システムソフトウェアと
オペレーティング・システム 73－15
（1996．8．27）

時刻印方式に基づく並行処理制御と優先度スケジューリング

足高 正訓† 大久保 英嗣††

†立命館大学大学院理工学研究科
††立命館大学理工学部情報学科

本論文では時刻印方式を基本としたトランザクションの並行処理制御と優先度スケジューリングの方式について述べる．まず時刻印方式を基本とした並行処理制御方式である優先度付き時刻印方式 (PTO: Priority based Timestamp Ordering) について述べる．PTO は従来の直列可能性理論を見直すことによって，時刻印方式で問題となるアボート率を低減すると同時に，トランザクションの優先度を反映した制御を可能としている．次に並行処理制御に時刻印方式を用いた場合のリアルタイム同期問題について述べる．特に，トランザクションの各々の特性に応じて静的および動的なスケジューリングを行なうための方式について述べ，ロックを使用したスケジューリング方式との比較を行なう．

A Concurrency Control and Priority Scheduling based on the Timestamp Ordering

Masanori Adaka† Eiji Okubo††

†Graduate School of Science and Engineering, Ritsumeikan University
1916 Noji, Kusatsu, Shiga 525, Japan

††Department of Computer Science,
Faculty of Science and Engineering, Ritsumeikan University
1916 Noji, Kusatsu, Shiga 525, Japan

In this paper, a concurrency control method and a priority scheduling method based on timestamp ordering for the transaction processing are described. First, a new concurrency control method called PTO(Priority based Timestamp Ordering) is proposed. PTO can decrease the rate of transaction abortions by reconsidering the serializability rule for transactions. Furthermore PTO makes it possible to schedule transactions based on their priorities. Next, in this paper, the real-time synchronization problem with timestamp ordering is described. Especially, static and dynamic scheduling methods of transactions with their characteristics are proposed, and its comparison with locking method is presented.

1 はじめに

近年，トランザクションのリアルタイム処理が注目を浴びている．リアルタイム処理ではトランザクション間の資源排他制御に加え，優先度やデッドラインを考慮した制御が要求される．トランザクション処理の正当性を保証する並行処理制御方式として，2相ロック方式と時刻印方式の2つが挙げられる．現状ではどちらの方式も一長一短があるが，リアルタイムトランザクションの優先度処理やスケジューリングに対しては，これまでロックを基本とした方式が主に考えられてきた．しかしロックを使用する以上デッドロックが避けて通れない問題となっている．そこで本論文では，時刻印方式を基本とした新しい並行処理制御方式とスケジューリング方式を提案する．時刻印を基に制御を行なうことにより，デッドロックフリーで並行性の高い処理が可能となる．

以下，本文では，**2.** で時刻印方式を基に改良を加えた並行処理制御方式である優先度付き時刻印方式 (PTO: Priority based Timestamp Ordering) について，処理方式および実験結果を述べる．PTO は時刻印方式で問題となるアボート率を低減すると同時に，優先度を考慮したアクセス制御を行なうものである．**3.** では時刻印方式を用いた場合のスケジューリング手法を示す．本手法は，ロックを用いた場合と比較するとブロック時間が短いことが特徴である．

2 並行処理制御方式 PTO

時刻印方式では，2相ロック方式のようにデッドロックが発生することはないが，トランザクションのアボート率が高いことが問題となる．本論文で提案する優先度付き時刻印方式 PTO は，その問題点を解消するものである．PTO では，並行実行されているトランザクションが直列可能であるための条件を見直すことによって，トランザクションのコミット率を向上させることを可能にしている．また同時に PTO は，トランザクションの優先度を考慮した並行処理制御も行なう．これによりトランザクションのコミットメント制御において，よりユーザの意図した優先度で処理を終了させることが可能となり，さらにシステムの性能向上も期待できる．

2.1 処理方式

時刻印方式では，主に遅い参照 (delayed read) および遅い更新 (delayed write) と呼ばれる操作要求を行なった時にアボートが発生する (図1参照)．これは，時刻印の順で直列化が可能でなくてはならないという時刻印方式の規則に基づくものである．しかし実際には，このような遅いアクセス (delayed access) が起きても直列化が可能となる場合は存在する．そこで PTO はこのような競合発生時に，トランザクションの優先度を基にアボート対象を決定する．さらに時刻印の順以外での直列化の可能性を解析し，不必要なアボートを回避する．

