分散データベースにおける同時実行制御のーアルゴリズム

丸本　悟　　岸本　一男　　翁長　健治
（広島大学工学部）

1. まえがき

データベース システムの利用者は，トランザクションと呼ばれる論理的な処理単位を通し，データの読み込み，書き出しを行う．具体的なトランザクションの例としては、利用者の発するonline queryの列とか、個々の応用プログラムといったものがあげられる．通常，個々のトランザクションを単独で実行した場合，システムのデータ一貫性は保たれるとする．しかし，複数のトランザクションが並列に処理される場合，トランザクション間の相互作用によりデータ一貫性が崩れることがありうる．従って，システムはデータ一貫性を保つように，各トランザクションの読み込み・書き出しを同期制御する必要があり、これをデータベースシステムの同時実行制御問題と呼ぶ(1)．

同時実行制御の方式は，集中型，分散型システムによらず数多く発表されている(2~4)．しかしながら、一般にトランザクションがアクセスするデータは既に読み込んだデータの値に依存して決まり，各方式にトランザクションを並列に処理する能力がどれだけあるかを議論するのは難しい．そのため，既に発表されている同時実行制御方式の多くは，システムのスループット向上等を目的としたトランザクションの実行順序の制御が十分でない．

本稿では，トランザクションの管理及びデータの管理が並列に行える分散データベースにおいて，システムのスループット向上を計るための方策を考察し、それに基づいた同時実行制御アルゴリズムを提案する．提案するアルゴリズムの基本的な特徴は、あるトランザクションが他の待ち状態にあるトランザクションの終了を待つといった待ちの連鎖を短くおさえようとする点にある．提案するアルゴリズムでは，待ちの連鎖が発生した場合，それに参加しているトランザクションに一時的な実行をゆるし、その実行結果により待ち関係の逆転を行う．そのため，トランザクションを部分的にロールバックする機能が必要である．

以下，2.で分散データベースモデル，3.で2相ロック方式による同時実行制御，4.で提案するアルゴリズムの基本的なアイディアについて触れた後，5.でアルゴリズムの提案を行う．

2. 分散データベース モデル

2.1 システムアーキテクチャ

分散データベースシステムの構成を図2.1に示す．システムは，トランザクションを管理するTM(Transaction Manager)，データを管理するDM(Data Manager)，及びこれらを結ぶ通信路により構成される．TM，DMは独立に動作し、TMは任意のDMと，逆にDMは任意のTMと通信可能であるが，TM同志又はDM同志の通信はできないとする．また，本稿では簡単化のためTMはひとつのトランザクションを，DMはひとつのデータ項目に対応するデータのみを管理しているとする．従って個々のTM，DMは，それぞれが管理する

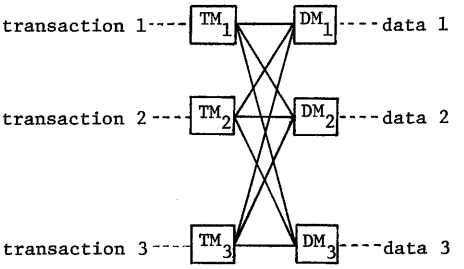

図2.1 システム アーキテクチャ

トランザクション名，データ項目名により識別される．

2.2 トランザクションモデル

データー貫性を保つ代表的な方法として2相ロック方式がある．この方式では，TMであるデータ項目にアクセスする必要が生じたとき，TMは対応するDMにデータをロックする要求を出す．TMは，ロックの要求が認められるのを待ちデータにアクセスする．一方DMがロックの要求を受け取った場合，要求されたデータが既にロックされているならこれを待たせる．ロックされていなければ，これを認め直ちにデータをロックする．通常ロックの開放はトランザクションの終了時に行われる．

ロックの種類にもいくつか存在するが，代表的なものとしてshared lock（又はread lock）とexclusive lock（又はwrite lock）がある．前者は，データを読み込むだけの場合に，後者はデータを書き替える場合に要求される．

本稿では，議論を簡単にするためトランザクションを同期制御の対象となるロック要求（以後単に要求と書く）の列とみなし，種類もexclusive lockのみとする．以下では，トランザクションを T_i $(i=1,2,\cdots)$ と書き，T_i における j 番目の要求を r_{ij} と書く．また T_i を管理するTMを TM_i と表わし，一時に存在するトランザクションの数は，システム内のTMの数を越えないとする．

