データベース・システム　72−17
(1989. 7. 21)

実時間データベースに適した2段階2相施錠方式

Two Step Two Phase Locking Mechanism Adapted for Real-Time Databases

上林彌彦　最所圭三　仲興国
Yahiko KAMBAYASHI Keizo SAISHO and Xingguo ZHONG
(九州大学)　(中立ソフト)
Kyushu University

あらまし　実時間データベースにおいては、トランザクションは終了期限や優先度によりトランザクションの処理順が決まる。従来の2相施錠方式では、優先度の低いトランザクションが施錠してもすくみを生じない限り解錠されないため、後からきた優先度の高いトランザクションが待たされることがあり、実時間システムには向かない。本稿では、並列トランザクション，施錠の予約，予約順の動的な変更を導入することにより2相施錠方式を実時間システムに適合させる。並列トランザクションを用いることにより、同時に処理するトランザクション数を減らすことができ競合の可能性を減らすことができる。施錠の予約を用いるため、本稿での2相施錠方式は予約と実際の施錠の2段階となる。予約の段階で、トランザクションの実行順を決めるが、その条件に優先度を導入することにより、優先度の高いトランザクションを先に実行でき、実時間性を持たせることができる。

Abstract　In real-time database systems, transactions have deadline and priority and we must execute higher priority transactions earlier than lower priority transactions. In this paper, parallel transaction model, reservation of lock and dynamic exchange of reservation order are utilized in order to adapt two phase locking mechanism to real-time databases. Parallel transaction can reduce the number of transaction executing concurrently and reduce the possibility of deadlocks. The mechanism consists of two levels, reservation level and real locking level. At the reservation level priority can be reflected.

1. まえがき

データベースの利用範囲が広まるにつれ、応答時間に制限がある環境でも使用されれうようになってきている。このような制限を持つデータベースを実時間データベースと呼ぶ。

これまでのデータベースにおける並行処理制御では、データベースに矛盾を生じないようにしてトランザクション処理の効率(単位時間あたりの処理量)を上げることが目的であり、応答時間に関してはあまり問題にされていなかった。これに対して、実時間データベースでは、それぞれのトランザクションは終了しなければならない期限を持っており、並行処理制御の目的も、終了期限を守るトランザクション数を最大にすることである。

これまでの実時間データベースにおける並行処理制御に関する研究では、トランザクション処理のモデル化[7]，トランザクションの優先度に関する議論[7,8]，それらの優先度を用いたスケジューリングの方法[8,9]等が中心になっている。スケジューラによる並行処理制御を行なうためには、各トランザクションが使用するデータ集合が全てわかっていなければならないなどの制限がある。

データベースに対する質問処理では、最初にメタデータと呼ばれるデータベースを管理するデータにアクセスして、それによって実際に操作するデータ集合を決定するという2段階で処理するため、トランザクションが扱うデータ集合が動的に決まり、スケジューラが使用できない場合がある。これらのトランザクションを扱うために一般の2相施錠方式をそのまま実時間データベースに用いた場合は、各トランザクションの終了期限を制御できないため実時間性を満足できない。

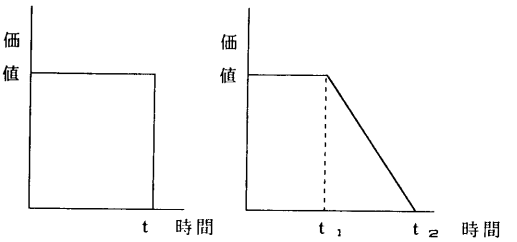
複数のトランザクションを同時に並行して処理する場合は、同時実行するトランザクションの数が少ないとCPUやディスクアクセスの空き時間が多くなり処理効率が低下する。逆に多すぎると、処理の切り替えのオーバヘッドの増加，並行処理制御に要する処理量の増加，トランザクション間の並行度を上げすぎる競合の増加により処理効率が低下する。このため、最大の効率を得ることができる処理の多重度が存在すること

