データベース・システム 54－6
（1986. 7. 21）

後退復帰対象を動的に決定する時刻印方式

仲 興国　上林 弥彦
(九州大学 工学部)

分散データベースシステムにおける代表的な並行処理制御方式として、時刻印方式が知られている。従来の時刻印方式では、操作衝突が起こった場合、必ず時刻印の小さい(従って先に生成された)処理単位が後退復帰される。そのため、従来の時刻印方式には、(1) 後退復帰コストが高い、(2) 部分後退復帰が不可能であるなどの問題が存在する。本稿では、それらの問題を指摘すると共に、操作衝突時にできる限り時刻印の大きい処理単位を後退復帰する方式を提案する。この方法はデータに対する操作履歴の記述を用い、それにより処理単位間の依存関係を表現する。最後に二相施錠方式、従来の時刻印方式および本稿の時刻印方式の比較を行う。

A Timestamp Ordering Mechanism with Dynamic Selection of Rollback Objects

Xingguo Zhong and Yahiko Kambayashi
Dept. of Computer Science and Comm. Eng., Kyushu University

Timestamp ordering is one of the typical methods in concurrency control for distributed database systems. In conventional mechanisms based on timestamp ordering, among conflicting transactions the transaction which has the smallest timestamp (the oldest transaction) is always rollbacked. By this reason, there are the following problems; (1) rollback cost is greater, (2) it is impossible to realize partial rollback. In order to overcome these problems, in this paper we have developed a new timestamp ordering mechanism that rollbacks the transaction that has the greatest timestamp whenever possible. This mechanism utilizes the description of the operation histories for data, by which the dependent relationships between transactions are shown. Comparisons among two-phase locking, conventional timestamp ordering based mechanisms and the new mechanism proposed in this paper are discussed.

1. まえがき

データベースシステムにおいて並行処理制御は重要な課題となっている。並行処理制御の目的はデータベースを操作する複数の処理を並行に実行させて、データベースの一貫性を保障した上でシステムの性能、効率性、応答時間均一化などをできる限り達成することである。その場合、データベースおよび利用者に対する効果が各処理を逐次に実行したときと同様であることを保障する必要がある。

並行処理制御をその判定条件によって分類すれば、主に二相施錠方式[EG76]と時刻印方式[RE78、BG80]がある。分散データベースでは、二相施錠方式におけるすくみの検出に各局間の通信コストが高いという問題がある[OB82、MM79、GS80]。そのため、時刻印方式がよく利用されている[BS80]。時刻印方式では、各局で起きた操作衝突がその局で局所的に判定できることが一つの特徴である。しかしながら、従来の時刻印方式には、操作の衝突が起こったときに必ず時刻印の小さい(従って早く実行始めた)処理単位が後退復帰されるため、次のような問題が起こる。

(1) 実行した部分が大きい処理単位を後退復帰するため後退復帰のコストが高い。

(2) 部分後退復帰の実現は原理的に不可能となっている。

本稿では、それらの問題を指摘し、それらに対処できる新しい時刻印方式、即ち、後退復帰対象を動的に決定する時刻印方式を提案する。この方式は従来の時刻印方式と比較して、分散データベースの場合に大幅な効率向上が期待できると考えられる。また、後退復帰の連鎖も効果的に扱うことができる。以下2節では、並行処理の基本的概念と従来の時刻印方式を簡単に紹介する。3節では、それに存在する問題点をより詳細に指摘する。4節は後退復帰対象を動的に決定する時刻印方式について述べる。5節では、よく知られている二相施錠方式、従来の時刻印方式および本稿で提案した時刻印方式との比較を行う。

