分散データベース—更新トランザクションの実行と制御に関する記号演算†

山崎 晴明††

分散データベースシステムにおいて，データの一貫性保存の問題は，最も重要な課題のひとつである．本稿では，この一貫性を保つための方法として，従来提案されてきた方式とは，全く異なる観点から導かれたひとつの手法を提案する．それは，従来のものが，リソースであるデータに対しロックをかける方式であったのに比し，本方式は，トランザクションが他のトランザクションに対しロックをかける方式であるといえる．

本稿では，入力される更新トランザクションの形態が，あらかじめ定められているいわゆる定型業務処理を扱いの対象とし，まず入力されるトランザクションの解析のための手法を論じ，ついでトランザクションの集合が同時に実行される際のそれらの制御形式について述べる．このために，まずトランザクション集合に対するひとつのシンボリックな記述が導入され，トランザクションの実行および終了は，その記述に対する演算として表現される．このようにして，分散データベースにおけるトランザクション群の実行は，ある簡単な代数的構造を持つ演算の逐次実行という形で表現でき，それに基づく制御が行われることになる．さらに，本方式では，デッドロックは起り得ないことおよびデーター貫性が常に保持されることの形式的証明が述べられる．なお本方式がどのように用いられるかを明確に示すため具体例による説明も含まれる．

1. まえがき

近年オフィスオートメーションを中心とするさまざまな分野において，分散型データベースというアプローチが注目され始めている．この方式は，従来の集中型データベースに比して数多くの利点を持つものであるが，このようなシステムを構築するに当っては解決すべき技術的問題も数多く[1]，特にデータのコンシステンシィ（一貫性）をいかに維持するかは，このときの主要な技術課題のひとつとなっている[1)~4),6)]．

本稿の目的は，このデータコンシステンシィを分散環境下で維持するという問題に対し，ひとつの解を示そうとするものである．一般に，多くのデータベース処理においては，そのアップデートの形態があらかじめ判明しているいわゆる定型業務が扱いの対象となることが多い (e.g. バンキング，座席予約 etc.)．このような業務では，入力されるトランザクション間の競合の形態を前もって解析しておき，トランザクション実行時の処理の並列性を向上させることが可能となる．この方式は，従来コンシステンシィ維持のために提案されていた多くの方式がリソースであるデータにロックをかける形態であったのに対し，いわばトランザクションにロックをかけるという意味で全く観点の異なるアプローチといえる．

これまでに，入力されるトランザクションの性質を前もって解析しておく（プレアナリシス）という手法は，CCA (Computer Corporation of America) の SDD-1 で提案されてはいたが[1),2)]，後述するリード／ライトコンシステンシィ，順序づけコンシステンシィという2つの問題の違いを明確に区別していないため，解析の手法が複雑であり，さらにタイムスタンプを用いた更新処理をベースとしているものであった．ここで提案する方式は，簡単な代数演算を逐次実施してゆくのみのより簡潔なアルゴリズムを用いるもので，更新処理には必ずしもタイムスタンプを必要とするものではなく，この意味でより一般化されている．なお，本稿では，障害生起時の扱いについては述べていない．この問題については，たとえば文献 4),7) を参照されたい．

2. トランザクション実行における問題点

データコンシステンシィを維持するためには，トランザクションの実行に際して次の2種の問題を考慮しなくてはならない：

(1) リード／ライトコンシステンシィ

たとえば，T_a をファイル R_1, R_2 を読み込んで更新データを作成し，それをファイル W_1, W_2 に書き込むことを実行するトランザクションであるとする（たと

† Distributed Database——A Symbolic operation for Execution and Control of Update Transactions by HARUAKI YAMAZAKI (Integrated Systems Division, Research Laboratory, OKI-electric Industry Co., Ltd.).