になる。並列トランザクションを用いることにより、少ないトランザクション数で最大の処理効率を得ることができるようになり、これによってトランザクションの競合を減らすことが可能である[6]。しかしながら、通常の2相施錠方式で並列トランザクションを制御すると、1つのトランザクションが複数のデータに対して同時に待ち状態にはいることがあり、これによりトランザクション間の競合が増加することで一般の直列トランザクションを用いた場合と全体の効率は変わらなくなる。このような問題に対して、著者らは施錠の予約を取入れ、各データにおける施錠待ち順が揃うように動的に施錠の予約順を変更することによる並列トランザクション特有のすくみを減少させることができる、多重待ち2相施錠方式を開発している[1,4]。また、2相施錠方式においては障害に対する対策およびすくみが生じた場合の後退復帰の効率化のために、書き込み操作によるデータベースの変更を作業領域に蓄えておき、トランザクション処理の最後にまとめてデータベースに反映させる狭義2相施錠方式を用いることが多い。このようなシステムでは、書き込み施錠を行っている場合でもデータベース上には元のデータが残っている。このデータを用いることにより読み出し－書き込み競合を解消することができる[2,3,5]。また書き込み－書き込み競合の場合でも書き込もうとしているデータを読み込まない場合は書き込み操作の順序を変更してもかまわない。また、従来すくみ検出グラフはすくみ(deadlock)の生じたときにコストが最小になるように後退復帰するトランザクションを選んでいたが、すくみが生じていなくても優先順を満足させるために後退復帰を必要とすることがある。

本稿では、以上述べてきた方法を実時間データベースに適用する方法について述べる。次節においては、実時間トランザクションのモデル化および優先度の決定法を説明し、3節では本稿で用いる並行処理制御方式を，4節では3節で述べた方法と優先順管理との結合について議論する。

2. 実時間データベース

実時間データベースにおいては、トランザクションの期限の持たせかたで2つの分けることができる。1つは、期限(t)が過ぎたトランザクションは価値がなくなるもので、もう1つは2つの期限($t_1 < t_2$)を持ちt_1を過ぎてからt_2まで、だんだんとその価値が下がりt_2を過ぎたものは価値がなくなるもので、データベースからみた期限はt_1になる(図1)。前者をハードな期限を持つトランザクション、後者をソフトな期限を持つトランザクションと呼ぶ。



(a)ハードな期限　　(b)ソフトな期限

図1　トランザクションの価値の変化

ハードな期限を持つトランザクションは制限が厳しすぎ、実際のデータベースに当てはめた場合でもある次期を境にしてデータが全く価値を失うようなことはあまりない。例えば、年齢と給料の相関関係を取る場合は、その時点の給料が分かっていなくても古い情報があれば、ある程度の精度で相関関係を取ることができる。このようなことから、実時間データベースにおける並行処理制御ではソフトな期限を持つトランザクションを対象にしているものが多く、本稿でもこの型のトランザクションに対する並行処理制御について扱う。

トランザクションの応答時間を決める要因として、計算時間，OSに関する応答時間，ディスクアクセスなどのI/O時間，並行処理制御の方法等の不確定要素が多いため、各トランザクションの終了時間を正確に予測することができない。このため、全てのトランザクションを期限内に処理することを保証するスケジュールを生成することも不可能である。このようなことから、各トランザクションに対してその重要性(価値)，期限，計算時間や使用するデータ量等により優先度をつけ、それに従ってスケジュールが組むことが一般的である。優先度の付け方としては基本的には以下に上げるような方法があり、これらの方法の組合せることで対象とするシステムに適した方法を決める。

a. 早く始まったトランザクションを優先

一般のデータベースでよく使用される方法で、各トランザクションの期限の情報を用いないため期限に余裕がない新しいトランザクションよりも、期限に余裕があるが古いトランザクションを先に実行してしまう欠点がある。この方法の基本概念は長く実行したトランザクションを後退復帰することによる無駄な処理の

量を最小にすることである。この方法はトランザクションが同じ評価関数を持つときに有効である。

b. 期限の早い持つトランザクションを優先

期限が近いトランザクションを先に実行させようとするもので、できる限り期限を守ることができるトランザクション数を多くしようとするものである。この方法の欠点は期限が過ぎたトランザクションを優先させることである。処理のほとんどが終了しているようなトランザクションが後退復帰の対象となることもあり、その結果後退復帰したトランザクションが期限が間に合わないことがある。