2. 従来の時刻印方式

並行処理制御を議論するには、一般にデータベースをデータの集合と考える。一つのまとまった論理的な処理を処理単位という。処理単位のデータに対する基本操作には、読み出しおよび書き込みがあるとする。データベースは利用者から見たときに、いつでも一定の一貫性制約を満たしている。一つの処理単位はデータベースに対して一貫した状態から一貫した状態への変換を行う。従って、複数の処理単位を直列(順次)に実行したときデータベースは一貫した状態から一貫した状態になることが判る。処理単位を並行に実行し、その結果がそれらの処理単位を直列に実行したときと等価であればデータベースは常に一貫した状態になる。そのような実行スケジュールは直列可能であるという[EG76]。並行処理制御は本質的には、処理単位の直列可能性を保障することである。各種の並行処理制御方式はそれぞれ一定の規約を用いて、処理単位の直列可能性を保障する。その規約に違反すると、操作の衝突と判定する。そのとき衝突する一方の処理単位の実行を停止し、既に実行した部分を無効にする。それを処理単位の後退復帰という。

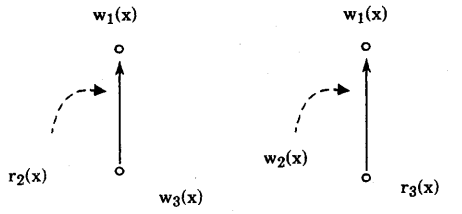
従来の時刻印方式[BG80]では、各処理単位の生成される順に唯一の時刻印を付けておく。二つの処理単位が同一のデータに対して、操作を行うときに、必ず先に生成した(従って、時刻印の小さい)処理単位が先に操作を行う。このような制約の元で処理単位の実行を制御すると、システムにおける処理単位の実行スケジュールは直列可能になる。その条件を満たさないときに、衝突した処理単位を後退復帰する。後退復帰された処理単位は新たな時刻印を与えられ、再び実行される。次の二つの時刻印方式が代表的である。

(1) 基本的時刻印方式

各処理単位T_iがその到着順に唯一の時刻印t_iを割り当てられる。データxに対して読み出し時刻印$t_r(x)$と書き込み時刻印$t_w(x)$を設定する。$t_r(x)$はxを読み出した処理単位中の最大の時刻印である。$t_w(x)$はxを書き込んだ処理単位の最大時刻印である。T_iがxに対して読み出しを行うとき、その時刻印t_iをデータxの書き込み時刻印$t_w(x)$と比較する。$t_w(x)<t_i$のとき、読み出すことができる。読み出し操作が行われると$t_r(x)$を$t_r(x)$とt_iの大きい方で置き換える。$t_w(x)>t_i$のときに読み出すことができず、T_iが後退復帰される。データxに対して書き込み操作を行うとき、T_iの時刻印t_iをデータxの$t_r(x)$および$t_w(x)$を比較する。$\max(t_r(x), t_w(x))<t_i$ならば書き込みができる。書き込みを行うと$t_w(x)$をt_iで置き換える。$t_i<t_w(x)<t_r(x)$或いは$t_r(x)<t_i<t_w(x)$のとき、書き込みを無視することができる(トーマスの書き込み則)[TH79]。その他の場合は操作の衝突となり、T_iを後退復帰する。

(2)　多重版時刻印方式

この方式は(1)の時刻印方式を基礎とし、各データxに対する書き込み操作は新たな版を生成する。データxの過去の値を多重版として持たせる。[RE78、BG80]。データxの各版にはそれぞれ書き込み時刻印と読み出し時刻印を持たせている。書き込み時刻印はそれを生成した処理単位の時刻印である。読み出し時刻印はそれを読み出した処理単位の内、最大の時刻印となる。T_iがxを読み出すとき、$t_i > t_w(x)$を満たすxの版の内最大の書き込み時刻印$t_w(x)$を持つ版を選択する。xを書き込むときに最新版の時刻印と比較して、$t_i > \max(t_r(x), t_w(x))$ならば書き込みができ、データxの新しい版を生成する。書き込みができないとき、T_iが後退復帰される。多重版方式の一つの特徴として、読み出し操作による衝突が生じない。衝突が起こるのは書き込み操作を行うときだけである。図1には、



(a) 読み出しによる矛盾がない　(b) 書き込みによる矛盾がある

図1.　多バージョン方式による操作矛盾例

T_1、T_2、T_3を順次に生成された処理単位とするときの例を示している。$w_i(x)$、$r_i(x)(i=1,2,3)$はそれぞれT_iのxに対する書き込み操作と読み出し操作を表している。iが大きいほど対応する時刻印も大きいと仮定している。従って、上から下へ操作の時刻印が小さいものから大きいものの順に並んでいる。