†† 沖電気工業(株)研究所複合システム研究部

えばこのとき W_2 が W_1 の重複コピーであることもあり得る)．このとき T_a の読み込み操作はコンシステントに行われなければならない．コンシステントでない読み込みとは，たとえば，T_b を R_1, R_2 に書き込みを行うトランザクションとしたとき，T_b が R_1 に書き込み，次に R_2 に書き込みを行う直前で T_a による R_1, R_2 の読み込みが行われたとすれば，T_a の読み込みはコンシステントでない．なぜならば，R_1 には T_b の実行結果が反映されているのに対し，R_2 には反映されておらず，したがって T_a が R_1, R_2 を読み込んで作成した更新データはコンシステントでないからである．同様のことが書き込み操作についてもいえる．たとえば，T_c をファイル W_1, W_2 に書き込みを行うトランザクションであるとしたとき，T_c が W_1 に書き込みを行った後，W_2 に書き込む直前で T_a による W_1, W_2 への書き込みが行われたとする．T_a 実行終了後 T_c による W_2 への書き込みが行われるから最終的には W_1 には T_a の実行結果が，W_2 には T_c の実行結果が反映してしまうため，W_1 と W_2 とではデータのコンシステンシィが保たれていない．これは書き込み時のコンシステンシィが失われる例である．明らかに，読み込み，あるいは書き込み実行の際，対象データに対し全く同時に操作を行うか，あるいは操作対象に対してロックをかけるかしてリード／ライト操作をアトミックなものとする機構があれば，このコンシステンシィは保証される．このような機構を設けるために種々の方式が提案されている．ひとつは，文献4),6)のように対象データにロックをかける方式であり，他のひとつは，データおよびトランザクションにタイムスタンプを用いる方式である[2),3)]．さらには，ネットワークの遅延によるリード／ライト操作の時間差を避けるため，Ethernet のようなブロードカストタイプのネットワークを利用するという方法も有り得る[5)]．

いずれの手法を用いるにせよこれらの方式の詳細を述べることが本稿の目的ではないため，以降ではこのコンシステントなリード／ライト機構は上述のいずれかの方式によって保証されていると仮定する．なお，分散データベースシステムで一般に論じられる冗長コピーの同期更新の問題は，このリード／ライトコンシステンシィを保証するという問題の一特殊例であることに注意されたい．

（2） 順序づけコンシステンシィ

トランザクション群が与えられたとき，各々のトランザクションの実行は前述のリード／ライトコンシステンシィを保っていても全体としてはコンシステントな実行とならないことがある．たとえば，T_a, T_b, T_c をトランザクションとし，T_a は F_1 を読み込んで F_2 へ書き込み，T_b は F_2 を読み込んで F_3 へ書き込み，T_c は F_3 を読み込んで F_1 への書き込みを行うものとする．この関係を**図1**に示す．

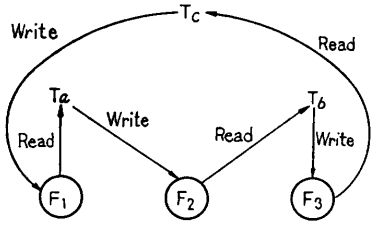

図1 3つのトランザクションの関係
Fig. 1 The relation-ships among three transactions.

各トランザクションの読み込みおよび書き込み対象はひとつであるから，どのような実行を行おうともリード／ライトコンシステンシィはこの場合保証される．しかしながら，たとえば3つのトランザクション T_a, T_b, T_c の読み込みを一斉に行い，その後 T_a, T_b, T_c の書き込みを一斉に行った場合，T_a は T_c によって書き込みの行われる前の F_1 を読み込んでいるため，T_a は T_c に先行したと考えられ，同様の理由で T_b は T_a に先行し，T_c は T_b に先行したと考えられる．したがって，T_a, T_b, T_c の間にどちらが先に実行されたかを決定する順序関係が定義できなくなってしまう．このようなトランザクション群の実行形態を順序づけコンシステンシィの失われた形態と呼び，本稿ではこの問題に焦点をあてる．

3. システムのモデルと記法

システムは，サイトと呼ばれるローカルなデータベースシステムとそれらを結合する通信チャネルより成るものとする．次に各サイトに分散配置されるデータの単位をフラグメントと呼び，必要に応じていくつかのサイトに重複して保持されるものとする．また，トランザクションは，それが参照するフラグメント集合（リードセット）と，それが更新するフラグメント集合（ライトセット）とを持つ．また，すべてのトランザクションの実行は，必要なデータを読み，それによって更新データを一時記憶域に作成するリードアクションと，その更新データを二次メモリ上に実際に書き込むライトアクションとから成る．さらに，前述のコンシステントなリード／ライト機構をシステムが持

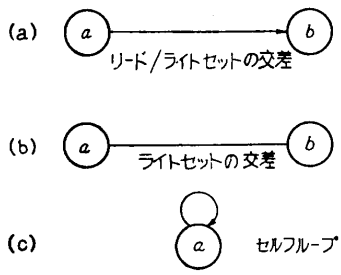

図 2　クラスの交差に関する記法
Fig. 2　Graphic notations for intersecting classes.