この方法を改良した方法として、期限内に終了することができるトランザクションに限ってのみ上記の条件を適用する方法がある。この場合、期限が過ぎたトランザクションのためにために競合しないデータをアクセスできるようにする。

c. 価値密度が大きいトランザクションを優先

価値密度として終了時の価値を計算時間で割ったものを用いる。これは同じCPU時間で最も大きな価値を生み出すことができるトランザクションを優先させるものである。

3. 並行処理制御

本節においては、並行処理制御においてその正しさの基準になる直列可能性，並列トランザクション，並列トランザクションを効率よく処理するための多重待ち2相施錠方式，さらに施錠の入れ替えによる読み出し操作の効率向上について述べる。

3.1 直列可能性

データベースに対して、複数のトランザクションを逐次的に実行した結果は矛盾がない正しい状態であると考えることができる。もし、任意のトランザクション集合を並行して実行した結果が、集合内のトランザクションを、ある順序で逐次的に実行した結果と等価になればその並行処理が正しく行われたとみることができる。このような実行系列を直列可能であるといい、直列可能な実行系列を生じる並行処理制御は直列可能性を満足するという。直列可能性を満足する並行処理制御は常に正しい結果を生じる。直列可能性を満足する並行処理制御方式としては2相施錠方式と時刻印方式の2つが代表的である。時刻印方式では、トランザクションの実行順が処理を始めた時点で決められるため、優先度の与え方のa)の方法以外は実時間データベースには向かないと考えられる。これに対して、2相施錠方式ではデータの競合が起きたときにそれらのトランザクションの優先度にしたがって先に実行させることができるため実時間データベースに向いてると考えられる。

3.2 並列トランザクション

同一トランザクション内の処理でも独立な処理は並行して実行できる。このようなトランザクションを並列トランザクションと呼ぶ。これに対して同一トランザクション内の処理を逐次的に実行するトランザクションを直列トランザクションと呼ぶ。トランザクション中の操作の処理の依存関係は、処理を節点，自分の処理結果を使用する処理に向かって有向枝が出るような半順序有向グラフで表すことができる(図2)。各節点にあたる処理を実行する条件は自分を指す枝を出している処理が全て終了していることである。例えば、startが終了すると、L(A), L(B), L(C) を並行して実行できる。

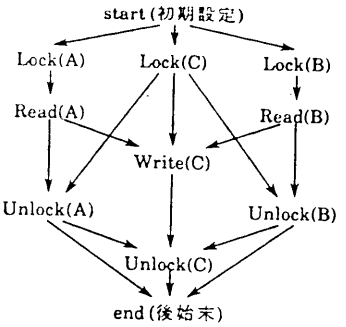


図2　並列トランザクションの例

並行処理制御の目的は、複数のトランザクションを同時に実行することによりあるトランザクションの計算が行われている間に別のトランザクションのデータアクセスを行うことにより、CPUとディスクを同時に使用でき、その結果処理効率が向上することと、応答時間の平均化である。処理の並行度(同時に実行するトランザクション数)が高くなるとCPUやディスクの遊び時間が小さくなるため処理効率が上がるが、高くなりすぎると1つのデータに対する競合が増加することによるすくみの増加や、処理の切り替えに要するオーバヘッドが大きくなるため逆に処理効率が落ちる。このため、処理効率を最大にする並行度が存在することに

なる。並列トランザクションを用いることの利点は、最大効率を得るためのトランザクション数を減らすことができ、その結果トランザクション間の競合が減少しすくみも減少する事が期待できる。しかしながら、従来の並行処理制御方式で並列トランザクションを制御した場合は、並列トランザクションが同時に複数のデータを待つことがあり、これによるすくみの増加を引き起こすため、最終的には直列トランザクションを用いた場合と処理効率が変わらないこともある。

