# 適応型時刻印方式に基づく同時実行制御方式†

國枝和雄††* 畑田孝幸††**
大久保英嗣††*** 津田孝夫††

本論文では，分散システムにおける同時実行制御方式として適応型時刻印方式を提案する．従来，同時実行制御方式として2相ロック方式や時刻印方式が提案されている．しかし，様々な種類のトランザクションに対して，単一の方式で効率良い制御を行うのは困難である．適応型時刻印方式では，トランザクションごとに，その性質や実行状況に応じて，複数の方式の中から最適な方式を選択することができる．これによって，トランザクションの後退復帰や，長時間トランザクションの影響による他トランザクションの実行遅延などの発生を抑え，その実行時間を短縮させることが可能となる．適応型時刻印方式は，(1)本論文で新たに提案する予約型時刻印方式，(2)排他ロック式時刻印方式，(3)従来型多重版時刻印方式の3方式からなる．各方式は，それぞれ(1)遅い更新を持つトランザクションやトランザクションの多重度が高くアクセス競合が起こりやすい状況，(2)長時間トランザクション，(3)競合がほとんど発生しない状況，においた優れた特性を持っている．この適応型時刻印方式をデータベースオペレーティングシステム *μOPT-R* の機能として実現し性能測定を行った．その結果，トランザクションの多重度が高い状況で予約型時刻印方式を用いることによって，トランザクションの実行時間が大幅に短縮されることが示された．また，長時間トランザクションに対して排他ロック式時刻印方式を適用することによって，競合する他のトランザクションの実行時間が短縮されることも確認した．

## 1. はじめに

分散データベースをはじめとする分散システムにおいて，トランザクションを用いた処理の抽象化は重要である．トランザクションは，原子性を備えることにより，ユーザから分散処理の複雑さを隠蔽する．このトランザクションを実現するための基本的技術として，トランザクションの同時実行制御が必要となる．同時実行制御は，並行して実行されるトランザクションの論理的な実行順序を決定し，トランザクション間の処理の無矛盾性を保証するものである．

本論文では，同時実行制御を行う方式として**適応型時刻印方式**を提案する．適応型時刻印方式は，多重版時刻印方式に基づく複数の方式の集まりからなっている．トランザクションの構成時に，それらの方式の中からトランザクションの性質に応じて方式を選択することによって，時刻印方式の問題点であるトランザクションの後退復帰の発生を低減させることができる．また，トランザクションの実行時においても，システムの負荷の状況やそれまでの実行状況（後退復帰の有無など）に応じて動的に方式を選択することが可能である．本論文では，適応型時刻印方式に属する一方式として，従来型の多重版時刻印方式にデータ更新の「予約」の概念（以下，これを「データ予約」と呼ぶ）を採り入れた予約型時刻印方式を新たに提案する．予約型時刻印方式では，データ更新操作の失敗に伴うトランザクションの中断確率を減少させることが可能である．さらに，長時間トランザクション（long-lived transaction）に適した方式として，時刻印方式とデッドロックフリーの2相ロック方式とを組み合わせた排他ロック式時刻印方式も実現した．以上の各方式は，従来型の時刻印方式を含めて，互いに矛盾なく組み合わせて用いることができる．本論文では，まず2章で従来型の各方式の問題点を挙げる．次に3章では，適応型時刻印方式の提供する各方式について述べ，トランザクションの性質に応じた予約型時刻印方式および排他ロック式時刻印方式の適用法についても議論する．また，4章で実現方式について説明する．最後に，5章で実験システム上で行った性能測定の結果を示し，本方式の有効性を示す．

## 2. 同時実行制御

### 2.1 トランザクションと直列化可能性

データベースでは，データ操作によってデータに矛盾を引き起こすことのないように，トランザクション

† A Concurrency Control Method Based on the Adaptive Time-Stamp Ordering by KAZUO KUNIEDA, TAKAYUKI HATADA, EIJI OKUBO and TAKAO TSUDA (Department of Information Science, Faculty of Engineering, Kyoto University).
†† 京都大学工学部情報工学科
* 現在 日本電気(株)
Now with NEC.
** 現在 (株)ワコム
Now with WACOM.
*** 現在 立命館大学理工学部情報工学科
Now with Department of Computer Science and Systems Engineering, Faculty of Science and Engineering, Ritsumeikan University.

の概念を用いる．トランザクションは，BEGIN と END で囲まれた一連のデータ操作であり，以下のような特性を備える[1]．

（1） 一貫性 (consistency)

データを一貫した状態から別の一貫した状態へ遷移させる．

（2） 原子性 (atomicity)

BEGIN から END までのデータベース操作は，完全に実行されるか，または全く実行されないかのいずれかである．

（3） 持続性 (durability)

いったん完了すると，それを破棄することはできない．

トランザクションを完了することをコミット (commit) といい，途中で破棄することをアボート (abort) という．トランザクションをアボートする場合，(2) の原子性により，そのトランザクションが更新したデータをすべてトランザクション実行前の状態に戻さなければならない．これをトランザクションの後退復帰 (rollback) という．また，分散データベースでは，性能を上げるために複数のトランザクションが並行に実行される．この時，データベースの一貫性を保つためには，トランザクションのスケジュールが直列化可能 (serializable) でなければならない．すなわち，トランザクションの並列スケジュールが，ある直列スケジュールに等価でなければならない．トランザクションの直列化可能性を保証するために，何らかの規則によってデータへのアクセスを制限することを同時実行制御 (concurrency control) という．

