Vol. 27 No. 2 情報処理学会論文誌 Feb. 1986

データベースのロック方式における並列性向上の一方式†

小林哲二††

ロック方式は，並列に処理を行う複数のトランザクションがデータベースにアクセスする際に，処理の無矛盾性を保つ手法として，広く用いられている．しかしながら，ロックは，多くの場合，トランザクションの並列性を減少させる．その要因の一つは，ロールバックの連鎖（障害等のために，多数の処理済みのトランザクションの処理の無効化が必要となる現象）を防ぐために，任意のトランザクションについて，データ更新のためのロックの解除は，そのトランザクションの最後の動作として行わねばならないことである．本論文では，この問題を解決するために，多バージョンを用いた新しいロック方式を提案する．このロック方式では，各トランザクションは任意のエンティティ（データに対するロックの単位）に対して，個別に更新した値を保持することにより，トランザクションが任意のエンティティに対するロックの解除の要求を行った時点から，その更新済みの値を，ロールバックの連鎖の危険なしに，他のトランザクションが参照・更新できるようにすることを可能とする．このロック方式の効果を評価し，また，分散システムへの適用性を示す．

1. まえがき

データベースの無矛盾性を保証するための技術として，同時実行制御がある．同時実行制御とは，共有データへの同時アクセス要求を制御することである．同時実行制御方式としては，ロック方式，競合グラフ解析方式，競合発生時再開方式，多数決方式等が主要な方式として，実現または提案されている[3),9)]．現在の大部分のデータベースシステムで，同時実行制御に使用されている方式はロック方式[4),5),8)]である．

この論文では，多バージョン[8),10)]を使用するデータベースシステムにおけるロック方式について，トランザクション（データベースへの処理の単位）の処理の並列性の減少を，通常のロック方式[4)]よりも改善することのできるロック方式（部分アンロック方式と呼ぶ）を提案する．多バージョンによってロック方式の並列性を改善する研究としては，Bayer らの提案[1),2)]（B法と呼ぶ）がある．B法では，多バージョンの実現方式として，System R のデータベースで障害回復のために使用されている影（shadow）空間[6)]を用いており，データベースを更新する場合には，新値，旧値という二つの値を一つのエンティティ（データに対するロックの単位）に対して保持できる．ロックモード（ロックの種類）としては，データの参照用にリードロック，データの参照・更新用に解析ロックおよびコミットロック，という三つのロックモードを設けており，各モード別にトランザクションによる新値と旧値へのアクセスを，矛盾が発生しないように制御する．矛盾の発生の検出は，複数のトランザクションによるエンティティへのアクセスの状態を表す従属関係グラフの作成により動的に行う．B法の長所は，リードロックが常に許可されることである．しかしながら，B法においても，トランザクションのロールバックの連鎖[2),5),8)]（障害等のために，一つのトランザクションの処理に関連した多数のトランザクションのロールバック（トランザクションの無効化）が必要となる現象）を防止するためには，通常のロック方式と同様に，狭義二相ロック法[2),8)]（一つのトランザクションが行ったすべてのライトロック（データの参照・更新用のロック）の解除は，そのトランザクションの最後に行う方法）を使用する必要があるので，解析ロックからコミットロックへの遷移は，一つのトランザクションが行ったすべての解析ロックについて，そのトランザクションの最後に行う必要がある．

本論文では，多バージョンを有効利用することにより，狭義二相ロック法を使用せずに，ロールバックの連鎖を防止できるロック方式を提案する．このロック方式により，トランザクションの更新後の値を，早期に他トランザクションに参照させることができ，他トランザクションが新たな更新をその値に対して行うこともできる．また，本論文のロック方式を分散システムに適用することも可能なことを示す．なお，B法によるリードロックを常に許可する機能は，任意数のバージョンを有するシステムにも拡張可能であることが Bayer ら[2)]によりすでに指摘されているので，本論文のロック方式をB法と合わせて使用することも可能

† A Locking Method for Increasing Concurrency in Database Systems by TETSUJI KOBAYASHI (NTT Electrical Communications Laboratories).