これに対し我々は多重待ち2相施錠方式と呼ぶ、並列トランザクション特有のすくみを回避することのできる並行処理制御方式を開発した。

3.3 多重待ち2相施錠方式

2相施錠方式では、施錠しているデータに対して複数のトランザクションから施錠要求が来た場合に次に施錠できるトランザクションの選択を簡単にするために、図3.bに示すように現在施錠しているトランザクションが先頭にくるようなリストを用いて施錠管理する。この例ではT_1がデータAを解錠するとT_2の施錠が許可されることを表わす。このリスト内の順序がトランザクションの実行順になる。各データで生じるトランザクション順が異なる場合にすくみを生じる。すくみが生じた場合はすくみに入っているトランザクションを1つ以上を後退復帰させることで解消する。すくみの検出は、互いに相手を待っているトランザクション集合の検出になるため、各データで生じる施錠待ちリストを合成した待ちグラフに閉路が生じていないかどうかの検出に置き換えることができる。図3の例に$L(T_1,B)$($L(T_1,D)$はT_1がDを施錠することを表わす)を加えると、待ちグラフに点線で示した枝が加わりすくみを形成する。

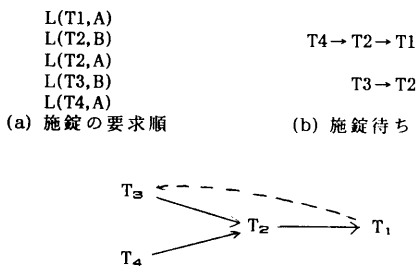


図3　すくみの検出

次に、並列トランザクション特有のすくみについて示しさらにその種のすくみを減少させる方法について示す。

直列トランザクションでは、施錠待ちの対象となるデータ数は1つのトランザクションにつき最大1個であった。ところが並列トランザクションでは、1つのデータで施錠待ち状態になっても独立した施錠操作を行うことができるため、別のデータででも施錠待ち状態にはいることがある。例えば、図3の例ではT_3とT_4は施錠待ちになっているため直列トランザクションではそれ以上処理を進めることができずすくみを生じない。これに対して、並列トランザクションでは、$L(T_3,A)$と$L(T_4,B)$が実行できその結果データA,B上の施錠待ちリストおよび待ちグラフは図4.aに示すようになりすくみを形成する。この状態ではまだT_3はまだデータAを実際には施錠していないので、データAの施錠待ちリストのT_3とT_4を入れ換えることが可能で、これによりすくみを解消できる(図4.b)。

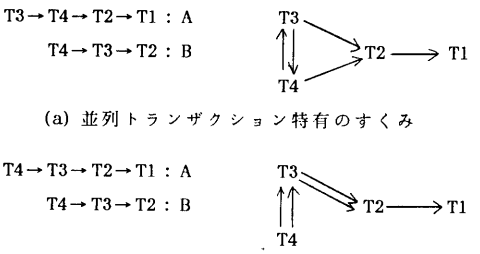


図4　施錠待ち順の変更によるすくみの回避

このように、各データ上の施錠待ち順を動的に変更し互いに揃うようにすることですくみを回避できる。但し、現在施錠しているトランザクションの順序の変更は、2相施錠規約に違反するためできない。

この施錠待ちリストの変更は、どのトランザクションがどのトランザクションを施錠するかの情報が多いほど、また実際に施錠しているトランザクション数が少ないほど柔軟に行うことができる。前者は施錠時期が早いほど有利である示し、後者は逆に遅いほど有利であることを示す。この2つの相反する要求を満足させるために、施錠命令を予約と実際の施錠の2つの命令に分離する。予約をできる限り早い時期に出し、実際の施錠をできる限り遅い時期に出すことで両方の要求に答えることができる。

各データに施錠待ちリストを持たせてトランザクシ

ョンの順序を管理すると、1回の施錠要求で複数の施錠待ちキューの変更を行わなければならないことがあり、冗長である。例えば、T_1がAを施錠し、B,CでT_2の次に登録されているときに、T_2がAに対して施錠要求すると、B,Cの施錠待ちリストのT_1とT_2の入れ替えを行わなければならないため冗長である。そこで、施錠待ちリストをトランザクションの順序を管理する部分と施錠を管理する部分に分けて管理することにより冗長性を少なくする。

施錠管理の部分では各データに、施錠しているトランザクション集合(Lock(A))およびそのモード(Lmode(A), 共有(S), 占有(X)), 共有施錠予約集合(Res_s(A)), 占有施錠予約集合(Res_x(A))を持たせることで管理する。