### 2.2 従来方式の比較

同時実行制御方式の伝統的な手法について特質を挙げ比較する．

（1） 2相ロック方式

2相ロック方式 (2-Phase Locking) とは，データにアクセスする前に対象となるデータをロックし，トランザクション終了時にロックを解除する方式である．この方式の特長は，比較的容易に実現できることである．しかし，トランザクションの衝突が多くなるとともに，待ち状態のトランザクションが急激に増加しシステム全体のスループットが低下する．また，デッドロックを発生する可能性があり，それに対する処理が必要となる．その一つとして，犠牲となるトランザクションを選出し，それを中断再実行することによってデッドロックを解消する手法が考えられる．この場合には，再実行によって再びデッドロックを引き起こし，中断再実行の繰り返しに陥る可能性がある．この状態をライブロック (livelock) と呼ぶ．ライブロックは再実行を用いた処理では不可避な問題である．

（2） 多重版2相ロック方式

通常の2相ロック方式では，更新のために排他ロックしたデータに対する更新および参照は許されない．この制限は，データを多重版化することで改善される．多重版化とは，各トランザクションが完了した時点のデータの内容を，その時点のバージョンとして保存することである．多重版2相ロック方式 (MV2PL: Multi-Version 2-Phase Locking) では，データの参照は，他のトランザクションからロックされていない最新のバージョンに対して行われる．したがって，データの参照についての制限がなくなり，参照を瞬時に行うことが可能となる．この方式での問題は，更新したバージョンのロックを解除する時に生じる待ち状態である．これは，トランザクションの実際の処理順序と直列化スケジューリング上の順序が逆転するために発生する．すなわち，あるバージョンに対するロックを解除する時に，それより古いバージョンを参照したトランザクションが存在すると，その終了を待ってからロックを解除しなければならない．そのため，長時間トランザクションと通常のトランザクションが混在するような状況では，長時間トランザクションの終了を待って他のトランザクションが長時間待たされる可能性がある．

（3） 時刻印方式

時刻印方式 (Time-Stamp Ordering) はデータに対するロック操作を行わず，時刻印に基づいて同時実行制御を行う．その処理は以下のように行われる[2]．

（a） 各トランザクションは，それが発行された時刻を示す時刻印 $TS$ を持つ．

（b） 各データ項目は，最後にデータを更新したトランザクションの時刻印 $WTS$ と，最後に参照したトランザクションの時刻印 $RTS$ を持つ．

（c） トランザクションがデータを参照する場合，$TS<WTS$ ならばそのトランザクションをアボートし，さもなければ参照を実行する．

（d） トランザクションがデータを更新する場合，$TS<\max(RTS, WTS)$ ならばそのトランザクションをアボートし，さもなければ更新

を実行する．

この方式においては，トランザクションは時刻印の順にしたがって直列化可能となる．また，トランザクションがデータをロックすることはないのでデッドロックは起こらない．しかし，不当な参照または更新をしようとするとトランザクションのアボートが発生し，後退復帰処理が必要となる．そして，後退復帰したトランザクションには新たな時刻印を付加して再実行する．この時，参照や更新に再度失敗するとアボート・再実行を繰り返すことになる．すなわち，この方式においてもライブロックの発生が問題となる．

（4） 多重版時刻印方式

時刻印方式においても，データを多重版化することでトランザクションがアボートする確率を小さくすることができる．これを，多重版時刻印制御方式（MVTO：Multi-Version Time-Stamp Ordering）と呼ぶ（処理アルゴリズムは次章で示す）．MVTO の参照操作では，時刻印を比較して複数のバージョンの中から参照可能なものを選んで参照することができる．したがって，参照の失敗をほとんどなくすことができる．トランザクション $T_i$ における参照が失敗するのは，トランザクション $T_k$ が更新中のバージョンを $T_i$ を先読みし，なおかつ $T_k$ がアボートした場合のみである（3.1 節参照）．しかし，更新操作は，より大きな時刻印を持つ後発のトランザクションが既に該当データを参照していると失敗する．このような遅い更新（delayed write）の起こる可能性は，トランザクション実行時間が長いほど大きくなる．すなわち，長時間トランザクションが存在する場合，遅い更新は大きな問題となる．また，あるトランザクションの後退復帰が連鎖的に他のトランザクションの後退復帰を引き起こすカスケーディングアボート（cascading abort）も問題となる．

このほかにも多くの同時実行制御方式が提案されている[3),4)]が，そのほとんどが以上の四つのクラスに属すると考えられる．

分散処理を行う場合，処理の多重度を上げることによって処理効率を高めることが重要である．この点から考えると多重版処理を行う多重版2相ロック方式と多重版時刻印方式が有利である．しかし，上で述べたように両方式ともすべての場合に効率が良いとは限らない．以下に，両方式の問題点をまとめておく．

（1） 多重版2相ロック方式の問題点

- デッドロックに対する処理が必要である．また，デッドロックの処理によってはライブロックを発生する可能性がある．
- 長時間トランザクションが存在すると，他のトランザクションが長時間待たされる可能性がある．

（2） 多重版時刻印方式の問題点

- 長時間トランザクションなどで遅い更新がある場合，後退復帰の発生する可能性が高い．
- カスケーディングアボートが発生すると効率が大きく低下する．
- 後退復帰後にライブロックを生じる可能性がある．

## 3. 適応型時刻印方式

分散環境で並行にトランザクションを実行する場合には，前章で述べたように多重版時刻印方式が比較的有利である．しかし，依然として遅い更新，カスケーディングアボートなどの問題が残される．そこで，多重版時刻印方式についてさらに詳細な考察を行い，このクラスに属する方式を応用することによって，残された問題を解決する．

