获得一个写前锁。这时在 SSG 中具有弧 $T_i \rightarrow T_j$。

(2)T_i 在 X 上获得一个写前锁,当 T_j 保持此锁时,T_i 在 X 上获得一个读锁。这时在 SSG 中具有弧 $T_i \rightarrow T_j$。

(3)T_i 在 X 上获得一个写前锁,当 T_i 释放 X 上的锁后,T_j 在 X 上获得一个读锁。这时在 SSG 中具有弧 $T_i \rightarrow T_j$。

(4)T_i 在 X 上获得一个写前锁,然后 T_j 在 X 上也获得一个写前锁。这时在 SSG 中具有弧 $T_i \rightarrow T_j$。

证明:根据协议很简单但比较冗长。 □

引理2 协议阶段(4)(FLP)中步骤①,⑤删去的结点及弧不会在以后的验证阶段成为环的组成部分。

证明:由协议可知在 FLP 阶段中步骤①、⑤删去的结点所代表的事务已结束,因此该事务无锁,包括写前锁。因此此事务以后只可能参与获锁阶段的前三种弧的生成,而这三种弧均是以该事务为源的,又因为该事务对应的结点本是源结点(删除时),所以以后该结点的入度一直为0,从而不可能参与环的生成。既然结点不是环的一部分,与其相连的弧亦不

维普资讯 http://www.cqvip.com

可能成为环的一部分。 □

定理1 协议保证优先关系的无环性，从而保证了其生成的经历是可串行化的。

证明：由引理1与引理2可知 SSG 的无环性保证了优先关系的无环性，而后者保证了串行图的无环，从而保证了经历的可串行化。 □

除了具有可串行性外，该混合调度协议还具有以下一些特性：

(1)可恢复性：为保证在有故障发生时的计算正确性，调度不仅是要可串行化的，而且在有事务撤回时还是可恢复的。可恢复性要求：如果 T_i 读了 T_j 写入的数，则 T_j 应比 T_i 先提交。显然由于本混合调度协议的实际写操作(写锁的设置)是在提交时完成的，故当一事务读 T_j 写入的数据项时，T_j 必已提交完成结束。

(2)无链锁中止性：要求经历 H 中若有 T_i 读 T_j 的数，则 T_j 在 T_i 的读操作之前提交。同上理，本混合调度协议生成的经历具有无链锁中止性。

(3)严格经历：要求经历中若有 $W_j<O_i(X)(i\neq j)$，则必有 $C_j<O_i(X)$。同上理，本混合调度协议生成的经历是严格经历。

由上述特性可见，本协议在提高了并行性的同时，仍保留了严格2PL 的优良特性。

该协议还有一个很好的特点是可避免死锁。在2PL 协议中死锁是保证串行化的重要手段，但死锁检测、消除的开销是很大的，尤其在分布式数据库中。

死锁是由于 LOCK 冲突而循环等待造成的。这类死锁在本协议中可利用读集与写集来避免。我们类似 OS 中解决资源申请死锁的方法，对资源(本协议中为数据项)定义一个优先级次序，所有加锁操作必须按此次序进行(按优先级由高到低申请)。首先，所有的写前锁是按此次序获得的，然后是所有读锁，写前锁升级为写锁也是按此次序进行的，称此为顺序封锁规则。我们有如下定理：

定理2 上述顺序封锁方法在协议中是不会造成死锁的。

证明：用反证法。假设上述顺序封锁方法在协议中是会造成死锁的。不失一般性，设造成死锁的等待链为 $T_1\rightarrow T_2\rightarrow\cdots\rightarrow T_k\rightarrow T_1$，由于顺序封锁方法，上述等待链中的事务不可能均在等待获得同一类锁(本协议中共有三类锁)。假设上述事务均不在获得写锁阶段等待写锁，则必有一事务在获得读锁阶段在 X 上等待读锁，而 X 上已有写前锁，但由协议读锁与写前锁并不冲突，所以上述等待链中必有事务在获得写锁阶段在 X 上等待由写前锁升级为写锁，不妨设为事务 T_1。又因为写前锁之间是互斥的，所以在 X 上必是有其它事务拥有的读锁阻塞了 T_1，不妨设为事务 T_2。同样由于写前锁之间的互斥，T_2 必在获得读锁阶段在 Y 上等待读锁，因为读锁只与写锁冲突，所以在 X 上必是有其它事务拥有的写锁阻塞了 T_2，不妨设为事务 T_3。以此类推可知，T_1，T_3，T_5，…处于获得写锁阶段，T_2，T_4，T_6…处于获得读锁阶段，k 必为偶数。不失一般性，我们来考察具有上述特性的等待链 $T_1\rightarrow T_2\rightarrow T_1(k=2)$，其中 T_1 处于获得写锁阶段，设其在 X 上等待获得写锁，但 T_2 在 X 上已拥有读锁而阻塞了 T_1，另外 T_2 正处于获得读锁阶段且在 Y 上等待获得读锁，但 T_1 在 Y 上已拥有写锁而阻塞了 T_2。由顺序封锁规则可知，因 T_2 先获得了 X 上的读锁，再申请 Y 上的读锁，所以 Y 的优先级必比 X 低，但 T_1 还没得到 X 上的写锁(T_2 阻塞了它)，故 T_1 不可能在 Y 上已拥有写锁(T_1 必须先获得 X 上的写锁才能申请 Y 上的写锁)，即 T_1 不可能阻塞 T_2。故由顺序封锁规则可知具有上述特性的等待链是不可能存在的，这与假设存在等待链矛盾。 □