トランザクションの順序の管理はTWO(Transaction Waiting Order)と呼ぶ表で一括して管理する。TWO内の順序は、以下の条件で決定される。

① T_iが待たなければならないトランザクション集合($Wait(T_i)$)を以下のように定義すると、$Wait(T_i)$に含まれるトランザクションはT_iの前に位置する。

$Wait(T_i) \equiv \cup Wait(T_j) \cup Wait_{dir}(T_i)$

$(T_j \in Wait_{dir}(T_i))$

$Wait_{dir}(T_i)$はT_iが直接待つトランザクション集合で以下のように定義される

$Wait_{dir}(T_i) \equiv \cup Lock(A)$

$(T_i \in Res_x(A)) \vee ((T_i \in Res_s(A)) \wedge (Lmode(A)=X))$

② ①以外はそのシステムで使用されている優先度に従って並べるが、特別な順位が無ければ入ってきた順になる。

以下に、施錠の予約, 実際の施錠, 解錠操作における処理をTiがAに対してモードMで要求している場合について示す。

[施錠の予約]

この予約の結果 $T_i \in Wait(T_i)$になると、すくみを生じることになるためT_iを後退復帰する。後退復帰ではT_iが施錠しているデータを解錠する。

予約が許されると、Tiは予約のモードにしたがってRes_s(A)またはRes_x(A)に加えられる。さらに$Wait(T_i)$および$Wait(T_j)$にT_iを含むものを更新する。さらに、新たに$Wait(T_i)$に加わったトランザクションの中で、それまでTWO内でT_iの後ろにあったものを前に出す。

[実際の施錠]

施錠の条件としては、2相施錠規約を守ることと、この施錠の結果すくみを生じないことである。これは、以下の2つの条件を満たせばよい。

I ((Lmode(A)=S)∧(M=S))∨(施錠されていない)

II $T_j \notin Wait(T_i)$ M=Xのとき$T_j \in Res_x(A) \cup Res_s(A)$

M=Sのとき$T_j \in Res_x(A)$

さらに、トランザクションの実行順の制御のために以下の条件を加える。

III Res_s(A)とRes_x(A)に含まれる全てのトランザクションがTWO内でT_iの後ろにある。

これらの条件が成り立つまで施錠が待たされる。許された場合は、Lock(A)にT_iを加えRes_x(A)およびRes_s(A)に含まれるトランザクションのWaitを更新する。

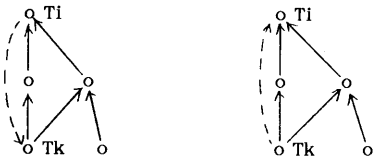
[解錠]

この操作は常に許可され、Lock(A)からT_iを抜き、さらにWaitにT_iを含むものを更新する。

3.4 施錠順の変更によるすくみの解消

2相施錠方式においては、後退復帰のコストの軽減および障害対策のために、各トランザクションの書き込みデータは作業領域に蓄えられていて最後にまとめてデータベースに反映させ、正常に書き込めたときにトランザクションを終了する。このような方式を狭義2相施錠方式と呼ぶ。このため、書き込みのための施錠が行われていてもデータベースに書き込みが反映されていない場合はそのトランザクションの実行前のデータを取り出すことができる。例えば、T_kがT_iを間接的に待つ($T_k \rightarrow T_i$)ときに、Tkが占有施錠しているデータにT_iが読み出しのために共有施錠を行おうとするとすくみを生じてしまい、T_kまたはT_iを後退復帰させなければならない(図5.a)。ところが、実際にはT_kがT_iを待っているためT_kの書き込みデータがデータベースに反映されていないため、データベース内のデータはT_kが実行される前のデータを保持している。そこで、T_kの施錠を中断してT_iに先に施錠させ読み出し操作を先に行わせることにより、このすくみを回避できる(図5.b)。

すくみを生じているトランザクションの書き込みは施錠がかかっていても実際の書き込みが行われていないため競合を回避するために後からきた共有施錠を許容することができる。そのような施錠を許容共有施錠という。よって、共有施錠をかけるときはすくみを生じないことになり処理効率を向上できる。