†† NTT 電気通信研究所

である.

2. 提案するロック方式

2.1 並行処理の問題点

複数のトランザクションの並行処理に際してロック方式を用いる場合，ロック方式は，トランザクションの直列可能性，デッドロック，およびロールバックの連鎖の問題を解決することが，トランザクションの無矛盾性の保持のために必要十分な条件である[8]．以下では，これらの問題について定義を行う．

［定義1］ トランザクション $T_1, T_2, \cdots, T_n, (n \geq 2)$ の一つの並列実行による実行結果* がそれらのトランザクションがアクセスするデータベースのあらゆる初期状態について，ある一つの直列な実行順序のトランザクション $(T_{p1}, T_{p2}, \cdots, T_{pn})$ の実行結果と等しいのであれば**，その並列実行は直列可能性を有すると言う[2)-4)]．ここで，$(p1, p2, \cdots, pn)$ は，$(1, 2, \cdots, n)$ のすべての置換のうちの一つを表す．

［定義2］ トランザクション $T_1, T_2, \cdots, T_m, (m \geq 2)$ について，T_1 が T_2 の占有資源（エンティティが資源である）の空きを待ち，T_2 と $T_3, \cdots, T_m$ と T_1 も同様な状態にあるとき，$T_1, T_2, \cdots, T_m$ はデッドロック状態にあると言う[2),7)]．ここで，トランザクションの順序は任意である．

［定義3］ トランザクション $T_1, T_2, \cdots, T_n$ の並列実行中に，任意のトランザクション T_i, $(i=1, 2, \cdots, n)$ の棄却をしたときに，T_i 以外のコミット（エンティティの値の実更新，およびトランザクションの再開点の設定）済みのトランザクションの実行結果に影響が生じるのであれば，$T_1, T_2, \cdots, T_n$ の並列実行にはロールバックの連鎖が生じると言う[2),5),8)]．

なお，直列可能性の実現のためには，次の定理を使用する．

［定理1］ 複数のトランザクションの並列実行時に直列可能性が成立するための十分条件は，個々のトランザクションについて，ロックとアンロック（ロックの解除）が二相ロック性（一つのアンロック後は，新たなロックを行わないこと）を有することである[2),4)]（証明略）．

2.2 多バージョンの実現方式

この論文で提案するロック方式では，多バージョンの実現方式として，仮定間と呼ぶ空間を使用する．データベースへのアクセス時には，実空間（実際にデータが格納されている二次記憶上のデータベース空間）の更新は行わず，仮空間（各トランザクションが各エンティティに対して所有することができ，動的に割付けられる二次記憶上の空間）に更新した値を保持し，仮空間の管理情報は主記憶上にのみ保持する．仮空間は，データベース管理プログラム（DBM；Database Manager）にしか見えない．実更新（実空間の更新）は，トランザクションの終了直前に行う．この方法により，仮空間の更新状態における障害発生時は，そのエンティティの媒体障害（二次記憶媒体に格納されているデータの，物理的または論理的な障害）を除き，ただちに直前の再開点からトランザクションを再開始できる．ただし，媒体障害時は，更新の履歴情報によるデータベースの復元処理が必要である（なお，仮空間を，ロック方式における並列性向上のためにのみ使用してもよい）．**図1**に仮空間と実空間の概念を示す．