### 3.1 保守的時刻印方式と積極的時刻印方式

まず，多重版時刻印方式の一般的処理アルゴリズムを示す[4)]．

各トランザクション $T_i$ には，開始時にシステムから基本時刻印 $TS(T_i)$ と最大時刻印 $TS(T_i)+\sigma_i$ が与えられる．基本時刻印はトランザクションの開始時刻を表し，最大時刻印はデッドラインを表す．トランザクションが終了すると確定時刻印 $CTS(T_i)$ が与えられる．また，トランザクションの発行する参照・更新要求に対しサイト $l$ のシステム $S_l$ は承認時刻印 $ackTS^{S_l}(T_i)=(tack_1^{S_l}(T_i), tack_2^{S_l}(T_i))$ を返す．ここで，$tack_1^{S_l}(T_i)$ と $tack_2^{S_l}(T_i)$ は，サイト $l$ において参照・更新要求を実行可能とする時刻印 $TS(T_i)$ の最小値および最大値である．さらに，データ $x$ の各バージョンを $\langle x^0, x^1, \cdots, x^g\rangle$ とし，バージョンの履歴を以下のように定義する．

$$H_{rw}(x)=\{(WTS(x^0), RTS(x^0), ERTS(x^0)),\ (WTS(x^1), RTS(x^1), ERTS(x^1)),\ \vdots\ (WTS(x^g), RTS(x^g), ERTS(x^g))\}$$

ここで，

$WTS(x^k)$ ：$x^k$ を作成したトランザクションの時刻印

$RTS(x^k)$ ：$x^k$ を参照し，なおかつコミットされ

た中で最も新しいトランザクションの時刻印

$ERTS(x^k)$：$x^k$ を参照した中で最も新しいトランザクションの時刻印

である．このアルゴリズムでは，$WTS(x^k)$ および $RTS(x^k)$ はトランザクションの実行にしたがって徐々に決まることになる．そのために，$ERTS(x^k)$ を記録しておく必要がある．

トランザクションからの各要求に対する処理アルゴリズムを示す．ここで $\nu$ は時刻印の刻み幅である．

（1） トランザクション開始

トランザクションに基本時刻印 $TS(T_i)$，最大時刻印 $TS(T_i)+\sigma_i$ を与える．

（2） データ更新

（a） すでにコミットされたバージョン $x^k$ で，

$$WTS(x^k)\leq TS(T_i)\leq TS(T_i)+\sigma_i\leq RTS(x^k)$$

を満たすものがあれば，要求を破棄する．

（b） $WTS(x^k)<TS(T_i)$ を満たす最大の $WTS(x^k)$ を持つバージョン $x^k$ に対して，

$$RTS(x^k)<ERTS(x^k)$$

すなわち，$x^k$ を参照したトランザクションでコミットされていないものが存在する場合，そのトランザクションが終了または中断されるまでこの更新要求をブロックし，その後もう一度同じアルゴリズムを適用する．最終的に $RTS(x^k)=ERTS(x^k)$ になるまで要求は待ち状態にされる．

（c） （a）および（b）に該当しない場合は，即座に更新を行う．

（d） 最後に $ackTS^{S_l}(T_i)$ を生成する．すなわち

$$ackTS^{S_l}(T_i)=(tack_1^{S_l}(T_i),\ tack_2^{S_l}(T_i))$$

において

$$tack_1^{S_l}(T_i)=\max\{TS(T_i),\ RTS(x^k)+\nu\}$$

$$tack_2^{S_l}(T_i)=\begin{cases}\min\{TS(T_i)+\sigma_i,\\ \quad WTS(x^{k+1})-\nu\}\\ \quad (x^{k+1}\text{ が存在する場合})\\ TS(T_i)+\sigma_i \quad (\text{その他の場合})\end{cases}$$

また，時刻印

$$WTS(x^p)=RTS(x^p)=\max\{TS(T_i),\ RTS(x^k)+\nu\}$$

を持つ未確定バージョン $x^p$ を生成する．

（3） データ参照

$WTS(x^k)<TS(T_i)$ を満たす最大の $WTS(x^k)$ を持つバージョン $x^k$ に対して，$x^k$ が確定していなければ，このバージョンを作成したトランザクションが終了または中断されるまで要求を待ち状態にし，その後でもう一度同じアルゴリズムを適用する．

もし確定されていれば即座に参照を行い，最後に $ackTS^{S_l}(T_i)$ を生成する．すなわち

$$ackTS^{S_l}(T_i)=(tack_1^{S_l}(T_i),\ tack_2^{S_l}(T_i))$$

において

$$tack_1^{S_l}(T_i)=TS(T_i)$$

$$tack_2^{S_l}(T_i)=\begin{cases}\min\{WTS(x^{k+1})-\nu,\ TS(T_i)+\sigma_i\}\\ \quad (x^{k+1}\text{ が存在する場合})\\ TS(T_i)+\sigma_i \quad (\text{その他の場合})\end{cases}$$