2.3 トランザクション処理モデル

データ項目名を D_i $(i=1,2,\cdots,m_d)$ と書き d を要求 r_{ij} の対象となるデータ項目名を与える関数とする．即ち $d(r_{ij})$ は，T_i の j 番目の要求の対象となるデータ項目名又はそれを管理するDMを表わす．これらの記号を用い，トランザクション処理をモデル化する．

トランザクション T_i を管理する TM_i はローカルな作業領域を持ち，r_{ij} を順に処理してゆく．TM_i は r_{ij} に出会ったとき，$D_k=d(r_{ij})$ にロック要求を送信し，D_k の応答を待つ．D_k から要求を認められれば，TM_i はデータへアクセスできる．TM_i がアクセスしたデータの値はすべて，TM_i のローカルな作業領域に記憶される．以後トランザクションの実行に伴う TM_i 上のローカルな処理ではこれらの値が参照される．また，データの書き替えも，この作業領域内に記録され，D_k の書き替えは，トランザクションの終了まで遅らされる．TMは，トランザクションの終了と同時に，アクセスしたデータを管理するDMに，書き替えるべきデータを送信し，DMはこれによりデータの変更を行う．これはシステムの障害回復を容易にする2相コミットと呼ばれる手法によるものである．従って，ロックの開放はトランザクション終了時にまとめて発生する．

3. 2相ロック方式による同時実行制御

3.1 2相ロック方式

2相ロック方式は既に述べたように次のような特徴を持つ．

i) トランザクションはデータにロックをかけなければアクセスできない．

ii) 一度データがロックされれば，これが開放されるまで，他のトランザクションが重複してロックすることはゆるされない．

iii) ロックの開放は，トランザクションの終了時にまとめて行われる．

通常2相ロック方式は，トランザクションの実行中にひとつでもロックを開放すれば，以後いかなるロックも要求できないというものであるが，ここでは2相コミットを考慮してiii)のような制約におきかえた．

上記i)～iii)により，あるトランザクションが実行中にアクセスしたデータは，それが終了するまで他のトランザクションにより変更されることはない．従ってある時点のシステムの状態は，それまでに終了したすべてのトランザクションをある順番に沿ってひとつずつ実行した結果に一致する．即ち直列化可能でありデーター貫性が保証され

る[3].

基本的な2相ロック方式では，DMはロック要求をFIFOに入れ管理する．ロック要求が認められるのは，FIFOの先頭に位置したときであり，対応するトランザクションの終了とともにFIFOから消去される．同一データ項目Dに対する二つの異なるロック要求を r_{ij}，$r_{i'j'}$ とする．FIFO中で r_{ij} が $r_{i'j'}$ より前にある，即ち r_{ij} の方が $r_{i'j'}$ より先にDに到着したなら $T_i \succ_D T_{i'}$ と書き，T_i は $T_{i'}$ に先行すると言う．複数のトランザクション T_i ($i=1,2,\cdots,n$) に対し，ある時点で発生する $T_1 \succ_{D_1} T_2 \succ_{D_2} \cdots \succ_{D_{n-1}} T_n$ のような関係を待ちの連鎖と呼ぶ．ただし，$D_1 \neq D_2 \neq \cdots \neq D_{n-1}$ である．2相ロック方式では，長い待ちの連鎖が発生する危険性がある．また $T_1 \succ_{D_1} T_2 \succ_{D_2} \cdots \succ_{D_{n-1}} T_n \succ_{D_n} T_1$ といった場合はデッドロックであり，デッドロックの検出及び回避のためのアルゴリズム[6,7]が数多く発表されている．