(a)R-W競合によるすくみ　(b)W-Rによるすくみの解消

図5　読み出し施錠の追越しによるすくみの回避

占有施錠がぶつかった場合でも特別な条件下では、共有施錠と同様に現在施錠中の施錠を中止して先に施錠させることができる。その条件は、占有施錠をかけているトランザクションがそのデータに対して読み出し操作を行っていないという条件である。上記の例を引用するなら、読み出し操作が行われていないときに後からきた共有施錠を先に行ってもTK→TIになるだけでそのデータに関して矛盾を生じることはない。読み出し操作が行われていたなら、TK→TI→TKになり、矛盾を生じる。占有施錠のうち読み出しと書き込みを行う施錠操作を明示的に更新施錠として分離することで、占有－占有または更新－占有の場合のすくみを回避する事ができ、さらにすくみの可能性を小さくできる。

4. 優先順管理と前節の方式の結合

優先順位を反映するためには、すくみが生じなくても優先順位の高いトランザクションが入ってきた場合はそのトランザクションと競合する優先順位の低いトランザクションを後退復帰させなければならない。このため、後退復帰の必要があるか，必要があればどのトランザクションを後退復帰させるかの条件を決めなければならない。

本稿で提案する並行処理制御方式では、施錠を予約と実際の施錠の2段階で行う。施錠の予約の段階で予約が実際の施錠になったときにすくみを生じないかを検出する。すくみを生じるときは、現在施錠中のトランザクションと予約を出しているトランザクションの間の優先順位でどのトランザクションを後退復帰させるか、また3.4節で述べた方法を適用できるか等を検査して最良の処理を選択する。また、実際の施錠の段階では、現在施錠中のトランザクションおよび同じデータを予約中(施錠待ちのトランザクションを含む)トランザクションの優先順位と施錠を要求しているトランザクションの優先順位との関係ですくみが生じなくても後退復帰させる場合が生じる。以上述べたような操作を実現するために、待ちグラフを用いて上記の検査を行う。

並列トランザクションを用いた場合の待ちグラフでは、1つのトランザクションが複数のトランザクションを待つことがあるため、1つのノードから複数の枝が出る。また、共有施錠を用いるため、同じデータで複数のトランザクションを待つことになる。このため、待ちグラフも木構造でなく網構造になる(図6)。

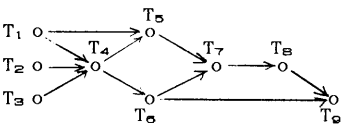


図6　並列トランザクションによる待ちグラフ

施錠の予約および実際の施錠のときの操作を説明するために、ここで用いる記号の説明をする。

- T_I：施錠の予約または実際の施錠を要求するトランザクション
- T_L：T_Iが要求しているデータを施錠しているトランザクション集合
- T_R：T_Iが施錠の予約または実際の施錠を要求するデータを予約または施錠待ちしているトランザクション集合
- P_I：T_Iの優先順位
- P_L：T_Lに含まれる最も大きい優先順位を持つトランザクションの優先順位
- P_R：T_Rに含まれる最も大きい優先順位を持つトランザクションの優先順位

まず始めに施錠の予約について説明し次に実際の施錠での操作を説明する。

施錠の予約の時は最初にすくみの検査を行う。各予約や実際の施錠のときにすくみの検査を行いすくみを生じていればそれを解消しているので、予約の前はすくみを生じていない。よって、すくみの検査は、T_Lから始めて枝に沿ってノードを巡回しT_Iのノードに達するかどうかで判断できる。Tiのノードに達する場合はこの予約の結果待ちグラフに閉路を生じることになりすくみを生じる。

すくみが生じない場合は予約を許可する。ここで、T_LとT_Iとの間で優先順位の比較を行わないのは、例えP_LがP_Iより優先順位が低くてもT_Iが実際に施錠する前にT_Lの処理が終了していることがあるからである。

すくみを生じる場合は、P_LとPiの優先順位および3.4節で述べた方法を適用してすくみを解消できるかで処理が異なる。P_IがP_Lより高い場合は、T_Iを先に実行しなければならないため、T_Lの施錠を中断しなければならない。もし3.4節で述べた方法が適用できた場合はT_Lの施錠を中断(取り消して予約の状態に戻す)する。それ以外は後退復帰させる。逆に、P_LがP_Iより高い場合は、P_Iを後退復帰させる。