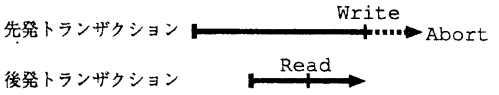

図 1 遅い更新の例

以降，PTO の処理方式を説明する際に用いる記号を次のように定義する．

$TS(TR_n)$	: トランザクション n の開始時刻印
$P(TR_n)$	: トランザクション n の優先度 (値が大きいほど優先度が高い)
DT	: 遅いアクセス要求をしたトランザクション
$\mathcal{ET}$	: 競合したオブジェクトを先にアクセスした後発トランザクションの集合

(このとき $TS(DT) < min\{TS(\mathcal{ET})\}$)

PTO はトランザクション間に競合が発生していない間は，従来の時刻印方式と同様の処理を行なう．オブジェクトの競合，いわゆる遅いアクセスが発生した場合に異なる処理が行なわれる．また PTO は多重版型の時刻印方式にも対応している．多重版時刻印方式では参照時の競合発生がなくなるため，遅い更新のみに注目すればよい．

具体的な処理方法を以降に示す．遅いアクセスを検知したときには次の条件を満たすかどうかで処理が分かれる．

条件: DT が今までにアクセスしたオブジェクトに関して，他のトランザクションによる $TS(DT)$ 以降のアクセス (ただし Read-Read の関係を除く) がない．

(1) 条件を満たしている場合

次の処理を行なうことにより，DTの遅いアクセスを許可する．DT は，以後通常のトランザクションと同様の扱いで処理され，いかなる特別な制約も受けない．

1. $TS(DT)$ を現時刻に書き換える．
2. DT が今までに行なった操作のログ (オブジェクトのアクセス時刻) を現時刻に書き換える．

(2) 条件を満たしていない場合

a). $\mathcal{ET}$の中に既にコミットしたものが含まれるとき
・DTをアボートする.

b). $\mathcal{ET}$の要素すべてが未完了のとき
・$max\{P(\mathcal{ET})\} \geq P(DT)$ なら DTをアボートする.
・$max\{P(\mathcal{ET})\} < P(DT)$ なら $\mathcal{ET}$をアボートする.

本制御方式は次のような考え方に基づいている. 上記の条件を満たすことはすなわち, 他のトランザクションとの間に共有オブジェクトのアクセスによる依存関係がないことの証明である. 従って時刻印の付け替えなどを行なうことにより, 記録上現時刻から処理を始めたように見せてアボートを回避する. 一方, 条件を満たさないときにはいずれかのトランザクションをアボートさせる必要があるので適切なものを選択する.

2.2 優先度の継承

多重版時刻印方式では, まだコミットしていないトランザクションが更新したオブジェクトに対して, より大きな時刻印を持つトランザクションによる参照要求があったときの処理に, 保守的時刻印方式と積極的時刻印方式と呼ばれる2つの方式を選択することができる. 積極的時刻印方式で処理を行なった場合, アボートしたトランザクションが作成したバージョンを先読みしていたトランザクションが連鎖的にアボートするというカスケーディングアボートが問題となる.

PTOを, 積極的時刻印方式と併用した場合には, これによる優先度逆転現象 (priority inversion) に注意しなくてはならない. 例えば $P(TR_1) < P(TR_2) < P(TR_3)$ のような3つのトランザクションがあるとする. TR_1 が更新を行なったバージョンを TR_3 は既に先読みを行なっている. 次に TR_1 より大きな時刻印を持った TR_2 が参照したオブジェクトに, その後 TR_1 が更新要求を行なうと競合アクセスとして検知され, 優先度の関係から TR_1 がアボートすることになる. するとその影響で, 先読みを行なった TR_3 もアボートすることになってしまう. つまり中間優先度の TR_2 が, 最高優先度の TR_3 のアボートを誘発したことになる.

これに対応するために, 優先度の継承を行なう. つまり先読みによって確定していない最新バージョンを参照したトランザクションは, そのバージョンを作成したトランザクションに, 自分の優先度の方が高いならばその優先度を継承させる. これにより高い優先度を持つトランザクションが, 低い優先度のトランザクションのアボートに巻き込まれて失敗することは回避できる.

2.3 実験結果

図2はPTOで処理を行なった場合の他方式との性能比較である. この図は, 各方式で1000個のトランザクションを100000単位時間の間に実行したときのコミット率を示している. 対象オブジェクト数はいずれも20であり, 単位時間を基準とした平均トランザクション長と, 各トランザクションごとのオブジェクトに対する平均アクセス数を変化させることにより, 競合の発生確率を操作している. 図2よりPTOは競合が多く起こるような場合特に有効であると考えられる. また表1は, PTOによる優先度別のコミット率の変化を表している. これにより優先度が高いトランザクションは他のものに比べ, より確実にコミットしていることがわかる.