つことを仮定することにより，これらのアクションのアトミック性は保証されることになる．

また，同一のリードセットおよびライトセットを持つトランザクションの集合をトランザクションクラス（または単にクラス）と呼び，記号 $a, b\cdots$ のようにあらわす．次に，クラス a のリードセットとクラス b のライトセットとが空でない共通部分を持つとき，**図2**(a)のような方向付きエッジであらわし，これを a のリードエッジと呼ぶ．これは，クラス a および b に属するトランザクションが同時に実行されたとき，作り出される結果は，a を実行後 b を実行することによって得られる結果と一致するということを表わしている．また，クラス a と b のライトセットどうしが交差するときは，図2(b)のような方向を持たないエッジであらわし，これを a または b のライトエッジと呼ぶ．また a のリードセットと a のライトセットとが交差するとき図2(c)のようにあらわし，このエッジを a のセルフループと呼ぶ．

次に，システムが扱うすべてのクラスをノードとして，それらの間の交差関係をこれらのエッジによりあらわしたものをクラス競合グラフと呼ぶ．図4はクラス競合グラフの一例である．

4. トランザクションクラスの解析

以降では，簡単のため，セルフループを含まないクラス競合グラフを対象とする．

4.1 トランザクション同時実行記述

a をトランザクションクラスとしたとき，a に属する任意のトランザクションのライトアクションをひとつのクラスのように扱い a_W と記述する．次に与えられたクラス競合グラフ上のクラスおよびそのライトアクションをあらわす記号の重複を許さない任意の組合せをトランザクション実行記述（または単に記述）と呼ぶ．ただし，ひとつのクラスとそのライトアクションをあらわす記号とは同一記述中には出現しないものとし，さらにライトアクションのアトミック性から同一記述中にはライトエッジを共有する2つのライトアクションは現われないものとする．たとえばクラス競合グラフ図4上の記述は，$a, a_Wcb, cb, \cdots$ のような記号列である．

次に，ある記述 $S=a_1a_2\cdots a_n$ と，現在すでに同時実行を行っているトランザクション集合 T とが与えられたとき，すべての i について a_i であらわされるクラスまたはアクションが T に含まれていれば，S は真であるといい $T(S)=1$ であらわす．もし T に含まれない a_i がひとつでもあれば $T(S)=0$ であらわし，記述 S は偽であるという．次に同一クラス競合グラフ上の2つの記述 S, S' が与えられたとき $T(S)=1$ ならば $T(S')=1$ が常に成り立つとき S は S' をインプライすると呼ぶ．明らかに記述 a_W は，a をインプライするが，a は a_W をインプライはしない（a のリードアクション実行時）．さらに，$T(S)=1$ であって，S をインプライするどのような $S'(\neq S)$ についても，$T(S')=0$ となるようなトランザクション集合 T の実行のことを記述 S の実行と呼ぶ．

4.2 ラベル付グラフ

クラス競合グラフと記述 $S=a_1a_2\cdots a_n$ とが与えられたとき，すべての a_i について次の操作を行ったものを S によりラベル付けされた競合グラフと呼ぶ：

1) a_i がクラス記号であれば a_i のすべてのリードエッジにラベル付けする．

2) a_i がライトアクション記号であれば，a_i のライトエッジにラベル付けを行う．このとき，まだ向きの定義されていないライトエッジには a_i をソースとする向きを定義する．

このようにして，記述 S のラベル付けされた競合グラフが与えられたとき，S の任意の要素記号から出発して，S でラベル付けされた各エッジの向きに従い，S の各要素記号を1回ずつ通り，もとの要素記号に至る道が存在するとき，記述 S はループであるという．たとえば，図4のようなクラス競合グラフが与えられたとき，記述 ba_W はループである（ただし ba はループではないことに注意）．次に，コンシステントなリード/ライト機構を持つシステムのもとで，ある記述 S の実行を行ったとき，その実行結果が S の各要素記号を（どのような順序でもかまわないが）逐次実行したときの結果と常に一致するとき，記述 S は順序付けコンシステンシィを満たす記述と呼ぶ．なお，この概念

は良く知られたデータコンシステンシィに関する基準であるシリアライザブル[3]という概念と等価である．

このとき，次の命題が成り立つ．

命題 1 Sを与えられたクラス競合グラフG上の記述とする．SがG上のどのようなループもインプライしなければ，記述Sは順序付けコンシステンシィを満たす．

証明 Sは，どのようなループもインプライしないからSの各要素記号間にラベル付きエッジに従った半順序関係を定義することができる．クラス競合グラフの定義から明らかなように，この半順序に従ってトランザクションを逐次実行させたときの結果とSの同時実行結果とは一致するから，Sは順序付けコンシステンシィを満たす． *q. e. d*