処理単位の実行にあたって、それが終了するまではその書き込み操作をデータベースに反映させないことにする。書き込まれるデータを読み出す処理単位はその反映を行うまで待たせる。従って、一つの処理単位の後退復帰が他の処理単位の後退復帰を引き起こすといった後退復帰の連鎖は生じない。

分散データベースでは処理単位の終了時に二相認証[GR78]を行わなければならない。二相認証とは、処理単位の終了時に、予備認証と認証との二段階を設定しておくことである。次の二つの性質がある。

(1) 予備認証した書き込み操作は認証または後退復帰のいづれの方向へも変更可能である。

(2) 処理単位全体を予備認証したという時点になると必ず認証までに行ける。

3. 問題点

上記の時刻印方式では、操作の衝突が起こったときに時刻印の小さい(従って早く生成された)処理単位が後退復帰される。そのため、次の問題が生じる。

(1)　まず、後退復帰のコストが高いという問題である。後退復帰のコストは一般に後退復帰される処理単位の実行した部分の再実行コストで評価する。実行した部分が大きいほど回復に要するコストも大きい。分散データベースの場合、回復に要するコストは実行した部分に含まれる通信コストによる。実行した部分が大きいほど、通信コストも大きくなる。衝突する二つの処理単位には平均して時刻印の小さい処理単位の実行した部分は時刻印の大きい処理単位の実行した部分よりずっと大きいと考えられる(図2を参照)。また、処理単位の

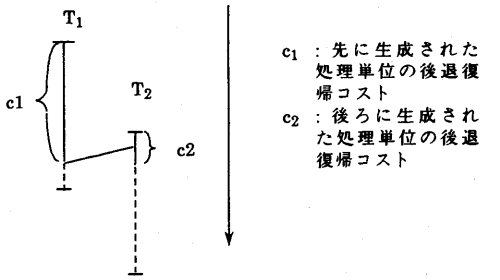


図2.　後退復帰のコスト

実行は一般に読み出し操作が先行し、次いで書き込み操作が続くことが多い。特に、書き込みと読み出しの二段階からなる処理単位を考えると、衝突の時に時刻印の小さい処理単位が第二段階にある可能性が高い。しかし、時刻印の大きい処理単位はまだ読み出し段階になっていることが多い。従って、時刻印の小さい処理単位の回復コストはより大きいことが分かる。時刻印方式における操作衝突のほとんどは図2に示したような小さい時刻印を持つ処理単位の書き込みと大きい時刻印の読み出しとの衝突であると考えられる。

(2)　衝突が起こったときに、処理単位全体ではなく、矛盾する操作を含む処理単位の一部分をやり直して済ませることは部分的後退復帰と呼ばれている[FK81]。部分的後退復帰の実現は難しいた

め、従来はあまり重視されていなかった。しかし、分散データベースの場合に処理単位の部分的後退復帰は特に重要である。何故ならば、全域的な処理単位は各局において、それぞれ一つの部分処理単位として扱われる。それらの部分処理単位は通信によりつながれる。処理単位の実行コストには(従って後退復帰のコストにも)データ通信が問題となっている。従って、一ヵ所での操作衝突はその局で局所的に扱えることが望ましい。即ち、全域的処理単位に対する部分後退復帰の実現は後退される処理単位の実行コストが減少されるとともに後退復帰のための通信は不要になる。前節で述べた時刻印方式では、矛盾が起こったときに時刻印の小さい処理単位が後退復帰されるため、部分後退復帰の実現は原理的に不可能となっている。なぜならば、時刻印の大きい処理単位が実行されてしまうため、時刻印の小さい処理単位に大きい時刻印を割り付けなければ、矛盾が解除できないからである。