3.2 2相ロック方式によるトランザクション実行順序

2相ロック方式によるトランザクションの実行順序を決める問題を考えるため，

A_i：T_i がアクセスするデータ項目の集合

S_i：T_i がデータ項目($\in A_i$)にアクセスする順番

ST_i：T_i がスタートする時刻

t^i_{jk}：r_{ij} から r_{ik} までに要する処理時間（TMのローカルな処理時間＋DMのデータアクセス時間）

とし，A_i，S_i が固定されているとして次のような重みつき有向グラフ $Gr=(V,E)$ を定義する．

$V = \{T_i \mid i=1,2,\cdots,m_t\} \cup \{s,e\}$

$E = E_s \cup E_c \cup E_e$

$E_s = \{(s,T_i) \mid i=1,2,\cdots,m_t\}$

$E_c = \{(T_i,T_j) \mid A_i \cap A_j \neq \phi,\ i \neq j\}$

$E_e = \{(T_i,e) \mid i=1,2,\cdots,m_t\}$

であり，枝重 $W=\{w_e \mid e \in E\}$ は次式で表わされる．

$$w_e = \begin{cases} 0 & e=(T_i,e)\in E_e \\ ST_i + t_{in_i} + C_i & e=(s,T_i)\in E_s \\ t_{jn_k} + C_k,\ j=\min\{l \mid d(r_{kl})\in A_i \cap A_k\} & e=(T_i,T_k)\in E_c \end{cases}$$

ただし，C_i は T_i のコミット処理に要する時間，n_i は T_i 中の要求の数，m_t はトランザクションの数とする．この時，2相ロック方式によるトランザクションの実行順序を決める問題は，Gr に存在する逆向きの枝の対 $(T_i,T_j),(T_j,T_i)$ のうちどちらか一方を消去してアサイクリックな有向グラフ Gd を求める問題となる．Gd において，w_e を枝の長さとしたSから T_i への最長パス長は，T_i の終了時刻を，SからeへのパスはT全トランザクションが終了する時刻を表わす．図3.1(b),(c)に図3.1(a)に対応する Gr，Gd の例を示す．図3.1(b)中の太線で示した経路は，S-e間の最長パスを表わす．

一般に，A_i，S_i は既に読み込んだデータの値によって変化し，t^i_{jk} も一定ではないの

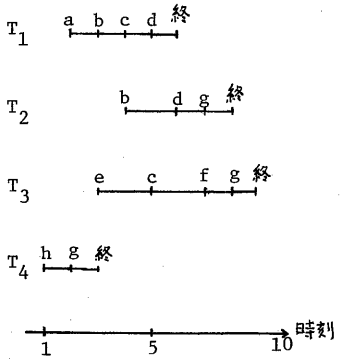

図3.1 (a) トランザクション例

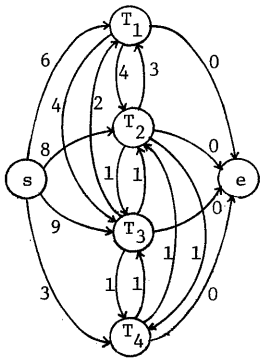

図3.1 (b) グラフ Gr

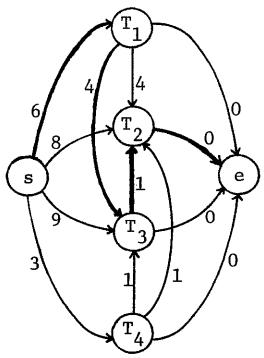

図3.1 (c) グラフ Gd

で、トランザクションの実行順番を決めるために Gr を使用するのは不可能である。2相ロック方式では、DMに要求が到着した順番により トランザクションの実行順序が決まり、デッドロックが発生しない限り、順番は変化しない。

4. 同時実行制御の基本的アイディア

4.1 待ち連鎖の縮小

2相ロック方式では、$T_1 \succ_{b_1} T_2 \succ_{b_2} \cdots \succ_{b_{n-1}} T_n$ のような待ちの連鎖が発生した場合、実行を継続できるのは先頭の T_1 のみである。さらに $T_n \succ_{b_n} T_1$ なる関係が加われば、デッドロックになる。従って、ひとつの待ち連鎖をより短い複数のものに分割することは、i) より多くのトランザクションに実行継続ゆるすことができる、ii) 大規模なデッドロックが発生する危険性が減少する、といった点で有利である。

本節では、図4.1に示すような三つのトランザクションにより構成される単純な待ち連鎖 $T_1 \succ_a T_2 \succ_b T_3$ の考察に基づき、待ち連鎖を分割する基準を設定する。図4.1中の記号は、次のような意味を持つ。