なお，コンシステントなリード／ライト機構を持つシステムにおける順序付けコンシステンシィを満たす記述のことを同時実行可能な記述と呼ぶ．

4.3 ベーシックループ

クラス競合グラフGが与えられたとき，G上のループに対応する記述のうち他のループをインプライするループをすべて取り去った記述の集合をベーシックループ集合と呼び，その各々をベーシックループと呼ぶ．たとえば図4において，acbはベーシックループであるが，$awcwbw$は，acbをインプライするからベーシックループではない．

命題 2 記述Sが与えられたとき，Sがどのようなベーシックループもインプライしなければ，Sは同時実行可能な記述である．

証明 ベーシックループの定義から，任意のループはあるベーシックループをインプライする．したがって，Sはどのようなベーシックループもインプライしないからどのようなループもインプライしない．ゆえに命題1より，Sは同時実行可能な記述である．

q. e. d.

こうして，クラス競合グラフとその上のすべてのベーシックループを導くことでトランザクションクラスの解析は終了する．なお，後述するように，トランザクションの制御の目的は，このような同時実行不可能な記述の実行を阻止することにある．次に具体例により，このクラス解析の手順を示す．

4.4 クラス解析の例

次の例は，ある販売会社の注文処理の流れを簡略化して示したものである．この会社は**図3**のように，営業，購入，管理／検査の3部門から成っている．

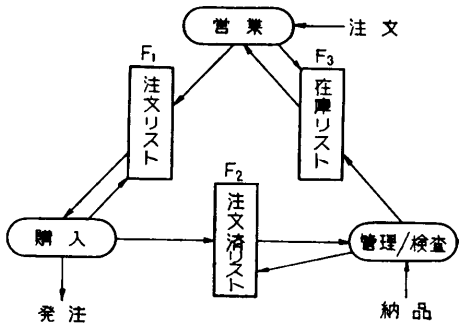

図 3 注文処理システム
Fig. 3 Order processing system.

（1） 販売業務

まず営業部門はユーザからの注文を受付け，注文品目が在庫リスト F_3 中にあれば，在庫から取り出し，ユーザに販売する．このとき，F_3 の内容から販売した品目は取り除かれる．もし注文品目が在庫リスト中になければ，それを注文リスト F_1 に付け加える．この品目は，後日在庫リスト F_3 に追加されて後，ユーザに販売されることになる．

（2） 発注業務

購入部門は，注文リスト F_1 に記載されている品目をメーカ側に発注する．発注の済んだ品目については，それを注文済リスト F_2 に付け加え，F_1 より除く．

（3） 納品，検査業務

管理／検査部門は，メーカ側より納品された品目の検査を行う．検査合格の品目については，それが注文済リスト F_2 に記載されていることを確認の上，注文済リスト F_2 からその品目を除く．さらにその品目を在庫リスト F_3 に追加する．

この3つの業務をそれぞれクラス a, b, c としたとき，そのリードセット，ライトセットは，**表1**のようになる（このとき品目の追加はライトのみの操作とな

表 1 各業務のリード／ライトセット
Table 1 Read/Write sets for each job.

トランザクションクラス名	リードセット	ライトセット	動作
a（販売業務）	F_3	F_3	F_3 から庫出品目を除き新しい F_3 とする（リードおよびライト）．
		F_1	注文品目を F_1 に追加（ライトのみ）．
b（発注業務）	F_1	F_1	F_1 から発注品目を除き新しい F_1 とする（リードおよびライト）．
		F_2	発注品目を F_2 に追加（ライトのみ）．
c（納品検査業務）	F_2	F_2	F_2 から検査合格品目を除き新しい F_2 とする（リードおよびライト）．
		F_3	合格品目を F_3 に追加（ライトのみ）．

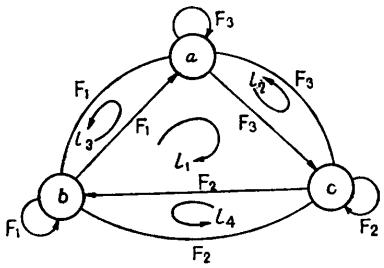

図 4 クラス競合グラフ
Fig. 4 Class conflict graph.

表 2 ベーシックループとその関連フラグメント
Table 2 Basic-loop and its related fragments.

ベーシックループ名	記　　述	関連フラグメント
l_1	acb	F_3, F_2, F_1
l_2	ac_W	F_3
l_3	ba_W	F_1
l_4	cb_W	F_2

ることに注意).

したがって，この例におけるクラス競合グラフは**図4**のようになり，また各ベーシックループとその関連フラグメントとの対応は**表2**で与えられる.