4. 後退復帰対象を動的に決定する時刻印方式

以上で述べた問題に応じて、本節では新しい時刻印方式の提案を行う。この方式では、小さい時刻印を持つ処理単位をできる限り優先することにする。これによって、上記の問題が解決できると考えられる。処理単位T_iがデータxに対する書き込み$w_i(x)$を行おうとするとき、処理単位T_j ($t_i<t_j$)の読み出し操作$r_j(x)$と衝突したとする。そのときT_jを認証していなければ、T_jを後退復帰することにする。それを実現するには、データの操作履歴の記述、処理単位の制御、データ版の管理などが必要である。このような対策は集中型データベースシステムに対しても分散型データベースシステムに対しても有効である。後者に対して前に述べた理由により、特に有効であると考えられる。従って、本稿では分散型の場合を中心に考察する。

本節では、分散データベースシステムの一つの処理モデルを記述するとともに、それに基づく新しい時刻印方式の実現について述べる。

4.1 データの操作履歴

後退復帰対象の置き換えを可能にした場合には、データベースが操作するすべてのデータの操作履歴を記述しなければならない。従来の多重版方式では、データxの各版に対してそれぞれ読み出し時刻印$t_r(x)$および書き込み時刻印$t_w(x)$を記憶すればよい。時刻印の大きい処理単位を後退復帰する場合はそれだけは不十分である。

図3において、従来の多重版時刻印方式では、も

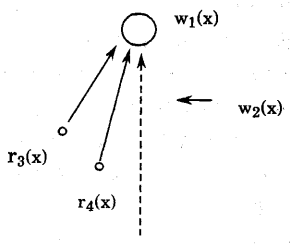


図3. データの操作履歴

しT_2による書き込み$w_2(x)$が到着したとき、$w_2(x)$と$r_4(x)$が矛盾し、T_2が後退復帰される。時刻印の大きい処理単位を後退復帰するときに、T_2の替わりにそれと矛盾する処理単位を後退復帰する。ここでは、T_3とT_4の二つである。

データxの操作履歴はxに対する操作が生じるたびに変化してゆく。図4は処理単位T_3の書き込み

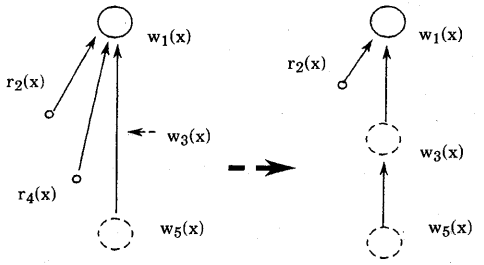


図4. 操作履歴の変更

$w_3(x)$を行うときの履歴変更を示している。その場合T_4が後退復帰されることが分かる。図4において実際に反映されていない書き込みおよび実行されていない読み出しを点線で表している。もしそのときT_1が認証されていないとすれば(図5を参照)、

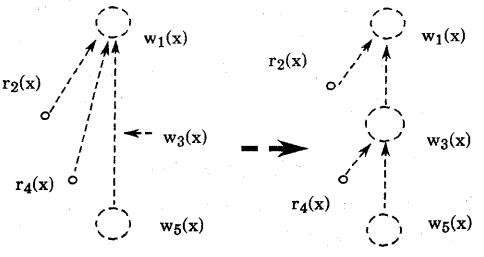


図5. 操作履歴の変更

$www_1(x)$はまだ反映されていない。その場合に$r_4(x)$はT_1の$w_1(x)$の反映を待っている。従って、T_4を後退復帰するかわりに操作履歴の変更だけで良くなる。以上により、データの操作履歴にはデータが実際に操作していたか或いは予約しているかの情報も必要となる。

データxの操作履歴は一般にxの書き込み時刻印により順につけた複数の版とそれらを参照する読み出しからなる。データベースに反映していないデータ版と既に反映した版の識別が必要である。それに対して実行した読み出しと待たせている読み出しの識別も必要である。データの最旧版はデータベースに格納されている。操作履歴はそのデータにおける処理単位の依存関係を明確にする。