また，

$$ERTS(x^k)<tack_2^{S_l}(T_i)$$

であれば

$$ERTS(x^k)=tack_2^{S_l}(T_i)$$

とする．

（4） トランザクション終了

次の条件を満たすように確定時刻印 $CTS(T_i)$ を与える．

$$\max_l\{tack_1^{S_l}(T_i)\}\leq CTS(T_i)\leq\min_l\{tack_2^{S_l}(T_i)\}$$

このような $CTS(T_i)$ が決まらない場合はトランザクションを中断する．もし決まれば，このトランザクションが更新したすべてのデータに対して

$$WTS(T_i)=CTS(T_i)$$

とする． □

このアルゴリズムで $\sigma_i=0$ に設定したものが，一般に用いられる Reed のアルゴリズム[5]である．このアルゴリズムは，参照要求の処理において最新のバージョンがコミットされていない場合，その要求が待たされることから保守的時刻印方式（CMVTO：Conservative MVTO）と呼ばれる．これは，最新のバージョンを作成したトランザクションが後退復帰によって破棄された場合に備えるためである．一方，最新バージョンがコミットされていない時に先読みによって参照操作を実行する積極的時刻印方式（AMVTO：Aggressive MVTO）もある．この方式では，トランザクション終了時に，先読みしたバージョンが破棄されていないことを確認しなければならない．すなわ

ち，先読みしたバージョンがコミットされるまで終了操作は待たされることになる．また，もし破棄されたバージョンがあれば，このトランザクションも連鎖的にアボートされる（カスケーディングアボート）．これは，AMVTO のように先読みを行う方式では回避することはできない．端末における入出力とデータベースの参照・更新を交互に実行するようなトランザクションでは，常にトランザクションの後退復帰が可能であるとは限らない．したがって，そのようなトランザクションは CMVTO で実行されるべきである．また AMVTO では，カスケーディングアボートの確率が高くなるが，実行中に余分な待機をせずに演算処理を行えるため，比較的データアクセスの少ないトランザクションの場合には実行速度の向上が望める．

### 3.2 予約型時刻印方式

トランザクションの終了確率を高めるためには，2.2 節で述べた遅い更新が問題となる．われわれは，従来の時刻印方式にデータ予約の概念を取り入れることによってこれを解決した．これを予約型時刻印方式と呼ぶ．予約型時刻印方式では，実行中のトランザクションは，優先して更新を行いたいデータに対して予約操作を行うことができる．時刻印 $TS$ を持つトランザクションが，あるデータ項目に予約操作を行うと，$TS$ より大きな時刻印を持つトランザクションからのデータへの参照要求は待ち状態にされ，予約を行ったトランザクションの終了によって再開される．したがって，予約操作によって $TS$ の大きなトランザクションの参照によって引き起こされる更新の失敗が回避されることになる．また，予約操作の特徴として，更新の失敗する確率を更新要求を出すトランザクションのほうで制御できる点が挙げられる．すなわち，トランザクション設計者が，遅い更新を行う可能性のあるデータに予約操作をすることで，トランザクションの終了確率を高めることが可能となる．ここで，予約操作によってデッドロックが生じるか否かという問題があるが，予約によって待たされるトランザクションは常により大きな時刻印を持つものだけなので，デッドロックを生じるような循環待ちが起きることはない，予約型時刻印方式では，3.1 節で述べたアルゴリズムに，以下の処理が追加される．

（1） 予約操作のために，各バージョンに予約時刻印集合 $\mathcal{R}(x^k)$ を付加する．ここで，$\mathcal{R}(x^k)$ は $x^k$ を予約しているトランザクションの時刻印の集合である．

（2） データ予約

（a） すでにコミットされたバージョン $x^k$ で，$WTS(x^k) \leq TS(T_i) \leq TS(T_i)+\sigma_i \leq RTS(x^k)$ を満たすものがあれば，要求を破棄する．

（b） $WTS(x^k) < TS(T_i)$ を満たす最大の $WTS(x^k)$ を持つバージョン $x^k$ に対して，その予約時刻印集合 $\mathcal{R}(x^k)$ に $TS(T_i)$ を加える．

（3） データ参照

3.1 節の（3）データ参照のアルゴリズムを実行する前にまず以下の処理を行う．

（a） $WTS(x^k) < TS(T_i)$ を満たす最大の $WTS(x^k)$ を持つバージョン $x^k$ に対して，$\min(\mathcal{R}(x^k)) < TS(T_i)$ ならば，この条件が成立しなくなるまで要求を待ち状態にする．

（b） さもなければ次の処理に移る．

（4） トランザクション終了

3.1 節の（4）トランザクション終了処理の先頭において，$\mathcal{R}(x^k)$ から終了するトランザクションの時刻印 $TS(T_i)$ を取り除く．

### 3.3 排他ロック式時刻印方式

2.2 節で述べた長時間トランザクションの問題は，予約操作によっても解決されない．トランザクション自体のアボートは回避できたとしても，他のトランザクションを非常に長く待たせる結果になってしまうからである．この問題は，文献 3) で提案されている時刻印方式と 2 相ロック方式を組み合わせた協調型方式によってある程度解決できる．この方式を排他ロック式時刻印方式と呼ぶ．長時間トランザクションは，実行に先だって更新および参照するデータ項目を排他ロック（exclusive lock）する．このロックは，排他ロック式時刻印方式を適用しているトランザクション間では，データアクセスについての排他処理を実現する．また，それ以外のトランザクションに対しては，以下の効果を持つ．

- ロックされたデータへの他からの更新操作は破棄される．したがって，ロックされている間は新しいバージョンは生成できない．
- ロックされたデータへの他からの参照操作は，通常の時刻印処理によって行われる．
- ロックをかけたトランザクションにおけるデータの更新を，ロック解放時まで，他のトランザクションから隠蔽する．言いかえると，ロックして