この例では，たとえばループ ba_W が実行されると次のような矛盾が起ることがわかる：F_1 の内容が A, B, C, F_2 の内容が空，F_3 の内容が X, Y であったとする．さらにユーザから営業部門が X, D なる品目の注文を受けたと仮定する．もし発注業務 T_b が F_1 を読むことと，販売業務 T_a による品目Dの F_1 への追加が同時に行われると，次に T_b のライトアクション実行後では F_1 は空となってしまい，品目Dの発注という業務が消失してしまう．したがって ba_W は実行されてはならないことがわかるが，どのようにして，このようなループの実行をシステムが阻止するかは後述する.

5. 実行ステータス

5.1 実行ステータスに定義される演算

あるベーシックループ S を $S=a_1a_2\cdots a_n$ とする．いまクラス a_i に属するトランザクションが m_i 個実行中であるとすれば，非負の整数を要素とする n 桁のベクトル $(m_1, m_2, \cdots m_n)$ で，S に属する各トランザクションの実行状態をあらわすことができる．この実行状態を表現するベクトルを<u>ロックステータス (Lst)</u> と呼ぶ．同様にして，実行終了したトランザクション数をベクトル $(m_1', m_2', \cdots, m_n')$ であらわすことができる．これを<u>リリースステータス (Rst)</u> と呼ぶ．さらにそのベクトル対 (Lst, Rst) をベーシックループ S に対する<u>実行ステータス</u>と呼ぶ．したがって，トランザクションの実行要求は Lst における該当桁への 1 の加算を，実行終了は Rst の該当桁に対する加算を意味することになる．さらに，Lst のすべての桁が非ゼロであったときその実行ステータスは<u>コンフリクティング</u>であると呼ぶ．またベーシックループの定義から，実行ステータスを巡回シフトしてもその意味はかわらない．ためにコンフリクティングでない実行ステータスはいつでも最下桁が 0 となるよう巡回シフトしておくものとする.

5.2 実行ステータスの変換

コンフリクティングでない実行ステータス：(Lst, Rst), $\mathrm{Lst}=(a_1, a_2, \cdots a_n)$, $\mathrm{Rst}=(b_1, b_2, \cdots b_n)$ が与えられたとする (ただし，$a_i \geq b_i$ for$\forall i$, $a_n=b_n=0$). このとき変換 φ：$\varphi((\mathrm{Lst}, \mathrm{Rst}))=(\mathrm{Lst}', \mathrm{Rst}')$ を次のように定義する：

$\mathrm{Lst}'=(a_1', a_2', \cdots a_n')$, $\mathrm{Rst}'=(b_1', b_2', \cdots b_n')$,

$$a_1'=a_1-b_1, \tag{1}$$

$$a_i'=\begin{cases} a_i-b_i & \text{if } a_{i-1}'=0 \\ a_i & \text{if } a_{i-1}' \neq 0, \end{cases} \tag{2}$$

$$b_1'=0 \tag{3}$$

$$b_i'=\begin{cases} 0 & \text{if } a_i \neq a_i' \\ b_i & \text{if } a_i=a_i' \end{cases} \tag{4}$$

たとえば，$\mathrm{Lst}=(4, 5, 3, 0, 1, 2, 0, 0, 1, 3, 1, 6, 0,)$,
$\mathrm{Rst}=(2, 2, 3, 0, 0, 1, 0, 0, 1, 3, 1, 5, 0)$ とすれば，
$\mathrm{Lst}'=(2, 5, 3, 0, 1, 2, 0, 0, 0, 0, 0, 1, 0)$,
$\mathrm{Rst}'=(0, 2, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)$ となる．変換 φ は，原則として Lst と Rst の対応桁ごとの減算により Lst′ の対応桁を生成するが，Lst′ の桁に 0 を生成したときのみ減算作用素が下位桁にもシフトし，そうでないときはもとの Lst の桁がそのまま Lst′ に引き継がれると考えると理解しやすい.

命題 3 実行ステータス $\alpha=(\mathrm{Lst}, \mathrm{Rst})$ とする.
$\varphi(\varphi(\alpha))=\varphi(\alpha)$ が成り立つ.

証明 $\varphi(\alpha)=((a_1', a_2', \cdots a_n'), (b_1', b_2', \cdots b_n'))$,
$\varphi(\varphi(\alpha))=((a_1'', a_2'', \cdots a_n''), (b_1'', b_2'', \cdots b_n''))$ とする.

(3)式より $b_1'=0$, (1)式より $a_1''=a_1'-b_1'$ が成り立つから，$a_1''=a_1'$ を得る．また(3)式より $b_1''=b_1'$ を得る．次に $i=k-1$ に対し；

$$\begin{cases} a_{k-1}''=a_{k-1}' \\ b_{k-1}''=b_{k-1}' \end{cases}$$ を仮定する.