由于事务因验证失败而延迟(或退回)，协议有可能产生活锁(livelock)或"饥饿"现象，从而不能保证一个被中止事务最终一定会被提交完成。我们给出如下一个解决方法。当一个事务 T 多次由于验证失败或死锁而被中止，则将此事务的写集扩充为包含其读集，这样一来 T 下次执行时 SSG 只可能单向生长而不会形成环，于是 T 一定能通过验证，由此保证了 T 的运行直至提交终止。

六、本协议与已有协议的对比

与2PL 协议相比，在本协议中读锁和写锁均只在其必要时出现，从而避免了2PL 协议中额外的等待，增加了并行性。只有部分共享性的写前锁存在的时间几乎相当于整个事务的生存时间。而且上述协议的算法中仅验证阶段的测试、对 SSG 的修改和 LOCK 过程必须保证操作上的原子性，其余均可并行执行。

在2PL 及 SGT 协议中，当一个事务对某一数据项加锁时，它总是将自己在串行图上放于其它事务之后，从来不会放在前面，我们将此现象称为串行图的单向生长，即冲突关系只能往前长。事实上，2PL 正是利用此性质来达到串行化的经历。而在本文给

(下转第51页)

维普资讯 http://www.cqvip.com

别[10]。

除了数据模型的不同，DBMS 在系统级的差异也是研究者们必须面对和解决的问题。系统级的差异涉及：事务管理的原语和技术（包括并发控制、提交协议和恢复）、硬件和系统软件的需求以及通信能力。

3）解决语义差异所带来的问题。由于网络上各数据库应用系统的开发往往是独立的，对相同或相关数据的意义或解释的不一致是不可避免的。这便会发生语义上的差异。下面的例子可以说明这个问题。

考虑数据库 DB_1 中关系 RESTAURANT 的属性 MEAL_COST，它表示餐馆中每个客人一次进餐的平均费用，但不含服务费和税。而数据库 DB_2 中关系 BOARDING 的同一属性 MEAL_COST 所表示的每个客人一次进餐的平均费用却包含了服务费和税。设两个属性具有相同的语法性质。将 DB_1. RESTAURANT. MEAL_COST 和 DB_2. BOARDING. MEAL_COST 进行比较是没有意义的，因为它们的语义有差异。

检查语义差异相当困难。通常，DBMS 模式没有提供对数据进行一致解释的足够语义。数据模式的不同也增加了识别和解决语义差异的困难程度。

除了以上提到的几个重要问题之外，在具体实现时还可能遇到这样那样的问题，如数据完整性等，研究与开发者们需要付出巨大的努力。

6 结束语

数据库互操作性所要达到的目标很有现实意义，吸引了越来越多的研究者。然而，所面临的问题对研究者们又是一种严峻的挑战。人们正在从各个方面采取各种途径，为最终实现完全可互操作的数据库系统而努力着。由于篇幅所限，关于数据库互操作性的实现方法在另一篇文章中讨论[18]。（参考文献共15篇略）

（上接第47页）

出的混合调度协议中，冲突关系可在前后两个方向上"生长"，即表现为 SSG 可双向生长。假设一个事务 T 已到达 ILP 且拥有一个写前锁，然后一个新事务 τ 到达，并在同一数据项上获得了一个读锁，当 τ 到达 ILP 时，关系 τ→T 建立了。现在即使 T 提交终止了，只要 τ 还活跃着（未到达 FLP），另一事务 τ' 仍可能到来被放于 τ 之前，从而也就在 T 之前。通过这种方式，将来的事务有可能在逻辑上处于过去。虽然已存在的二版本封锁协议如2V2PL 协议也允许一个事务读"前值"，但那些协议没摆脱2PL 的限制，写事务仍要等待读事务的提交，从而无法在两个方向上"生长"。之所以有这么个区别，本质上是由于本协议与2PL、SGT 协议在保证串行化的机制上不同。在本协议中，串行化是通过混合调度实现的，综合了不同机制的特点，由此可生成非2PL 的经历。例如下述非2PL 经历便是本协议允许的：

经历：$R_1(X)$ $R_2(Y)$ $W_1(Y)$ $R_3(Z)$ $W_2(Z)$

串行顺序：$T_3 \rightarrow T_2 \rightarrow T_1$。

七、结论

本文给出的混合调度协议尽管比纯协议（2PL 及 SGT）复杂得多，但它还是值得研究的。串行性是经历的一种句法特性，判断一个经历的可串行性是一个 NP-完全问题。所以现有的调度算法均只能产生可串行化经历的一个子集，对此子集的判定是有效的。无论2PL、SGT 及本协议均是如此。但本协议在调度中更宽容些（相对允许更大的并行性），且完全有可能通过对读集、写集提供的信息作进一步的分析来识别允许更多的可串行化经历，进一步提高并行性。

参考文献

[1] Philip A. Bernstein, Vassos Hadzilaco, Nathan Goodman, Concurrency Control And Recovery in Database Systems

[2] P. Dasgupta and Z. M. Kedem, The Five Color Concurrency Control Protocol: Non-Two-Phase Locking in General Databases, ACM TODS, Vol. 15, No. 2, June 1990

[3] 瞿兆荣，分布式数据库，华东计算技术研究所，1989年

[4] K. Vidyasankar, A Non-Two-Phase Locking Protocol for Global Concurrency Control in Distributed Heterogeneous Database Systems, IEEE transactions on knowledge and data engineering, Vol. 3, No. 2, June 1991