4.2 システムモデル

図6は二つの局からなる分散データベースシステムの論理構成を示している。各局には

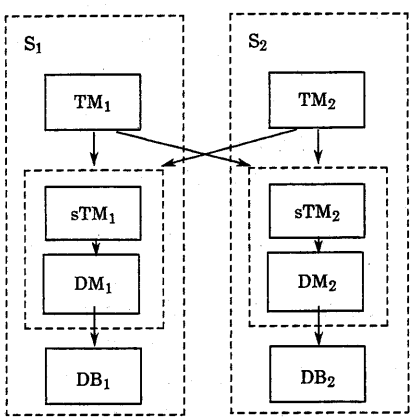


図6. 分散データベースの論理構成

TM(Transaction Manager) 、sTM(subTransaction Manager)、DM(Data Manager)およびDB (Data-Base)を持たせている。処理単位を全域的番号を付けた部分処理単位の列として考える。TMはその局で生成された処理単位の実行を制御し、各局での部分処理単位を管理する。sTMはその局で生成された部分処理単位を制御し、各部分処理単位のデータに対する操作を記述する。一つの局から生成された処理単位に対して、関連するすべての局にそれぞれ唯一の代表者(agent)を生成する。一つの処理単位が同一の局を複数回訪れるときには、複数の部分処理単位が生成される。一つの部分処理単位の内では、他の局との通信が存在しない。処理単位の実行にあたって、各部分処理単位の初期状態を処理単位の認証まで保存しておく。部分処理単位に関する記述には、それが書き込んだり読み出したりしたすべてのデータの登録がある。DMはその局におけるすべてのデータに対する操作履歴を管理し、操作時の衝突検査を行う。

一つの処理単位が生成されるときに、全域的に唯一の時刻印が所属TMにより割り付けられる。その時刻印は処理単位の識別子と一対一になっている。データの各版に対してそれを生成した処理単位の時刻印を版の時刻印として割り当てる。上述の操作履歴における読み出し操作$r_i(x)$および書き込み操作$w_i(x)$の記述には、その操作がどの処理単位のどの部分処理単位に属するかの情報を含める。

処理単位の実行時に読み出しは適当なデータ版を読み出す(2節を参照)。書き込みデータは処理単位の局所領域に一時的に置かれ、処理単位の認証時にデータベースに反映する。従って、一つの処理単位による同一のデータに対する書き込みおよび読み出し操作は二度以上行われないと仮定できる。処理単位の認証は二相認証で行われる。

4.3 操作衝突の判定と後退復帰

並行処理制御における操作の衝突は一般に、WR(Write-Read) 、WW(Write-Write) 、RW(Read-Write)に分類される。

処理単位T_iがデータxを読み出すときに、T_iの時刻印t_iより小さい時刻印を持つ版の中から最新版を読む。よって、読み出し操作による操作の衝突(WR)は生じない。読み出し操作を行う或いは読み出し操作を待たせると、xの操作履歴を変更する。

データxの操作履歴を完全に記述しているので、書き込み間の衝突(WW)は存在しない。即ち、トーマス書き込み則を使用せずに書き込み$w_i(x)$が$w_j(x)(t_i<t_j)$の後ろに来たときに、操作を行うとともにxの操作履歴を変更すれば良い。

T_iがxに対して書き込むときに、t_iより大きい時刻印を持つ処理単位がt_iより小さい時刻印を持つ版を既に読んだ或いは既に待っているときに操作の衝突(RW)となる。

読み出しがより古い版の反映を待っているときに、処理単位を後退復帰せずに操作履歴の変更だけで済ませる。読み出し操作が既に行われたときに、矛盾する両方の処理単位の一方を選択して後退復帰を行う。その場合時刻印の大きい処理単位が複数個ある場合もある(4.2節を参照)。従来の時刻印方

式では、必ず時刻印の小さい処理単位を後退復帰する。ここでは、次の二つが考えられる。

(1) 時刻印の大きい処理単位の認証を開始していなければ、その局のsTMによりそれを後退復帰する。認証しているときには時刻印の小さい処理単位を後退復帰する。

(2) 矛盾する処理単位両方の時刻印の差がある程度を超えたら時刻印の大きい処理単位を後退復帰する。時刻印の大きい処理単位が複数あるときもあるが、その場合何らかの対策が考えられる。