もし $a_{k-1}''=a_{k-1}'=0$ であれば(2)式より，

$a_k''=a_k'-b_k'$，$a_k'=a_k-b_k$ を得る．$b_k'\neq 0$ であれば(4)式より，$a_k=a_k'$，$b_k'=b_k$ となる．

したがって，$a_k'-b_k'=a_k-b_k$ だから $a_k''=a_k'$．

一方，$a_{k-1}''=a_{k-1}'\neq 0$ であれば，(2)式より $a_k''=a_k'$ を得る．ゆえにいずれの場合も $a_k''=a_k'$ となる．

したがって，(4)式より，$b_k''=b_k'$ が成り立つ．

q. e. d.

なお，このφはトランザクションの実行終了通知があったとき，実行ステータスをリセットするのに用いられる．一般に，トランザクションの実行終了通知があったとき，Lst の該当桁から1を減算するだけでは順序づけコンシステンシィが保たれないことは，次の例からも明らかである；

今クラス競合グラフ図4におけるトランザクション T_a, T_c が同時実行されているとする ($Lst=(\overset{a\ c\ b}{1\ 1\ 0})$)．したがって，このときには，たとえ T_c の方が先に実行終了しても，T_a が T_c に先んじて実行されていると解釈される．次に，実際に T_c の実行終了通知がなされたとする．このとき，単純に，該当桁の減算により Lst を $Lst=(\overset{a\ c\ b}{1\ 0\ 0})$ に遷移させたとすれば，T_b の実行要求は受付可能となり（コンフリクティングとならないため），$Lst=(\overset{b\ a\ c}{1\ 1\ 0})$ となる．このときクラス競合グラフの定義から明らかなように，T_b は T_a に先んじて実行されていると解釈できる．しかしながら，すでに実行を終了した T_c は，明らかに T_b に先んじて実行されているから，T_a, T_b, T_c の間では順序づけコンシステンシィが成り立たなくなってしまう．つまり，T_a, T_c がともに実行中の状態において，次に T_b が実行可能となる唯一の条件は，T_a が実行終了することである．この関係を形式的に扱ったものが変換φである．

6. トランザクション実行時の制御

分散データベースシステムに伴う一般的問題点のひとつに，ネットワーク内の伝送遅延には，そのトポロジに依存したばらつきがあるため，制御情報等の複数サイトへの伝送においても，それの各サイトへの到着時間に，ばらつきが生じてしまうということがある (Ethernet のようなブロードカスト伝送向きのネットワークでは，この限りではない)．このため，トランザクション実行要求等の制御情報伝送時には，1対1通信を基本とする通常の通信プロトコルとは異なる（1対多通信を基本とする）プロトコルを導入する必要がある．本稿の方式では，以降に示すようにこのプロトコルは，多数決原理に基づくものとなっている．

6.1 トランザクション実行プロトコル

（1） リードアクション実行要求の送信

ユーザからトランザクション T_a の実行要求を受け付けたプロセスは，そのクラス（たとえば a）と，それを含むベーシックループとを判別し，関連フラグメントを持つすべてのサイトに(自身のサイトも含めて)リードリクエスト (REQ RD) を送信する．これを受信したサイトでは，Lst の a に対応する桁に1を加算し，その結果コンフリクティングとならなければACKを，そうでなければ NACK を返送する．

（2） リードアクションの実行

もしプロセスが過半数からの ACK を得られなかったようなベーシックループがひとつでもあった場合，ACK を応答したサイト（もしあれば）に対し ABORT を送信し，T_a の実行を放棄する．なお ABORT 受信サイトは，Lst の該当桁から1を減ずる．

一方すべての関連ベーシックループに対し過半数の ACK を受信したプロセスは，NACK を応答したサイト（もしあれば）に，REQ RD を送信し続ける．すべてから ACK が得られたとき，プロセスは T_a のリードアクションを実行する．

（3） ライトアクションの実行

読み込みを終了し，更新データを作成したプロセスは，次に a_W を含むベーシックループを判別し，関連フラグメントを持つすべてのサイトに REQ WT を送る．その後の処理は，(2)と同様であるが，ライトアクション実行要求時に ABORT された T_a は，実行中止とせず，一定の時間間隔をおいて再度実行が試みられる．

（4） 終了コマンドの送出

コンシステントなライト機構により更新を終えたプロセスは，COMPLETE を，リード／ライトアクションに関与したすべてのサイトに送信する．これを受信したサイトでは Rst の該当桁に1を加え，変換φを実行する．