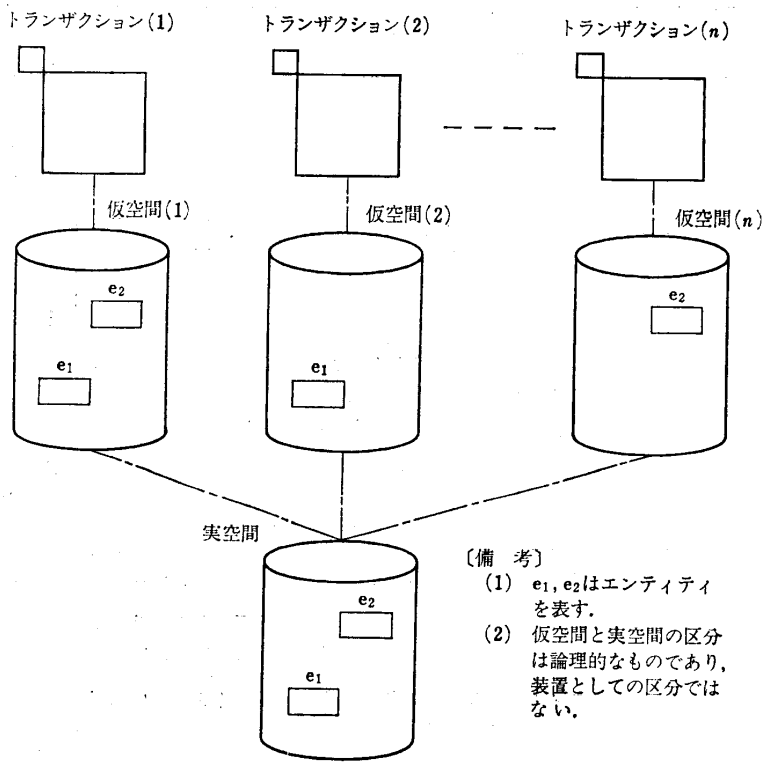

図 1 仮空間と実空間の概念
Fig. 1 The concept of the virtual space and the real space.

* 実行結果とは，データベースの最終状態および処理の出力結果を意味する．
** 一つの並列実行と等しい実行結果を有する直列な実行順序は，二つ以上存在してもよい．

任意のエンティティ e_k について，R_k を実空間の値，$V_k(i)$ をトランザクション T_i，$(i=1,2,\cdots,n)$ の仮空間の値（空集合の場合もある）とすると，e_k の値の状態は，次のように表すことができる．

$e_k=\{R_k, V_k(1), V_k(2), \cdots, V_k(n)\}$

なお，B法で使用している影空間による方式は，$V_k(i)$ の値が，すべての T_i，$(i=1,2,\cdots,n)$ に対して，一つのみしか許されない場合に相当する．

2.3 部分アンロック方式

本節では，多バージョンを使用して，トランザクションの並列性を改善したロック方式（部分アンロック方式）の状態遷移，制御方式等を述べる．

（1） ロックに関する状態

ロックモードとして，リードロック（データのリード（参照）用）とライトロック（データのライト（更新）とリード用）を用いる．各トランザクションは，いずれか一方のロックモードを使用しても，両方のロックモードを使用してもよい．各トランザクションは，アクセスしようとするエンティティに対して，ロック（リードロックまたはライトロック）を要求することができるが，自トランザクションですでにロックしているエンティティに対して，再度ロック要求を行うことはないものとする．

リードロックまたはライトロックの解除を要求するのは，アンロック要求である．また，DBM のみに適用する状態として，“部分アンロック状態”を導入する．したがって，任意のトランザクション T_i について，任意のエンティティ e_k に対するロックに関連する状態としては，リードロック状態 $S(T_i, e_k, L_r)$，ライトロック状態 $S(T_i, e_k, L_w)$，リードロックに対する部分アンロック状態 $S(T_i, e_k, P_r)$，ライトロックに対する部分アンロック状態 $S(T_i, e_k, P_w)$，および完全にアンロックされた状態である，アンロック状態 $S(T_i, e_k, U)$ がある．リードロック状態またはライトロック状態を，ロック状態と呼び，リードロックまたはライトロックに対する部分アンロック状態を，部分アンロック状態と呼ぶ．

（2） 状態遷移，ロック制御，およびコミット

表1に，プロセス T_i のエンティティ e_k についての状態遷移を，アクセス制御との関係とともに示す．

表1 プロセス T_i のエンティティ e_k についての状態遷移
Table 1 State transitions of a process T_i for an entity e_k.