分散データベースシステムの場合に、複数のプロセッサーが存在する。一つの処理単位が実行しているところに、その以前の読み出しが他の処理単位の書き込みと衝突して後退復帰することがある。即ち、処理単位を実行するプロセサ間の追跡をしなければならない。その内には局を渡る追跡もある。上記の(2)は、そのような追跡に要する時間およびその間の実行コストを考慮した対策である。

時刻印の大きい処理単位を後退復帰するときには、書き込みを行う処理単位は実行し続けることになる。以下では、(1)の方法のみ考慮する。(2)に対しては、実際のシステムの通信速度などにより対策を設定すればよい。

時刻印の大きい処理単位T_iが認証されていないときに、まず衝突する操作に対する後退復帰を考慮する。操作を行った部分処理単位$T_{i,j}$($T_i=T_{i,1},T_{i,2},\ldots T_{i,n_i}$)が実行中であれば、その部分処理単位のみを後退復帰する。それは局所的に処理できる。その部分処理単位$T_{i,j}$($j\leq n_i$)が既に完了していたときには、所在sTMから処理単位を生成したTMに通知する。TMが管理している処理単位T_i中に$T_{i,j}$以後の部分を後退復帰する或いは、T_i全体を後退復帰する。

(1) 部分後退復帰の場合に対しては、後退復帰された操作は同一時刻印で再び実行されるため、操作履歴の変更の際登録された操作を除去しない。

(2) 全体的な後退復帰の場合に、処理単位のデータに対する操作を操作履歴から削除する。読み出し操作が対応するデータの操作履歴から消去される。書き込み操作が対応するデータの操作履歴から消去され、その版を待っている読み出し操作はより古い版に変更する。

4.4 操作履歴の変更

データベースにおけるデータがその操作履歴により記述される。各データの複数の版があるが、実際に存在する版は認証された版だけである。データベースに反映されていない版は履歴の記述情報だけが存在する。それは実際に書き込み操作の予約と考えてよい。操作履歴に記述された書き込みに対しては次の状態がある。

(a) 実存版 : 既にデータベースに反映されている版である。それを書き込んだ処理単位は既に認証されている。

(b) 予約版 : データは処理単位の実行により生成されていたが、処理単位が認証されていないため、データベースに反映されていない。

それに対して操作履歴における読み出しは次の二つの状態がある。

(c) 待ち状態 : 読み出す対象が実存版でなければ、読み出すことができず、読み出しがデータのデータベースへの反映を待っている。

(d) 実施状態 : 読み出し操作が実際に実行された状態である。そのとき、読み出した版が実存版でなければならない。

処理単位の処理時に書き込みを先に予約すれば、操作の衝突が減少する。処理単位の実行のときそれが読み出したデータが後で書き込まれる可能性の情報を利用し、書き込み操作を行う可能性が非常に大きければ、データの操作履歴に対して先に予約することにする。この方法によって、より時刻印の大きい処理単位の読み出し操作を待たされるため潜在的な後退復帰が無くされる。

予約した操作が実際に行われないことがある。また、部分的後退復帰の再実行時に、操作対象が変わるかもしれない。そういったときに、処理単位全体の後退復帰のときと同様に操作履歴上に操作の解除を行う。読み出しの解除はその自体を消去すればよい。書き込みを解除するときに、それを待っている読み出しがより古い版に変更する。

以上により、データの操作履歴に対する変更は次の基本操作で行うことができる。

(1) 版予約 書き込み操作を行う時に、判定条件を満たすならば、対応する版を予約する(操作履歴の変更)。判定条件を満たさないときに、後退復帰対象を決めて後退復帰を行うか或は操作履歴の変更だけで済ませるかのどちらである。

(2) 版反映 処理単位が認証されたときその版をsTMから受け取り、データベースに反映する。その版を待っている読み出し操作を引き起こし、操作履歴を変更する。

(3) 読み出し試行 読み出し操作を行うときに、適当な版を選択する。選択された版がデータベースに反映されていなければ、待つことにす

る。既に反映されているときに、直ちに(4)に行く。

(4)　読み出し実施　(3)の場合に版を反映しているとき、および(2)により読み出し操作を引き起こされたときに、読み出し操作を行う。操作履歴における読み出しの待ち状態から実施状態への変更を行う。