こうして，このプロトコルにより，すべてのベーシックループの実行は阻止され，順序づけコンシステンシィは保たれることになる．

6.2 トランザクション実行とデッドロック

本節では，ステップ(3)でともにペンディングとなった2つのライトアクション a_W と b_W とがデッドロ

表 3　実行ステータスの遷移

Table 3　Transitions of execution status

保持サイト	ベーシックループ ＼ 時点		t_0	t_1	t_2	t_2'	t_3	t_4	t_5
S_1 S_2 S_3	acb		$a\ c\ b$	$a\ c\ b$	$b\ a\ c$	$b\ a\ c$	$b\ a\ c$	$b\ a\ c$	$b\ a\ c$
		Lst	0 0 0	1 0 0	1 1 0	1 1 0	1 1 0	1 1 0	0 1 0
		Rst	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	1 0 0	0 0 0
			$a\ c_W$	$a\ c_W$	$a\ c_W$	$a\ c_W$	$a\ c_W$	$a\ c_W$	$a\ c_W$
S_3	ac_W	Lst	0 0	1 0	1 0	1 0	1 0	1 0	1 0
		Rst	0 0	0 0	0 0	0 0	0 0	0 0	0 0
			$b\ a_W$	$b\ a_W$	$b\ a_W$	$b\ a_W$	$b\ a_W$	$b\ a_W$	$b\ a_W$
S_1	ba_W	Lst	0 0	0 0	1 0	1 1	1 0	1 0	0 0
		Rst	0 0	0 0	0 0	0 0	0 0	1 0	0 0
			$c\ b_W$	$c\ b_W$	$c\ b_W$	$c\ b_W$	$b_W\ c$	$b_W\ c$	$b_W\ c$
S_2	cb_W	Lst	0 0	0 0	0 0	0 0	1 0	1 0	0 0
		Rst	0 0	0 0	0 0	0 0	0 0	1 0	0 0

ックを引き起すことはないことを示す.

もし2つのライトアクション a_W と b_W とがデッドロックを引き起しているとすれば，これは次の2条件が成り立っていることと等価となる；

(1)　2つのベーシックループ；

$l_1=x_1x_2\cdots x_ka_W$ $(1\leq{}^{\exists}s\leq k\Rightarrow\ x_s=b)$

$l_2=y_1y_2\cdots y_jb_W$ $(1\leq{}^{\exists}t\leq j\Rightarrow\ y_t=a)$

が存在する.

(2)　l_1, l_2 に対応するロックステータス $\mathrm{Lst}_1, \mathrm{Lst}_2$ が；

$$\overset{x_1\ x_2\cdots\ x_k\ a_W}{\mathrm{Lst}_1=(m_1, m_2, \cdots, m_k, 0)},\quad \overset{y_1\ y_2\ \cdots y_j\ b_W}{\mathrm{Lst}_2=(n_1, n_2, \cdots n_j, 0)}$$

となっている．ただし，$x_1, \cdots, x_k, y_1, \cdots y_j$ はクラスまたはライトアクションで，$m_1, \cdots m_k, n_1, \cdots n_j$ は正の整数である．このとき次の命題が成り立ち，デッドロックは起り得ないことがわかる.

命題 4　上の2条件が成り立っているときには必ずあるベーシックループが存在して，その実行ステータスはコンフリクティングとなっている.

証明　l_1, l_2 はともにベーシックループだから，記述 $ay_{t+1}\cdots y_jbx_{s+1}\cdots x_ka$ はループである．したがって，これをインプライするベーシックループ l が存在する．条件(2)より，l の実行ステータスはコンフリクティングとなっている.　　$q.e.d$

6.3　セルフループの扱いについて

今までは，便宜上セルフループは除いて議論してきた．しかし，これをベーシックループに含めることは容易である．ただし，コンフリクティングか否かの判定は，Lst が2となったか否かで行う．これはセルフループを含むクラスにおいては，自身のリードセットとライトセットとが交差するため，そのようなクラスに属する2つのトランザクションを同時に実行することは，順序づけコンシステンシィを損なうことになるという事実に基づく．その他の制御方式については，通常のベーシックループと同様となる.

6.4　トランザクション実行の例

第4.4節の例に基づき，トランザクションの実行例を以下に示す.

例における営業，購入，検査の各部門はそれぞれフラグメント F_1, F_2, F_3 を保持する分散データベースのサイトを構成していると仮定する．このとき，各々を S_1, S_2, S_3 で表わす．次に，営業部門 S_1 がユーザより注文を受け，販売業務を行おうとしているとする.

(1)　S_1 は自サイト内に販売業務処理のためのプロセス P を発生させる．このときの各実行ステータスを表3中の t_0 で表わす.