T_i の状態	T_i の e_k へのアクセス制御		次の状態遷移
$S(T_i, e_k, U)$	T_i は，e_k にアクセス不可．		T_i の e_k へのリードロックが許可されたときに，$S(T_i, e_k, L_r)$ に遷移する．
			T_i の e_k へのライトロックが許可されたときに，$S(T_i, e_k, L_w)$ に遷移する．
$S(T_i, e_k, L_r)$	[e_k が，T_i に先行するすべてのプロセスについて，部分アンロック状態ではないとき] 実空間 R_k にアクセスする．		T_i が e_k にアンロック要求を行ったときに，$S(T_i, e_k, U)$ に遷移する．（T_i が e_k にリード動作を行うことなく e_k にアンロック要求を行ったときも同様である．）
	[e_k が，T_i に先行する一つ以上のプロセスについて，部分アンロック状態であるとき] 最新の仮空間を有するプロセス T_j の仮空間 $V_k(j)$ にアクセスする．		T_i が e_k にアンロック要求を行ったときに，$S(T_i, e_k, P_r)$ に遷移する．
$S(T_i, e_k, P_r)$	T_i は，e_k にアクセス不可．		T_j が，$S(T_j, e_k, U)$ となるときに $S(T_i, e_k, U)$ に遷移する．
$S(T_i, e_k, L_w)$	リード動作	[T_i が e_k にライト動作を行う前のリード動作] $S(T_i, e_k, L_r)$ のときと同じである．	T_i が e_k にアンロック要求を行ったときに，$S(T_i, e_k, P_w)$ に遷移する．（ただし，T_i がライト動作を行うことなく e_k にアンロック要求を行ったときには，次の状態遷移は，$S(T_i, e_k, L_r)$ のときと同じとする．）
		[T_i が e_k にライト動作を行った後のリード動作] T_i の仮空間 $V_k(i)$ にアクセスする．	
	ライト動作	[T_i のライト動作] T_i の仮空間 $V_k(i)$ にアクセスする．	
$S(T_i, e_k, P_w)$	T_i は，e_k にアクセス不可．		T_i についての e_k のアンロック完了時に，$S(T_i, e_k, U)$ に遷移する．

部分アンロック状態となった仮空間は，各エンティティごとにFIFO (First In First Out) 方式の待ち行列を形成し，コミット待ちとなる．

任意のトランザクション T_i が，同じエンティティの異なる値を参照するのは，T_i 自身で更新した場合のみとする．したがって，任意のトランザクション T_j の仮空間の値は，T_j のコミット完了後もそれにアクセスした T_j 以外のトランザクションのリードロックまたはライトロックがアンロック要求されるまでは，保持される必要がある．一つのエンティティ e_k についての複数のトランザクションによる実更新は，ライトロックが許可されてからアンロック要求が出されるまでの期間 ($S(T_i, e_k, L_w)$ 状態) には，他のライトロック要求に対しての排他性があることによって，ライトロックが設定された順序で行われる．トランザクションのコミットは，次の二つの条件を満たすものを順に行う—①そのトランザクションのアクセスしたすべてのエンティティが，アンロック状態または部分アンロック状態となっている，②そのトランザクションの部分アンロック状態となっているエンティティは，すべてコミット待ちの待ち行列の先頭にある．一つのトランザクションのコミットは，そのトランザクションが更新したすべてのエンティティについて，同時に行う．完全なアンロックは，コミット後に行う．一つのエンティティに対して，複数のトランザクションによるロック要求が競合時のロックの両立性について，ロックの状態と要求されているロックモードの関係は，通常のロック方式と同じであり，**図2**に示す．なお，部分アンロック状態は，ロック競合の観点では，アンロック状態と同じである．

（3） アンロックのタイミング

ライトロックについて考える．個々のトランザクションは，二相ロック性を保持することとする．ロック要求は，各トランザクションで任意に行えばよいが，アンロックについては次の方式が考えられる．

［S1］ コミット前に任意にアンロックする．

［S2］ そのトランザクションで更新したすべてのエンティティについて，コミットの完了直後に一度にアンロックする（狭義二相ロック法である）．

［S3］ 個々のエンティティについて，アンロック要求時に部分アンロック状態となり，コミット完了後に一度にアンロック状態とする．

状態＼要求	リードロック	ライトロック
リードロック	可	不可
ライトロック	不可	不可

図2 ロックの両立性

［備考］ 不可のときには，要求側は，ロック可能となるまで待つこととする．

Fig. 2 The locking compatibility.