次に、実際の施錠について説明する。施錠の予約では3.3節で述べた方法を適用しているため、追越しが無ければすくみを生じないが、優先順位によっては追越しが生じる。従って、実際の施錠の時は、現在施錠中のトランザクションおよび予約中のトランザクションの優先順位によって以下のように場合分けできる。

① $P_L \geq P_I$のとき：T_IはT_Lの後に実行することになり、T_Iを施錠待ち状態にする。

② $P_I > P_L$のとき：基本的にはT_Lを中断または後退復帰させてT_Iに施錠させるが、その施錠の結果T_Rとの間ですくみを生じることがあるため、T_IとT_Rの待ち関係および優先順位によって処理が異なる。

(a) $Wait(T_I) \cup T_R = \phi$のとき：$T_I$が先に施錠しても$T_R$に含まれるトランザクションとの間ですくみを生じないためT_Iの施錠を許可する。T_Iに施錠させるために、T_Lに含まれるトランザクションとT_Iに対して3.4節での方法を適用してすくみを解消できるかで施錠の中断または後退復帰を行う。

(b) $Wait(T_I) \cup T_R \neq \phi$のとき：$T_I$に施錠させることにより$T_R$に含まれるトランザクションとの間ですくみを生じるため、T_IをT_Rに含まれるトランザクションの優先順位に従って処理する。

(b.1) T_Rに含まれるトランザクションの優先順位がT_Iより高いとき：T_IはT_Rより後に処理するため、T_Iを施錠待ち状態にする。

(b.2) T_Rに含まれるトランザクションの優先順位がT_Iより低いとき：T_Iに施錠させるために$Wait(T_I) \cup T_R$に含まれるトランザクションを後退復帰し、さらに(a)のときと同様の処理を行う。この場合、T_Iと$Wait(T_I) \cup T_R$の間で3.4節で述べた方法で待ち関係を逆転できれば$Wait(T_I) \cup T_R$に含まれるトランザクションを後退復帰しなくて済む。ところが、この検査のためには待ちグラフ内のTiから$Wait(T_I) \cup T_R$に含まれるトランザクションまでの全ての経路を調べ、さらに各ノード間の枝の付け換えを要するためそのオーバヘッドが大きくなる。オーバヘッドが後退復帰のコストより小さい場合は有効である。

以上述べた方法は、優先順位で処理順を決める場合の基本的な概念である。そこでは、これまで消費されたI/Oアクセスや CPU資源の消費量，後退復帰に要するコストおよび終了時間等を考慮に入れていなかった。そこで以下ではこれらの要因を考慮にいれた場合の処理について議論を行う。

これまで消費されたI/Oアクセスや CPU資源の消費量，後退復帰に要するコストは無駄になる処理量を表わすものと考えられる。従って、この2つコストの和を後退復帰コストとして考え、それに適当な係数を掛けたものを後退復帰させないための優先順位として考えることができる。後退復帰しなければならないかどうかを検査するときの優先順位として元々の優先順位にこの優先順位を加えたものを用いることにより、後退復帰コストも組み込むことができる(図7)。例えば、施錠の段階で後退復帰しなければならないトランザクションを選択するときに以下の条件を使用する。

$P_L + \alpha * C_L > P_I + \alpha * C_I$

α ：後退復帰コストの重み

Ci：T_Iの後退復帰コスト

C_L：T_Lに含まれる全てのトランザクションを後退復帰する総コスト

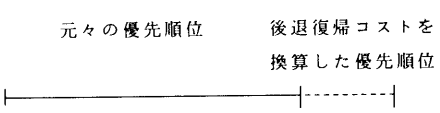
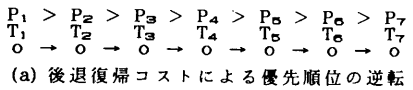


図7　後退復帰時の優先順位

また、この条件を加えた場合の問題点として、待ちグラフの先頭に優先順位の低いトランザクションが位置し、順に少しずつ優先順位の高いトランザクションが並びその結果、最後に最も高い優先順位を持つトランザクションが位置することがある(図8.a)。その結果優先順位の高い後ろのほうのトランザクションほどまたされ期限に間に合わないことがある。このような状況は実時間データベースに取って好ましくない。途中のトランザクションの1つを後退復帰させることにより待ちグラフが2つの小さなグラフに分割できその結果同時に2つの処理を実行できるようになり、後ろの優先順位の高いトランザクションの期限が守られる可能性が高くなる(図8.b)。