いる間は，他のトランザクションにとって最新のバージョンはロックをかけた時のバージョンとなる．そのため，ロックをかけたトランザクションが更新したデータには，トランザクション開始時の時刻印ではなく，ロック解放時の時刻印を生成時刻として付加する．

このように，ロックはデータのバージョンを一定期間固定するにすぎず，時刻印処理自体に影響を与えることはない．したがって，この方式は時刻印方式と混在させることが可能である（**図1参照**）．

ここで，2相ロック方式ではデッドロックの発生が問題となるが，本方式のロックはデッドロックの原因となることはない．なぜならば，本方式においてロックが競合した場合には，後から要求されたロックは破棄され，トランザクションに待ち状態を生じさせないからである．また，トランザクション $T1$ が $A \rightarrow B$ の順でデータをロックし，トランザクション $T2$ が $B \rightarrow A$ の順でデータをロックするような場合に，$T1$ および $T2$ が共に破棄されることも問題である．われわれのシステムでは，ロッキングフェーズの処理を論理的に不可分なものとし，他のトランザクションにおけるロッキングフェーズと排他的に実行することによってこれを回避している．

### 3.4 各方式の選択

適応型時刻印方式では，以上で説明した予約型 AMVTO，予約型 CMVTO，従来型 AMVTO，従来型 CMVTO，排他ロック式 MVTO の各方式から，任意の方式を選択してトランザクションに適用することができる．予約型方式と従来型方式の相違点は，トランザクション中の予約操作の有無だけであり，実際には，ユーザはトランザクション起動時に BEGIN の引数として，AMVTO，CMVTO，排他ロック式 MVTO のいずれかを指定する．特に指定がない場合には，AMVTO がシステムによって設定される．AMVTO あるいは CMVTO を選択した場合には，END に至るまでの任意の時点でデータを予約することができる．排他ロック式 MVTO を選択した場合には，BEGIN の直後で更新および参照の対象となるすべてのデータをロックしなければならない．いずれの方式を用いた場合も，トランザクションが参照・更新要求を行うと次の手順で要求の可否が判定される．

（1） データが既にロックされているか否かに基づく判定．

（2） データが既に予約されているか否かに基づく判定．

（3） 多重版時刻印方式の時刻印に基づく判定．

ここで，各判定は**表1**に示した基準で行われる．表1は，左見出しで示すデータ操作要求が発行された時に，同じデータに対して上見出しのトランザクションが既に操作を行っている場合に，システムが取るべき処理内容を表している．上見出しの先発トランザクションとは，要求を現在行っているトランザクションより古い時刻印を持つトランザクションのことであり，後発トランザクションは，逆に新しい時刻印を持つトランザクションを表す．

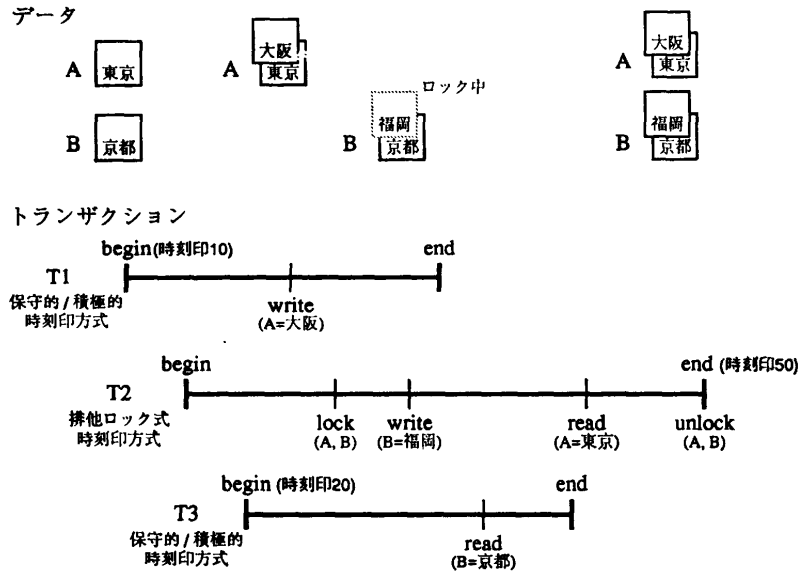



**図 1** 排他ロック式時刻印方式による処理例

Fig. 1 Example of processing by exclusive locking time-stamp ordering.

このように，いずれの方式を選択しても，トランザクションの参照・更新操作は，最終的に時刻印処理に基づいて可否が判定され，すべてのトランザクションは時刻印方式に従って直列化可能性が保証される．排他ロック式 MVTO を選択した場合には，これにロック操作が加わり，予約型の AMVTO および CMVTO を選択した場合には予約操作が加わる．しかし，これらの操作はデータを操作するのではなく，他からの要求を待機・破棄させる効果があるにすぎないので，時刻印に基

表 1 データ操作の種別と動作
Table 1 Types and actions of data operations.

| | | 先発トランザクション | | | | 後発トランザクション | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 参照 | 更新 | 予約 | ロック | 参照 | 更新 | 予約 | ロック |
| 積極的時刻印方式 | 参照 | ○ | ○ | △ | ○ | ○ | ○ | ○ | ○ |
| | 更新 | ○ | ○ | ○ | × | × | ○ | ○ | × |
| | 予約 | ○ | ○ | ○ | × | × | ○ | ○ | × |
| 保守的時刻印方式 | 参照 | ○/△ | ○/△ | △ | ○/△ | ○/△ | ○/△ | ○/△ | ○/△ |
| | 更新 | ○ | ○ | ○ | × | × | ○ | ○ | × |
| | 予約 | ○ | ○ | ○ | × | × | ○ | ○ | × |
| 排他ロック時刻印方式 | 参照 | ○/△ | ○/△ | ○/△ | — | ○/△ | ○/△ | ○/△ | — |
| | 更新 | ○ | ○ | ○ | — | ○ | ○ | ○ | — |
| | ロック | ○ | ○ | ○ | × | ○ | ○ | ○ | × |