4.5 処理単位の認証

処理単位の実行終了時に認証を行う。そのとき、二相認証に予備認証と認証の二段階がある。

(1)　予備認証　各局においてT_iの予備認証を実行する。T_iが行った各書き込みに対してログを生成し、その書き込みが必ず達成できることを保障する。各局の予備認証が完了すると対応するTMへ返信する。T_iに関連するすべての局が予備認証の返信をした上で認証が開始できる。予備認証が完了するとT_iの認証が必ずできることになる。予備認証ができなかったとき、その原因によって後退復帰を行う。予備認証ができない原因には、(a)他の処理単位と衝突して、後退復帰されるとき、(b)システム故障により認証できなかったときがある。(a)に対して、部分後退復帰するならば、認証が遅延する。全体的な後退復帰のときには認証を中止する。

(2)　認証　　認証は必ず終了できる操作である。認証する時に処理単位T_iの書き込んだデータをデータベースに反映する。それに伴って、それらの版を読み出し可能にする。従って、それを待っている他の処理単位を復活させる。

5. 比較

本稿で提案した方式において、操作衝突の判定基準は従来の時刻印方式と同様である。それに対して後退復帰対象の動的な決定は後退復帰コストを考慮した二相施錠方式に似ている。この方式の正当性は自明である。以下本節では、二相施錠方式、従来の時刻印方式および本稿で提案した時刻印方式の比較を行う。

5.1 操作履歴の記述

従来の時刻印方式では、各データ版に書き込み時刻印および読み出し時刻印を付ける。それに対してこの方式では、データのすべての版を集中して操作履歴の記述で管理する。一つの版が複数の処理単位に読み出されたときに、そのすべてを記述する。それに多くのメタ情報がいるようであるが、実際には従来の時刻印方式にも二相認証を適用するためにデータの読み出しに対する待ちキューが必要である[BG80]。このメタ情報の量は操作履歴の記述の量とほとんど変わらない。

一方操作履歴の記述は非常に動的なメタ情報である。それにおける読み出し操作および書き込み版は処理単位の開始、終了するたびに生成、消去されてゆく。実際に特定の時点に存在する版数および読み出し操作の数(即ち操作履歴の大きさ)はデータが利用される重複度による。従って、操作履歴の大きさが大きければ大きいほど、データの利用率が高いといえる。

5.2 効果に関する比較

時刻印方式は、二相施錠方式に対して、分散データベースシステムにすくみが起こらないための対策として導入されたと考えられる。表1では、二相施錠方式、従来の時刻印方式および本稿の時刻印方式の効果的な比較を示している。

(a)　二相施錠において、実行したコスト或いは実行開始した時間による後退復帰の動的な決定ができる。我々の時刻印方式はそのような動的な決定を適用することにした。

(b)　二相施錠では部分的後退復帰が実現可能である。本稿の時刻印方式は大きい時刻印の処理単位を後退復帰するときに部分的後退復帰ができることとなった。分散データベースに部分的後退復帰の必要性を3節で述べた。

(c)　(a)と(b)とを合わせて、後退復帰のコストを減少させた。

(d)　文献[KO86]では、二相施錠方式に対して連鎖を許した場合の認証条件を記述している。それに対して従来の時刻印方式では、多重度を上げるため、後退復帰の連鎖を扱う場合、結局操作履歴を扱う必要がある。我々の時刻印方式では並行性を上げ、後退復帰の連鎖が扱えることになった。

(e)　従来の時刻印方式と同様に我々の方式にはすくみの問題は存在しない。

(f)　時刻印方式では、同一のデータに複数の書き込みを並行に行うことができる。その意味で二相施錠方式よりは時刻印方式のほうがデータ操作の並行性が高い。

(g)　二相施錠方式ではすくみの検出に要する通信コストが高い。それに対して時刻印方式では衝突の確率が大きいため後退復帰に要する通信コストが問題である。部分的後退復帰により通信コストを減らしたことは本方式の長所である。