(a) 後退復帰コストによる優先順位の逆転

(a) 待ちグラフの分割

図8　中間トランザクションの後退復帰による待ちグラフの分割

次に処理期間の予想がつく場合について考えてみる。この場合の特徴として優先度が低いトランザクションでも、優先度の高いトランザクションを期限以内に終了できる時間までに終了するならその処理を続けたほうが有利なことである(図9)。また、予約の段階でも、優先度の低いトランザクション(T_1)が施錠しているデータに対して優先度の高いトランザクション(T_2)が施錠の予約し、T_2を期限以内に終了できる時間までにT_1が終了しなければ、最終的にはT_1を後退復帰しなければならない。従って、T_1をT_2の予約の段階で後退復帰することでT_2が実際に施錠するまでのT_1を処理しなくてすみ、その分処理効率および応答時間が早くなる。また、実際に施錠するときでも優先度が低いために後退復帰させられていたものでも、すぐに終了でき優先度の高いトランザクションの終了期限を守らせることができれば後退復帰の必要がなくなる。

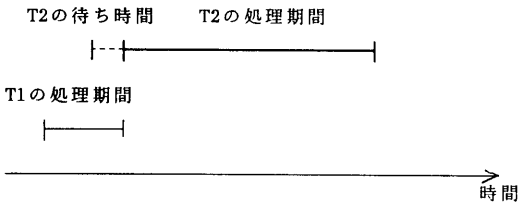


図9　終了時間が判る場合のスケジュール

5. まとめ

以上、2相施錠方式を実時間データベースで用いる方法を示した。ここで示した方法は、我々が既に開発している多重待ち2相施錠方式の機能を拡張するだけで比較的簡単に実現できる。

ここでは、終了期限が来たトランザクションの処理については述べていないが、図1.bで使用した価値関数を用いて優先順位を漸次下げていく。

ここで示した施錠トランザクションの選択法は、2相施錠方式を実時間データベース向きにするための基本的な考え方を示したもので、応用に従って優先度の決め方、特に後退復帰のコストに対する優先度の与え方は検討しなければならない。

今後は、本稿で示した方法がどの程度有効であるかをシミュレーション等によって検証する予定である。

参考文献

[1] K.Saisho and Y.Kambayashi, "Multi-Wait Two-Phase Locking Mechanism and its Hardware Implementation," Database Machines and Knowledge Base Machines, Kluwer Academic Publishers, pp.143-156, 1988.

[2] X.Zhong and Y.Kambayashi, "A Two-Phase Locking Mechanisms Avoiding Deadlock Caused by Read-Write Conflicts," RIMS of Kyoto Univ., Vol. 625, pp.186-195, 1987.

[3] 仲, 上林, "二相施錠方式における読み出し操作," 情報処理学会第34回全国大会, 7C-7, 1987.

[4] 最所, 上林, "並列トランザクションのための二相施錠方式," 電子情報通信学会・システム部門別全国大会, 591, 1987.

[5] 仲, 上林, "並行処理制御を考慮した選択実行対策," 情報処理学会第36回全国大会, 7E-6, 1988.

[6] J.A.Brozozowski and S.Muro, "On Serializability," Int. Journal of Computer and Information Science, Vol.14, No.6, 1985.

[7] J.A.Stankvic and W.Zhao, "On Real-Time Transactions," SIGMOD Rec., Vol.17, No.1, pp.4-18, 1988.

[8] R.Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions," SIGMOD Rec., Vol.17, No.1, pp.71-81, 1988.

[9] L.Sha, "Concurrency Control for Distributed Real-Time Databases," SIGMOD Rec., Vol.17, No.1, pp.82-98, 1988.

[10] S.H.Son and C.-H. Chang, "Distributed Real-Time Database Systems: Prototyping and Performance Evaluation," Int. Symp. on Database Systems for Advanced Applications, pp.251-258, 1989.

[11] M.Singhal, "Issues and Approaches to Design of Real-Time Database Systems," SIGMOD Rec., Vol.17, No.1, pp.19-33, 1988.