○：実行（アクセス），△：待機，×：破棄，—：要求不可，○/△：データが確定していれば実行し，未定ならば待機

づく直列化可能性には影響を与えない．したがって，異なる方式を選択したトランザクションを混在させることが可能である．

以上で述べたトランザクションの設計時あるいは実行時における方式選択によって効率的な処理を行うには，以下のような応用が考えられる．

（1） 中断したトランザクションの自動再実行機能

AMVTO や CMVTO で更新を失敗したデータ項目に予約を施し，終了する確率を高めた上でトランザクションを再実行する．

（2） 統計情報を利用した制御方式の自動切り替え機能

アクセス頻度の高いデータ項目に対して自動的に予約をかけたり，システムの負荷に応じて方式を切り替えること等が考えられる．

（3） タスク管理機能との結合

ユーザやシステムが，トランザクションに優先度を設定することで方式を選択する．

## 4. 実　　現

以上で述べた適応型時刻印方式を，現在われわれが開発中のデータベースオペレーティングシステム *μOPT-R*[6] の機能として実現した．*μOPT-R* の対象とするシステムは，複数のサイトが多重アクセスバス方式の LAN によって結合された分散データベースシステムである．各サイトは，単一のプロセッサ，ローカルメモリ，ローカルディスク，通信装置および入出力装置より構成される．したがって，サイト間にはメモリおよびクロックの共有はなく，サイト間通信はメッセージ交換によってのみ可能となる．*μOPT-R* のハードウェア構成は，分散システムの各サイトにパーソナルコンピュータ（PC-9801 RA 5：NEC 製，CPU：80386，clock：16 MHz）および LAN 制御ボード（SIC 98：Allied Telesis 社製），それらを結合する LAN にイーサネット（IEEE 802.3）を用いている．また，ネットワーク通信のプロトコルは TCP/IP である．*μOPT-R* は，ユーザインタフェース，トランザクション管理部，データベース制御部，分散処理システムサーバ群およびオペレーティングシステム核から構成されている．適応型時刻印方式による同時実行制御部は，*μOPT-R* におけるシステムサーバの一つとして実現した．

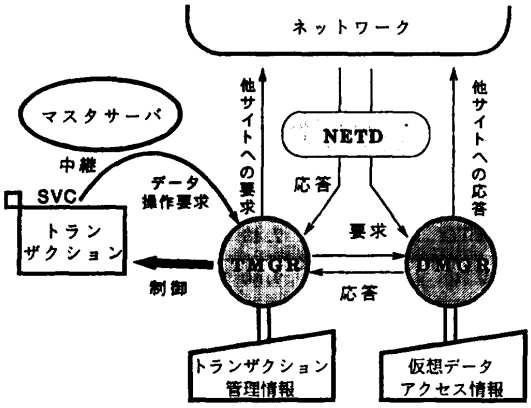
### 4.1 同時実行制御サーバの構成

同時実行制御サーバ（以下，CC サーバと記す）は，トランザクションからの要求を受け付けるマスタサーバ，トランザクション情報を管理するトランザクションマネージャ（TMGR），データのアクセス状態を管理するデータマネージャ（DMGR）および他サイトからの要求を非同期に受け付けるネットワークデーモン（NETD）から成る（図 2 参照）．トランザクションから CC サーバへの要求は Supervisor Call (SVC) として送られる．また，TMGR と DMGR は，相互に通信しながら要求に応じた処理を行う．これらのメッセージ交換に必要な，各サイトの NETD と TMGR，DMGR の間の通信コネクションはシステム起動時に張られる．

### 4.2 データ構造

（1） トランザクション管理データ

TMGR は自サイトで実行中のトランザクションに関するデータ（識別番号，制御方式，時刻印，実行履




TMGR: Transaction Manager
DMGR: Data Manager
NETD: Network Daemon

図 2 同時実行制御サーバの構成
Fig. 2 Configuration of the concurrency control server.

歴等）をキューイングして管理する．このデータは，トランザクション開始時に初期化され，トランザクションのアボートまたはコミットによって削除される．トランザクションは実行開始時に，何れの同時実行制御方式を使用するかを指示しなければならない．後退復帰処理は，保持している実行履歴を参照して，個々の要求の破棄を DMGR に指令することによって行われる．

（2） アクセス管理データ

CC サーバは複数の仮想データ項目を持ち，これを同時実行制御の対象とする．この仮想データ項目を実際のデータ項目に写像することによって実際の同時実行制御が行われる．したがって，この写像を変更することによって同時実行制御の単位を変更することができる．

各仮想データ項目に対するアクセス情報（時刻印，アクセス状態，アクセス履歴等）は DMGR が管理する．この仮想データに対して，トランザクションは COPY（データ内容をトランザクション内部に複写する），RESERVE（データの予約を行う），PUT（トランザクションの内部からデータを書き出す），LOCK（データをロックする）の操作を行うことができる．