(2)　P は REQ RD (T_a) を S_1, S_2, S_3 に送る（このとき P と S_1 とは自サイト内通信，他は回線を介した通信となる).

(3)　REQ RD (T_a) を受信したサイトは ACK を応答する（実行ステータスは t_1 となる).

(4)　S_1, S_2, S_3 から ACK を受信した P はリードアクション（F_3 の読み込み）を実行し，更新データ（F_1 への追加内容）を作り出す.

(5)　次に，P がライトアクションを実行する直前に，購入部門が S_2 内に発注業務処理のためのプロセ

ス P' を発生させ，P' は S_1, S_2, S_3 に REQ RD (T_b) を送信したとする．

(6) REQ RD (T_b) を受信した各サイトは ACK を P' に返す．(t_2 に示す)．

(7) 更新データを作成した P は，T_a のライトアクション実行のため REQ WT (T_{aw}) を S_1 に送信する．

(8) REQ WT (T_{aw}) を受信した S_1 はベーシックループ ba_w をコンフリクティングとしてしまうため NACK を P に返送する (t_2' に示す)．

(9) NACK を受取った P は ABORT (T_{aa}) を S_1 に送り，実行ステータスは t_2 に戻る．

(10) 一方，S_1, S_2, S_3 より ACK を受信した P' は，リードアクションを実行し，更新データ作成後 REQ WT (T_{bw}) を S_2 に送る．

(11) S_2 は ACK を送る．(t_3 に示す)．

(12) P' は F_1, F_2 に対するライトアクションを実行する．

(13) P' は更新終了後 COMPLETE (T_b) を S_1, S_2, S_3 に送る．COMPLETE 受信後の実行ステータスは t_4 となる．このあと変換 φ により実行ステータスは t_5 のようになる．

(14) (9)でペンディングとなっていた a_w は受付可能となり，P による REQ WT (T_{aw}) 送信へと続く．

7. 結　論

本稿では分散データ処理におけるデータの一貫性維持のため，入力トランザクションを事前に解析し実行を制御する手法を提案した．特に筆者は，本方式の分散型オフィスシステムへの適用を検討しているが，もちろんこの方式は，分散データ処理のみならず複数ユーザがデータを共有するときすべてに適用可能である．なお，リソースデータをロックする従来の方式では，ロック要求を，リソースを保有するサイトにのみ送れば良いのに対し，本稿で提案したトランザクションによるトランザクションのロック方式は，ロック要求を関連するすべてのサイトにブロードカストしなければならないため伝送コストが増すことが考えられる．一方，リソースロックの方式では，一旦ロックを行えばそのリソースを用いるすべてのトランザクションがブロックされるのに対し，本方式では実行ステータスがコンフリクティングとならない限り，リソースを共有する多数のトランザクションが同時実行できるため，処理の並列性は向上する．このため，たとえば Ethernet のように，ブロードカスト伝送のコストが1対1通信のそれとあまり変らないようなネットワークには本方式は最適な方式となっている．残された課題として，筆者等は，一般のネットワークの場合について，この並列性向上と伝送コストの増加というトレードオフを定量的に評価すべく検討をすすめている．

謝辞 最後に本研究をすすめるに当り多大の援助と助言を与えていただいた沖電気工業株式会社研究所，松下温博士，長谷川潔氏ならびに研究室諸兄に深謝する．

参 考 文 献

1) Rothnie, J. B. and Goodman, N.: A Survey of Research and Development in Distributed Database Management, Proc. of Int. Conf., 3rd VLDB, pp. 48-62 (1977).
2) Rothnie, J. B. et al.: Introduction to a System for Distributed Database (SDD-1), ACM Trans. on Database System, Vol. 5, No. 1, pp. 1-17 (1980).
3) Bernstein, P. A. et al.: The Correctness of Concurrency Control Mechanisms in a System for Distributed Database (SDD-1), ACM Trans. on Database System, Vol. 5, No. 1, pp. 52-68 (1980).
4) Stonebraker, M.: Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES, IEEE Trans. on software engineering, Vol. SE-5, No. 3 (May 1979).
5) Metcalfe et al.: Ethernet: Distributed packet switching for Local Computer Networks, CACM, Vol. 19, No. 7 pp. 395-404 (July 1976).
6) 山崎他：分散データベースにおける同期制御のための階層型プロトコル，情報処理学会，分散システム研究会 2-1 (1979年9月)．
7) Yamazaki, H. et al.: A Graph Theoretic Approach for Fault Detection and Recovery in a Distributed Database, Proc. of ICCC, pp. 237-242 (1980).

(昭和56年3月2日受付)
(昭和56年7月13日採録)