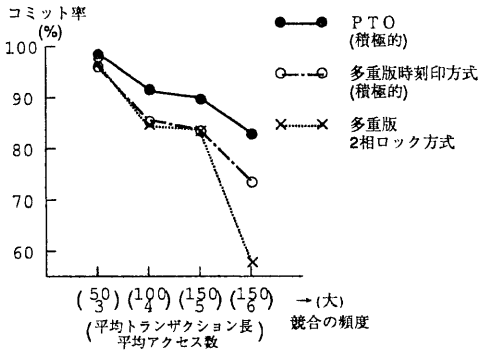

図2 各方式によるコミット率の比較

表1 優先度ごとのコミット率 (%)

	(低) ← 優先度 → (高)					
優先度	1	2	3	4	5	平均
コミット率	75.5	80.3	81.3	84.5	90.4	82.4

3 時刻印に基づくスケジューリング

本章では共有資源の排他制御を含めたリアルタイム同期プロトコルとして, 時刻印方式を基にトランザクションをスケジューリングする方法について述べる. また最終的には前述のPTOと組み合わせて用いることにより, スケジューリングを含めた一貫した並行処理制御を確立することを目的としている. ただし, 以下では外部記憶との入出力時に発生する待ち時間も考慮に入れた1プロセッサ上でのスケジューリング問題を対象とする.

3.1 従来のスケジューリング手法

タスクのリアルタイムスケジューリング手法として，レートモノトニック (RM: Rate Monotonic)，最短デッドライン順 (EDF: Earliest Deadline First)，最短スラックタイム順 (LS: Least Slack Time First) などがよく知られている [1]．また，リアルタイム同期問題では優先度継承プロトコルや優先度シーリングプロトコルなどが提案されている．従来のトランザクションのリアルタイムスケジューリングに関しても，これらの方式を基本とするロックを用いたアクセス制御が主に行なわれてきた [2]．しかし時刻印方式ではロックをかけるという概念が存在しないため，根本的に今までのスケジューリング技法を見直す必要がある．次節からはその有効性と具体的な手法について述べる．

3.2 スケジューリングにおける時刻印方式の有効性

並行処理制御は，複数のトランザクションが存在する状況で，実行中のトランザクションの I/O による待ち時間 (CPU を使用しない時間) が生じることから，この間に他の実行可能なトランザクションの処理を行なって処理効率を上げるために行なわれる [3]．しかし，このような待ち時間が無視できる場合，即ちすべてのデータが主メモリ上に存在し，割り付けられた CPU を自分から放棄することがない場合について考えてみる．このような条件のもとで，時刻印方式とロック方式による共有オブジェクトに対する制御を比較する (図 3参照)．ロック方式ではアクセスしたいオブジェクトに既にロックがかかっている時には，一度ロックを所有するトランザクションに CPU を譲って処理をさせ，ロックが解放されるまで待つ必要がある．これに対し，時刻印方式では，常に最高優先度を持つトランザクションに CPU が割り当てられ，中断することなしに処理を終了することができる．最高優先度のトランザクションのアクセス要求は基本的に何者にも妨害されない．

このことは時刻印方式の特性に起因している．優先度の高いトランザクションにとって，競合の原因となるアクセスを行なうトランザクションが，2 相ロックでは時間的に自分の前後に存在する．しかし，時刻印方式では自分の後にしか存在せず，しかも優先度が低い場合は処理を譲ってもらえないので競合アクセスを行なう機会すら与えられないためである．従って優先度の低いトランザクションのアクセスが原因でアボートするトランザクションは存在しない．

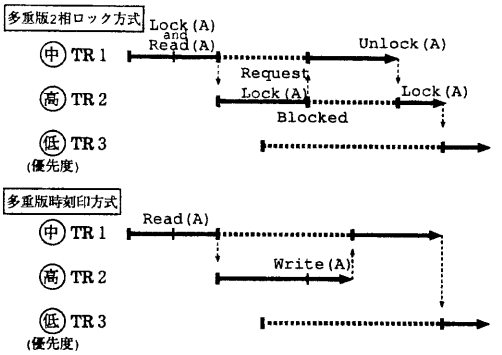

図 3　2 相ロック方式と時刻印方式の比較

EDF のような優先度割り付けを行なう場合，優先度が高いということはデッドラインが近いことを意味している．このような場合では，他のトランザクションに処理を譲る余裕がない状況も存在し，時刻印方式を基にした制御が有効であると考えられる．