## 5. 評　　価

本章では，前章で述べたシステム上で行った適応型時刻印方式の性能評価について述べる．

### 5.1 予約型時刻印方式の評価

実験は，前章で述べたシステムにおいてサイト数を3とし，100 個のデータとそれを操作するトランザクションを用いて行った．各トランザクションは，無作為に選んだ 15 個のデータを参照した後に5個のデータを更新する．トランザクションが破棄された場合は，自動的にデータを選び直して再実行する．現実のトランザクションでは同一のデータで再実行すべきであるが，ここでは，予約を行わない場合の後退復帰回数を測定可能範囲に納めるためにこの方法で測定を行った．ただし，予約を行う方式では**表2**に示すとおり後退復帰をほとんど生じないので測定結果への影響は無視できると考えられる．なおトランザクションには均等に遅延が入れてあり，単独での実行時間は2秒である．実験は，AMVTO および CMVTO とそれら各方式にデータ予約を採り入れた4方式について，トランザクション数を変化させて行った．各トランザクションを1秒間隔で順次異なるサイトで起動し，それが完了するまでの時間を測定した．データ予約を行う方式では，更新を行うすべてのデータについて予約を行っている．**図3**に各方式における1トランザクションの平均実行時間の変化を示す．また，表2はその時の各トランザクションの平均後退復帰回数である．平均後退復帰回数は全トランザクションで発生した後退復帰の回数をトランザクション数で単純に除算したものである．

まず，データ予約を行わない場合の CMVTO と AMVTO を比較する．図3から，トランザクション数が比較的少ない場合には，AMVTO が有利であることがわかる．これは，この条件では後退復帰の発生回数自体が少ないので，カスケーディングアボートはほとんど問題とならず，逆に遅い更新によるトランザクションの遅延が影響を与えていると思われる．一方，トランザクションの増加に伴って後退復帰が頻発し，いずれの方式も性能の低下が見られる．特に AMVTO

表 2 トランザクションの平均後退復帰回数
Table 2 Average number of rollback transactions.

| トランザクション数 | 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| AMVTO （予約なし） | 0.2 | 2.3 | 5.8 | 12.0 | 77.3 |
| CMVTO （予約なし） | 0.6 | 5.7 | 4.9 | 8.6 | 34.8 |
| AMVTO （予約あり） | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 |
| CMVTO （予約あり） | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 |

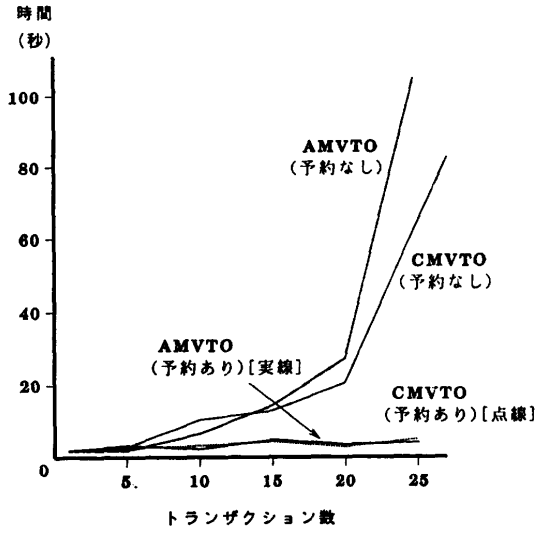



図 3 トランザクションの平均実行時間
Fig. 3 Average time of transaction execution.

では，カスケーディングアボートによって極端に性能が低下し，この状態では実用に耐えることはできない．

次に，データ予約を行った場合について述べる．この場合，AMVTO と CMVTO のいずれについても，後退復帰は発生していない．これはトランザクション数が増加した場合も同様である．後退復帰のオーバヘッドは，非常に大きいものとなるので，若干の遅延があったとしてもデータ予約の効果は大きいと言える．また，表 3 に最初のトランザクション開始から最後のトランザクションが終了するまでの全実行時間を示す．この表から，データ予約を行っている場合には，全実行時間も短縮されていることがわかる．すなわち，データ予約による後退復帰の減少は，トランザクションが逐次的に実行されている状態ではなく，並列に実行されている状態で達成されている．以上のように，トランザクションの多重度が大きい場合には，データ予約が非常に有効であることが分かる．

表 3 トランザクションの全実行時間（単位は秒）
Table 3 Whole execution time of transactions (Computed times are in sec.).

| トランザクション数 | 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| AMVTO （予約なし） | 6.86 | 17.71 | 34.54 | 69.30 | 437.02 |
| CMVTO （予約なし） | 7.61 | 26.25 | 34.66 | 52.33 | 173.07 |
| AMVTO （予約あり） | 7.46 | 11.21 | 19.73 | 23.86 | 31.39 |
| CMVTO （予約あり） | 7.60 | 14.25 | 20.80 | 24.59 | 32.80 |

### 5.2 長時間トランザクションに関する評価

次に，長時間トランザクションを含む複数トランザクションに対する制御の評価を行う．長時間トランザクションは，無作為に選んだデータに対し，15 回の参照と 5 回の更新を繰り返し 20 回実行するものとした．また，その他に参照のみ 20 回行うトランザクションを 9 個用意した．まず，長時間トランザクションを発行してから，約 1 秒間隔で他のトランザクションを順次発行する．トランザクション内部の遅延は前節までと同様であり，実行時間は長時間トランザクションが 40 秒，他のトランザクションが 2 秒である．実験は，長時間トランザクションに CMVTO，予約操作付き CMVTO，排他ロック式時刻印方式の各方式を適用した場合について（その他のトランザクションには CMVTO を適用），それぞれのトランザクションの実行時間を測定した．表 4 に測定結果を示す．表中のトランザクション #1 が長時間トランザクション，#2 から #10 が通常のトランザクションである．長時間トランザクションを排他ロック式時刻印方式で制御することにより，トランザクションの後退復帰および実行時間のいずれについても，参照トランザクションに対する影響をほとんど抑えられることが分かる．長時間トランザクションの更新操作と他のトランザクションの更新操作を並列実行することは困難であるが，長時間トランザクションと参照トランザクションを同時に実行する方法としては，この排他ロック式時刻印方式が非常に効果的な方式であると考えら