S1 方式は，ロールバックの連鎖の危険がある．S2 方式は，通常のロック方式で用いられている方式であり，ロールバックの連鎖は発生しないが，ロック時間が長いという欠点がある．S3 方式は，本論文の方式であり，任意の時点で各エンティティごとに部分アンロック状態に遷移できるので，S1 方式と同じ性能を実現でき，ロールバックの連鎖も防止できる．

2.4 分散システムにおけるロック制御

前節までに述べたロック方式を，計算機網に分散したデータベースの更新に適用する方法を述べる．

（1） 分散システムのモデル

分散システムのモデルを次のように設定する．分散システムは K 個のノードから構成されており，各ノードに任意のデータベースが存在する．各ノードからは，DBM 間の通信によって，他の任意のノードにアクセスできる．一つのトランザクションは，一つ以上のサブトランザクションから構成されている．一つのトランザクションに属するサブトランザクションは，1ノードに一つとする．また，一つのノードのデータベースには，そのノードの DBM のみが直接にアクセスできるものとする．

（2） 二相コミットプロトコル

分散データベースにおける同期更新方式として，二相コミットプロトコルが知られている[5),9)]．これは，トランザクションに二つのフェーズを設け，第一フェーズは，更新履歴情報の保存によって，各サブトランザクションがコミットと棄却のうちのいずれでも可能な状態とし，第二フェーズで全サブトランザクションを同期してコミットを行う方式である．この論文の方式においても二相コミットプロトコルを使用するものとし，第一フェーズはデータベースの利用者の処理の終了である仮コミット，第二フェーズは実空間へのデータ更新および再開点の設定である実コミットを行うことにより実現する．

（3） アンロックのタイミング

二相コミットプロトコルとロック/アンロックの関係について考える．個々のトランザクションは，二相ロック性を保持することを仮定する（したがって，一つのトランザクション下の最初のアンロック要求を行

うサブトランザクションは，そのトランザクション下の他のすべてのサブトランザクションもアンロック要求を行うことのできる状態にあることを確認後に，最初のアンロック要求を行う必要がある)．アンロックを完了するタイミングには，次の方式がある．

[D1] 各サブトランザクションが，実コミット終了直後にアンロックを完了する．

[D2] すべてのトランザクションについて，一つのトランザクションに属しているすべてのサブトランザクションの実コミットが終了後にアンロックを完了する（狭義二相ロック法である)．

[D3] この論文の部分アンロック方式．

D1 方式では，ロールバックの連鎖が発生する危険がある．D2 方式では，ロールバックの連鎖は発生しないが，ロック時間が長いという欠点がある．D3 方式では，D1 方式と同じ性能を実現でき，かつロールバックの連鎖も防止できる．

2.5 正 当 性

この論文で提案しているロック方式（部分アンロック方式）の正当性について，以下で証明する．

[定理2] 任意のトランザクションが，ライトロックのアンロック要求後でアンロック未完かつコミット前のエンティティ（部分アンロック状態）を，他のトランザクションに対して参照・更新することを許しても，直列可能性は保証できる．

(証明) 2.3 節から，部分アンロック状態では，他のトランザクションからのそのエンティティへのアクセスに対しては，完全にアンロックされた状態と同じエンティティの値を参照させている．したがって，トランザクションが二相ロック性を保持することにより，並列実行する複数のトランザクションについての直列可能性が成立する． (証明終)