3.3 スケジューリング方式

トランザクションのスケジューリングを次の 2 つに分けて考える．

(1) ハードデッドラインを持つトランザクションを静的にスケジューリングする場合

(2) ソフトデッドライン，またはファームデッドラインを持つトランザクションを動的にスケジューリングする場合

ここでハードデッドラインとは応答時間制限を満足できない時にシステムが障害状態になるものを意味し，ソフトデッドラインとは応答時間制限をある程度過ぎてもトランザクションに価値が残るもの，ファームデッドラインはそれを過ぎると価値はなくなるがシステム障害状態にはならないものを意味する．

3.3.1 静的スケジューリング

ハードデッドラインを持つトランザクションは，次のような性質を持っていると考えられる．

- トランザクションの発生は周期的である．
- 共有オブジェクトのアクセスパターンやタイミングがほぼ判明している．
- トランザクションのアボートは致命的である．
- 共有オブジェクトに対する競合が比較的少ない．

以降ではこれらの特徴を前提として議論を進めて行く．またトランザクションの実行時間については，I/O 操作による遅延も既知のものとする．

トランザクションが周期的に発生するので，優先度の割り付け方法として，周期の小さなものほど高い優先度を与える RM を採用する．

ハードデッドラインが要求される処理では，完了時刻を予測不可能にするトランザクションのアボートは許されない．従って，後発のトランザクションがアクセスを要求するオブジェクトに対し，それよりも先発のトランザクションが後に競合アクセスを行なうことが分かっている場合は，その要求を先発トランザクションのコミットまで待機させる必要がある．また同様の理由により，多重版時刻印方式では保守的時刻印方式を使用すべきである．

そこでトランザクション開始時に，自分がアクセスする予定のオブジェクト (多重版では更新オブジェクトのみ) に対し予約を行なうことにする．トランザクションがオブジェクトにアクセス要求を発行したとき，そのオブジェクトが自分よりも時刻印の小さいトランザクションによって予約されていれば，その予約を行なったトランザクションのコミットまで要求は待たされる．このとき，優先度逆転現象を防止するため，待機するトランザクションの優先度の方が高ければ，当該オブジェクトを予約したトランザクションにその優先度を継承する．

ここで述べた予約の操作はロックをかけることに類似しているが，ロック方式との違いは，予約されたオブジェクトに対する要求 (多重版では参照要求のみ) が待たされるのは，予約を行なったトランザクションの時刻印よりも大きな時刻印を持つトランザクションが要求を行なった時のみという点である．このような一方向性があるために，この予約操作が原因でデッドロックのような相互待ち状態が発生することはない．この方法は，アクセスするオブジェクトがあらかじめ予測できれば，時刻印方式でもすべてのアボートを回避することができることを示している．従って，新たなトランザクションがスケジュール可能か否かの判定には，他の優先度の高いトランザクションの実行時間だけでなく，この予約による待機時間も考慮しなければならない．

3.3.2 動的スケジューリング

ソフトデッドライン及びファームデッドラインを持ったトランザクションを動的にスケジューリングするのは次のような状況であると考えられる．

- トランザクションの発生は非周期的である．
- 共有オブジェクトのアクセスパターンやタイミングは予測不能である．
- トランザクションのアボートはある程度認められる．
- 共有オブジェクトに対する競合が多いこともあり得る．

この条件下ではいくつかの優先度割り付け方法が選択可能であるが，ここでは EDF を用いるものとする．

時刻印方式を用いてスケジューリングを行なうときに問題となるのは，遅いアクセスによって高優先度のトランザクションがアボートしてしまうことである．これは I/O 操作による待ち時間に，後発のトランザクションが競合アクセスを行なうことにより発生する．このアボートを回避するための手段として以下の 2 通りが考えられる．

(1) 悲観的手法

I/O 操作などにより自分から CPU を放棄する場合には，他に優先度が高いトランザクションがあったとしても，常に自分の時刻印よりも小さな値の時刻印を持つトランザクションの中から処理を譲るトランザクションを選択する．(図 4参照)．