表1　二相施錠方式、従来の時刻印方式および本稿の時刻印方式の比較

制御方式	コストによる後退復帰の選択	部分的後退復帰	後退復帰のコスト	連鎖の扱い	すくみの検出	データの利用率	通信コスト	衝突の確率
二相施錠方式	可能	可能	小さい	可能	必要	小さい	すくみの検出	小さい
本稿の時刻印方式	可能	可能	小さい	可能	不必要	大きい	減小	大きい
従来の時刻印方式	不可能	不可能	大きい	不可能	不必要	大きい	後退復帰の実行	大きい

(h) 同じ処理単位の場合、時刻印方式は二相施錠方式より衝突の確率が高い。本方式は衝突の確率が高いという問題が依然として存在する。

表1において、斜線の部分は比較的優れている特性を示している。本時刻印方式は従来の時刻印方式の長所を保持し、二相施錠方式の多くの特長も加わったことが分かる。また、時刻印を用いてすくみを避ける二相施錠方式に比較して、時刻印方式のほうが効率が良いことは(f)により分かる。

7. 結論

従来の時刻印方式に存在する問題を指摘し、それに対処できる新しい時刻印方式を提案した。この方式は後退復帰コストをできる限り減少させる方式であるといえる。局所的に扱える部分的後退復帰も可能となった。この方式において、処理単位の振る舞いは処理単位、部分処理単位およびデータに対する操作履歴の記述の三階層により制御される。処理単位間の依存関係および操作衝突の判定はデータの操作履歴により行われる。従って、並行処理制御機構の構造は明確に定義されている。

操作の衝突の対処方法から見た場合に、従来の時刻印方式は、本稿で提案した方式の特例であるといえる。即ち、操作衝突が起こったときに、いつでも時刻印の小さい処理単位を後退復帰するならば、従来の時刻印方式となる。本論文の方式は従来の時刻印方式の利点を保持し、二相施錠方式の多くの利点も持たせることができた。

操作履歴の管理にオーバヘッドが少し増えるが。効率向上の効果に比較して問題とならないと考えられる。操作衝突のほとんどない場合を除いてオーバヘッドによる損失は問題とならない。

参考文献：

[BG80] Bernstein, P. A. and Goodman, N. : Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems, Proc. 6th Int. Conf. VLDB, Oct. 1980.

[BS80] Bernstein, P. A., Shipman, D. W. and Rothnie, J. B. : Concurrency Control in a System for Distributed Database(SDD-1), ACM Trans. Database Systems Vol. 5, No. 1, pp.18-51, Mar. 1980.

[EG76] Eswaran, K. P., Gray, J., Lorie, N. R. A. and Traiger, I. L. : The Notions of Consistency and Predicate Lock in a Database System, Comm. ACM Vol. 10, No. 19,pp. 624-633,Nov. 1976.

[FK81] Fussell, D., Kedem Z. M. and Silberschatz, A. : Deadlock Removal Using Partial Rollback in Database Systems, ACM Proc. SIGMOD, 1981.

[GR78] Gray, J. N. : Notes on Data Base Operating systems, IBM Report RJ2188, 1978.

[GS80] Gligo, V. D. and Shattuck, S. H. : On Deadlock Detection in Distributed Systems, IEEE Trans. Softw. Eng. SE-6, No. 5, pp. 435-440, Sep. 1980.

[HA82] Hadzilacos, V. : An Algorithm for Minimizing Roll Back Cost, ACM Proc. Symposium on PODS, Mar. 1982.

[KO86] 小林哲二：データベースのロック方式における並行性向上の一方式、情報処理学会論文誌 Vol. 27, No. 2, pp. 236-242、1986.

[MM79] Menasce, D. and Muntz, R. : Locking and Deadlock Detection in Distributed Databases, IEEE Trans. Softw. Eng., SE-5, No. 3, pp. 195-202, May 1979.

[OB82] Obermarck, R. : Distributed Deadlock Detection Algorithm, ACM Trans. Database Systems, Vol. 7, No. 2, pp. 187-208, June 1982.

[RE78] Reed, D. : Naming and Synchronization in a Decentralized Computer System, Tech. Rep. MIT/LCS/TR-205, Dept. Electrical Eng. and Computer Science,Massachusetts Institute of Technology, Sept. 1978.