[定理3] 部分アンロック方式を用いた場合であっても，デッドロックの発生の検出は，通常のロック方式と同じ方法[5),7)]を使用可能である．

(証明) トランザクションがエンティティにリードロックを設定すると，そのエンティティの実空間および仮空間は，リードロックを要求する他のトランザクションに対しては共有の扱いとなり，またそのエンティティに対してライトロック要求を行うトランザクションは，ライトロック待ちの状態となる．トランザクションがエンティティにライトロックを設定すると，そのエンティティの実空間および仮空間は，そのトランザクションに占有され，そのエンティティにリードロックまたはライトロックを要求するトランザクションは，待ち状態となる．したがって，エンティティを資源として扱う場合のデッドロックの解決法は，仮空間と実空間を意識する必要がないので，通常のロック方式の場合と同じ方法を使用できる． (証明終)

[定理4] 任意のトランザクション T_i が任意のエンティティ e_k の更新後に $S(T_i, e_k, P_w)$ の状態になったとする．このとき，他の任意のトランザクション T_j, $(i \neq j)$, が e_k にリードロックまたはライトロックを行って，e_k の参照または更新を行ったとしても，ロールバックの連鎖は発生しない．

(証明) T_i がコミット以前に棄却されるときには，2.3 節から，T_j, $(i \neq j)$, は e_k について，$S(T_j, e_k, L_r)$, $S(T_j, e_k, L_w)$, $S(T_j, e_k, P_r)$, $S(T_j, e_k, P_w)$ のうちのいずれか一つの状態になっているので，T_j の棄却も可能である．それゆえに，ロールバックの連鎖は発生しない． (証明終)

定理2～定理4により，本論文で提案する部分アンロック方式は，2.1 節で述べた並行処理の問題点を解決しており，正当なロック方式であることが証明された．また，2.4 節で説明したように，分散システムにおける各ノードのサブトランザクションから構成されている一つのトランザクション，および全ノードのデータベースの集合である一つのデータベースを考え，二相ロック性の保持およびコミットは，サブトランザクション間の同期を保って行うこととすれば，分散システムにおけるトランザクションについても，並行処理の問題点は一つのノードに閉じたトランザクションと同じであるので，定理2～定理4が成立する．

2.6 効 果

本論文で提案した部分アンロック方式を，通常のロック方式と，トランザクションの並列性の観点から比較する．部分アンロック状態の導入により，狭義二相ロック法を用いる通常のロック方式と比べて，ライトロックの排他的なロック時間が短縮され，更新済みの値をより早く，他トランザクションに参照・更新ならしめることができる．一つのトランザクションがライトロックを行っているエンティティ数の変化の例を，**図3** に示す．

ライトロックのみを行う一つのトランザクション T_j について考える．2.4 節で述べた分散システムのモデルを用いて，T_j は K 個のノードに散在する K 個のサブトランザクション $T_j(k)$, $(k=1, 2, \cdots, K)$, から構成されているものとする．T_j が一つのサブト

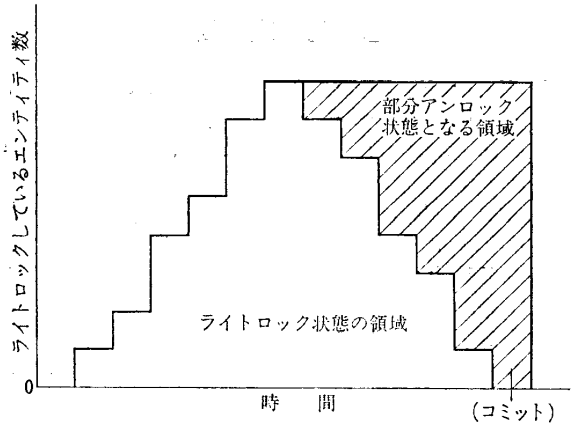

図 3 一つのトランザクションがライトロックしているエンティティ数の変化の例
［備考］ 通常のロック方式（狭義二相ロック法）では，部分アンロック状態もライトロック状態である．
Fig. 3 An example of the variation of the number of entities locked in the write-lock mode by a transaction.