(2) 楽観的手法

I/O 操作などにより自分から CPU を放棄する場合には，たとえ自分の時刻印より大きな値の時刻印を持つトランザクションであっても処理を譲ることを許す．ただしこの場合には処理を譲ってもらったトランザクションの終了までコミットは待たされる．競合が発生した時には優先度の低いトランザクションがアボートしなければならない．(図 5参照)．

どちらの方式も現時点で最高優先度を持つトランザクションがアボートすることはない．悲観的手法は楽観的手法に比べ，処理を譲ってもらった後発のトランザクションが違法なアクセスをする予定の場合でもアボートすることはない．しかし，図 4中の TR2 以降に発生したすべてのトランザクションはある程度高い優先度を持っていたとしても処理を譲ってもらえず，優先度を反映した処理が実現できない．また，楽観的手法を用いた場合はその逆である． 従って，デッドラインを多少ミスしても確実にトランザクションをコミットさせた方が良いのか (悲観的手法)，全体的な効

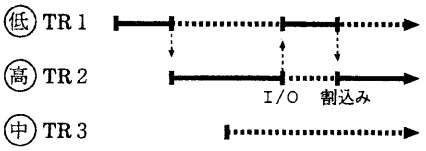

図 4 悲観的手法

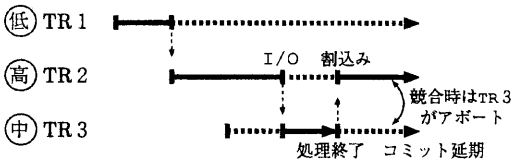

図 5 楽観的手法

率及びデッドラインを重視して，もし不幸にも競合が起こってしまった場合にはアボートも仕方がないと考えるか (楽観的手法) のトレードオフといえる．これはあらかじめ予想される競合の頻度にも依存する．選択の目安として図 6 のアルゴリズムの使用を考える．

```
if T_P request I/O
  then SelectMethod();
endif

SelectMethod(){
T = the transaction with the next highest priority
if TS(T) > TS(T_P)
  then
  if (E_{T_P} − S_{T_P}) > d_T − (t + E_T − S_T)
    then process T ; /* optimistic method */
    else SelectMethod();
  endif
  else process T ;    /* pessimistic method */
endif
}
```

E : 全処理時間 (runtime estimate)
S : 既処理時間 (serviced time)
d : 締切時刻 (deadline)
t : 現時刻 (current time)

図 6 手法選択アルゴリズム

このアルゴリズムでは基本的に悲観的手法を用いる．悲観的手法でデッドラインを満たせないトランザクションが出現したときに，楽観的手法に切り替えるものである．T_Pは，現在 CPU を与えられて処理を行なっているトランザクションを意味する．

この動的スケジューリングにおいて，優先度の継承は行なわない．従って悲観的手法では，I/O の待ち時間に大きな時刻印を持つトランザクションに処理を譲らないため，優先度逆転が発生する．しかしこれは大きな問題とはならない．優先度が逆転する時間は，I/O 操作終了の割り込み通知により処理が戻されるまでの確定的な時間であることが保証されるためである．

4 おわりに

本論文では，トランザクションの優先度処理に対応するため，時刻印方式を基に改良を加えた並行処理制御方式 PTO と，時刻印方式を基本としたスケジューリング手法について述べた．PTO は他の各方式との性能比較を行なった結果，ロックを用いた方式に比べ時刻印方式の長所を損なうことなく，従来の時刻印方式よりも全体的に高いコミット率を実現することが判明した．また排他制御に時刻印方式を用いてスケジューリングを行なうことで，ロック方式に比べ他のトランザクションによりブロックされる機会が減少し，デッドロックや優先度逆転の問題も解消される．時刻印方式を用いて動的スケジューリングを行なう際には，アボートの危険性があるためトランザクションの終了が予測不可能になるという問題がある．しかし，事前に共有オブジェクトのアクセスパターンが分からない場合でも効率の良い処理を実現できるという利点がある．

今後はスケジューリング手法の実装を進めるとともに，分散環境上でのスケジューリング問題について検討して行くつもりである．

参考文献

[1] Lih Chyun Shu and Michal Young: "Correctness Criteria and Concurrency Control for Real-Time Systems: A Survey", *Technical Report SERC-TR-131-P, Purdue University* (1992).

[2] Robert Abbott and Hector Garcia-Molina: "Scheduling Real-time Transactions: a Performance Evaluation", *Proceedings of the 14th VLDB Conference*, pp. 1-12 (1988).

[3] Yi Lin and Sang H.Son: "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order", *Proc. of 11th Real-Time System Symposium*, pp.104-112 (1990).