表 4 長時間トランザクションの制御方式の比較（単位は秒）
Table 4 Comparison between control algorithms for a long-lived transaction (Computed times are in sec.).

| 番　　号 | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---|---|---|---|---|---|---|---|---|---|---|
| CMVTO (予約なし) | 43.98 | 56.00 (5) | 54.88 (5) | 54.88 (6) | 55.17 (6) | 48.11 (3) | 47.55 (3) | 42.56 (1) | 38.24 | 41.00 (1) |
| CMVTO (予約あり) | 43.76 | 45.87 | 44.52 | 44.29 | 43.50 | 42.55 | 41.57 | 40.66 | 39.82 | 38.97 |
| 排他ロック式時刻印方式 | 48.37 | 2.45 | 2.66 | 2.66 | 2.46 | 2.54 | 2.69 | 2.26 | 2.30 | 2.33 |

※ 下段（ ）内は後退復帰の発生回数

れる.

## 6. 結　　論

本論文では，適応型時刻印方式に基づく分散同時実行制御方式について述べた．適応型時刻印方式では，従来の多重版時刻印方式，データ予約を用いた予約型時刻印方式，排他ロック式時刻印方式の各方式を提供することができた．実験によって，以上の方式をうまく組み合わせることで単一の制御方式では対応できなかった問題点を解決できることが示された．また，その実現は OS の機能に依存しておらず，UNIX 上などの分散アプリケーションにも適用可能である．本論文で述べた同時実行制御方式は，トランザクションに対して柔軟な制御を行うことが可能であるが，その適用方法についてはさらに考察が必要である．

**謝辞**　日頃から貴重なご意見，ご討論いただいている國枝義敏助教授，岡部寿男助手を始めとする津田研究室の諸氏に感謝いたします．

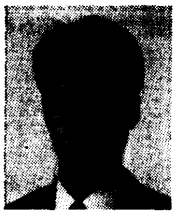
## 参 考 文 献

1) Gray, J. N.: Transaction Concept: Virtues and Limitations, *Proceedings of 7th International Conference on Very Large Data Bases*, pp. 145-154 (1981).
2) Bernstein, P. A. and Goodman, N.: Timestamp Based Algorithms for Concurrency Control in Distributed Database Systems, *Proceedings of 6th International Conference on Very Large Databases*, pp. 285-300 (1980).
3) Bernstein, P.A., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley (1987).
4) Cellary, W., Gelenbe, E. and Morzy, T.: *Concurrency Control in Distributed Database Systems*, North-Holland (1988).
5) Reed, D. P.: Implementing Atomic Actions on Decentralized Data, *ACM Trans. Comput. Syst.*, Vol. 1, No. 1, pp. 3-23 (1983).
6) 國枝和雄，大久保英嗣，津田孝夫：データベースオペレーティングシステム *μOPT-R* における分散セグメンテーション方式，情報処理学会論文誌，Vol. 32, No. 4, pp. 470-480 (1991).

(平成3年7月24日受付)
(平成4年4月9日採録)

**國枝　和雄**（正会員）

昭和39年生．昭和62年京都大学工学部情報工学科卒業．平成元年同大学院修士課程修了．平成4年同大学院博士後期課程単位修得認定退学．同年日本電気(株)に入社，現在に至る．オペレーティングシステム，データベースシステム，分散処理等に興味を持つ．日本ソフトウェア科学会会員．

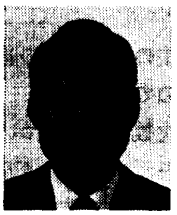


**畑田　孝幸**

1967年生．1991年京都大学工学部情報工学科卒業．同年(株)ワコムに入社，現在に至る．在学中，オペレーティングシステムの研究に従事．

**大久保英嗣**（正会員）

1951年生．1974年北海道大学理学部数学科卒業．1977年同大学工学部情報工学科大学院修士課程修了．同年(株)日立製作所ソフトウェア工場に入所．主として FORTRAN コンパイラの開発に従事．1979年より京都大学工学部情報工学科助手．1985年同講師，1987年同助教授，1991年立命館大学理工学部情報工学科教授，現在に至る．工学博士．オペレーティングシステム，データベースシステム等の研究に従事．日本ソフトウェア科学会，システム制御情報学会各会員．



**津田　孝夫**（正会員）

1957年3月，京都大学工学部電気工学科卒業．現在京都大学工学部情報工学科教授．計算機ソフトウェア講座担当．工学博士．自動ベクトル化/並列化コンパイラ，スーパコンピューティング，オペレーティングシステムの研究に従事．「モンテカルロ法とシミュレーション」(培風館)，「現代オペレーティングシステムの基礎」(オーム社，共著)，「数値処理プログラミング」(岩波書店)などの著書がある．昭和63年度情報処理学会論文賞受賞．前本学会関西支部長．ACM, SIAM 各会員．IFIP/WC 2.5 (Numerical Software) 委員．