ランザクションだけで構成されている場合を考えると，単一ノードの場合となる．処理の並列性に対する影響を考えるための評価基準として，T_j がノード k, $(k=1,2,\cdots,K)$, で m_k 個のエンティティをライトロックしているときに，T_j 以外のトランザクションからのそれらのエンティティへのリードロックまたはライトロックの要求が待ちとなる排他的な時間の和である，延べロック時間を用いる．A_1 を通常のロック方式（狭義二相ロック法を使用）を用いる場合の延べロック時間，A_2 を部分アンロック方式を用いる場合の延べロック時間とする．ここで，A_2 の延べロック時間として，部分アンロック状態は加算されない（この理由は，部分アンロック状態のエンティティに対しては，他のトランザクションは，それが完全にアンロックされている場合と同じ値に対して参照・更新を行うことができるからである）．すると，A_1, A_2 は次のようになる．

$$A_1=\sum_{i=1}^{K}\sum_{l_i=1}^{m_i}[t_{UL}(i,l_i)-t_L(i,l_i)] \quad (1)$$

$$A_2=\sum_{i=1}^{K}\sum_{l_i=1}^{m_i}[t_\alpha(i,l_i)-t_L(i,l_i)] \quad (2)$$

ここで，$t_{UL}(i,l_i)$ は，$T_j(i)$ のエンティティ $e(l_i)$ についてのアンロック完了時刻，$t_L(i,l_i)$ は，$T_j(i)$ の $e(l_i)$ についてのライトロック時刻，$t_\alpha(i,l_i)$ は，$T_j(i)$ の $e(l_i)$ についてのアンロック要求時刻である．$e(l_i)$ は，ノード i における l_i 番目のエンティティであり，$l_i=1,2,\cdots,m_i$ である．式(1)と式(2)から，$\Delta A=A_1-A_2$ は次のようになる．

$$\Delta A=\sum_{i=1}^{K}\sum_{l_i=1}^{m_i}[t_{UL}(i,l_i)-t_\alpha(i,l_i)]$$

したがって，部分アンロック方式は，通常のロック方式と比較して，$t_{UL}(i,l_i)$ と $t_\alpha(i,l_i)$ の差が大きい場合に有利となることが分かる．アンロックの完了はコミットの直後であるので，$t_{UL}(i,l_i)$, $(i=1,2,\cdots,K;\ l_i=1,2,\cdots,m_i)$ の分散は小さい．また $t_{UL}(i,l_i)$ は $t_\alpha(i,l_i)$ の最大値よりも必ず大きいので，$t_\alpha(i,l_i)$ の分散が大きいときには部分アンロック方式の効果は大きくなる．**図4**に，ロックに関連したサブトランザクション間の通信の例を示す．

2.7 オーバヘッド

部分アンロック方式では，仮空間の追加が必要であり，このためのオーバヘッドを，入出力 (I/O) 回数と二次記憶空間量について考察する．

g をトランザクションの実行中におけるデータベー

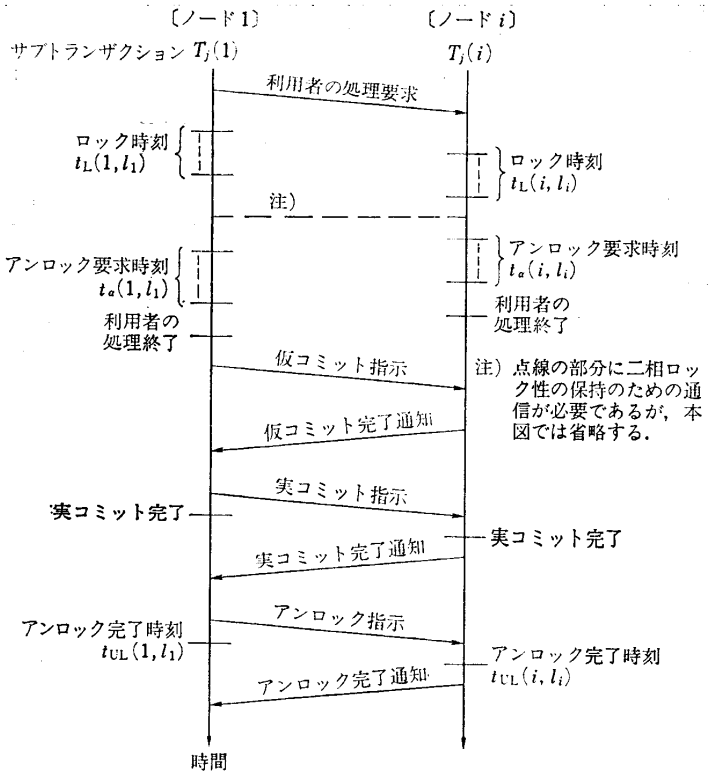

図 4 分散システムにおけるロック制御のためのサブトランザクション間の通信の例
Fig. 4 An example of the communications between subtransactions for locking in a distributed system.

ス更新のための I/O 要求回数，h を g のうちの再 I/O 要求回数として，I/O は仮空間から実空間への転写のために，$2(g-h)$ 増加する．また，u を1トランザクションの平均更新空間量，m を u の変動に対する余裕の空間量，l を瞬時最大トランザクション数とすると，前もって用意すべき仮空間用の空間量は $(u+m)l$ である．ただし，トランザクションが新規に作成を要求するエンティティについては，トランザクションの途中で障害が発生時には廃棄できるので，実空間を仮空間の扱いで使用できるため，空間量と I/O 回数の増加はない．また，多くの場合において，トランザクションの実行中に更新されるエンティティ数は，エンティティの総数よりも，十分に小さい．したがって，仮空間の追加による I/O 回数および二次記憶空間量のオーバヘッドは，実用的に十分に小さくすることが可能である．

3. むすび

データベースの更新における多バージョンの実現方式として，仮空間を用いる場合について，ロールバックの連鎖への対策を考慮して，通常のロック方式よりもトランザクションの並列性を高めることのできる方式を提案し，その効果，および分散システムへの適用法を示した．この方式は，多バージョンを用いるシステムには一般的に使用できる方式である．今後の課題としては，ロックを用いない方式との比較検討などがある．

謝辞 有益なご意見をいただいた横須賀電気通信研究所データ通信研究部の新井克彦部長，苗村憲司統括調査役，西川清史応用プログラム研究室長に感謝します．

参考文献

1) Bayer, R., Elhard, K., Heller, H. and Reiser, A.: Distributed Concurrency Control in Database Systems, Proc. VLDB, pp. 275-284 (1980).
2) Bayer, R., Heller, H. and Reiser, A.: Parallelism and Recovery in Database Systems, ACM Trans. Database Syst., pp. 139-156 (1980).
3) Bernstein, P. A. and Goodman, N.: Approaches to Concurrency Control in Distributed Data Base Systems, Proc. NCC, pp. 813-820 (1979).
4) Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L.: The Notions of Consistency and Predicate Locks in a Database System, *Comm. ACM*, Vol. 19, No. 11, pp. 624-633 (1976).
5) Gray, J. N.: *Database Operating Systems, Lecture Notes in Computer Science 60* (*Operating Systems*), pp. 393-481, Springer-Verlag, Berlin (1978).
6) Gray, J., McJones, P. et al.: The Recovery Manager of the System R Database Manager, *Comput. Surv.*, Vol. 13, No. 2, pp. 223-242 (1981).
7) Isloor, S. S. and Marsland, T. A.: The Deadlock Problem: An Overview, *Computer*, Vol. 13, No. 9, pp. 58-78 (1980).
8) 上林：共有データベースの諸問題に対する理論，情報処理，Vol. 24, No. 8, pp. 1020-1037 (1983).
9) Kohler, H. W.: A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems, *Comput. Surv.*, Vol. 13, No. 2, pp. 150-183 (1981).
10) Papadimitriou, C. H. and Kanellakis, P. C.: Concurrency Control by Multiple Versions, ACM Trans. on Database Syst., pp. 89-99 (1984).

(昭和59年8月29日受付)
(昭和60年10月17日採録)