- a, b ---- T_1, T_2, T_3 において要求が発生した点
- α ---- T_2 においてbへの要求が発生し、aへの要求が発生するまでの処理時間
- C ----- max (T_2が待ちになる時刻, T_3が待ちになる時刻)
- E_1, E_2, E_3--- 2相ロック方式による T_1, T_2, T_3 の終了時刻
- β_2, β_3--- T_2, T_3 を終了させるための処理時間

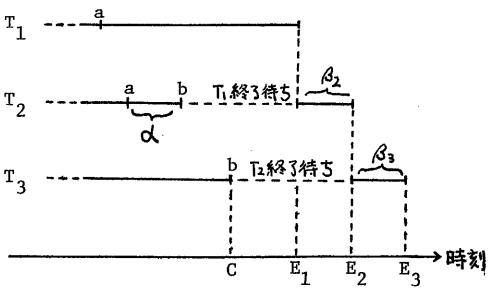

図4.1 待ち連鎖 $T_1 \succ_a T_2 \succ_b T_3$

T_1, T_2, T_3 において他の待ち関係は発生せずこれらをどの順番に実行しても、処理内容は変化しないとすれば、次の性質1が言える。

性質1 T_1, T_2, T_3 すべてが終了する時刻を T_1, T_2, T_3 の順に終了させた場合の E, T_1, T_3 を並列に実行し、両方が終了した後 T_2 の残りを実行した場合の E' に分ける。この時、$E_1 > C + \alpha$ なら $E \geq E'$ である。

証明 $T_1 \succ_a T_2 \succ_b T_3$ から $\alpha > 0$ で、$E = E_1 + \beta_2 + \beta_3$, $E' = \max(E_1, C + \beta_3 + \alpha) + \beta_2$ である。

i) $E' = C + \alpha + \beta_2 + \beta_3$ の場合、$E - E' = E_1 - (C + \alpha)$ であり、$E_1 > C + \alpha$ なら $E > E'$.

ii) $E' = E_1 + \beta_2$ の場合、$\beta_3 \geq 0$ より明らかに $E_1 > C + \alpha$ であり、$E - E' = \beta_2 \geq 0$. ■

これより、次のような待ち連鎖の分割基準を設定する。

待ち連鎖分割基準 ある時点で $T \succ T'$ であるとする。T' を T より先に終了させるために T で無効となる処理時間を α, T, T' 両方が待ち状態になってから、T が実行を再開するまでに、T' が処理を継続できる時間を t とする。t は、T' に一時的な実行継続をゆるすことにより計測できる。$t > \alpha$ なら $T' \succ T$ として、待ち連鎖を分割する。

4.2 ロールバック

4.1で述べた基準により、トランザクション間の待ち関係を調整するためには、各トランザクションの既に実行された任意の要求発生時から実行を再開する必要がある。そのためには、TMが新しい要求が発生するたびに、その時点のローカル変数の値を記録保存しておけばよい。本稿では、トランザクションをある要求発生時の状態まで戻すことをロールバックと呼ぶ。ロールバックが発生すれば、TMは指定された要求発生時以降のローカル変数の記録を消去する。実行を再開する場合、その要求発生時に記録保存されたローカル変数の値を使用して、引続く処理を行う。以下の章では、ロールバックに必要な処理すべてを、TMが自動的に行っているものとする。

5. 同時実行制御アルゴリズム

5.1 メッセージ

TMとDM間の通信メッセージとして、次のものを使用する。前半の六つは、TMからDMへ，後半の三つはDMからTMへ送信されるメッセージである。

1) $R_{ij}(s_i)$--- T_iのj番目の要求r_{ij}に対応するメッセージで、s_iは送信時のTMの状態を表わす。

2) $CANCEL_i(b)$--- TM_iがDMに対し既に送信したメッセージを取り消すか又は変更するためのメッセージである。($b=0,1,2$)

3) $END_{ij}(v)$ TM_iの終了を表わし、vは$d(r_{ij})$が変更されるべき値である。

4) $DOWN_i(d)$---TM_iが待ち状態になったことDMに伝えるメッセージである。DMが$d(r_{ij})$を管理している場合，dはT_iをr_{ij}までロールバックさせた場合に無効となる処理時間である。

5) $READY_i(ts)$--- TM_iがDMに対し、トランザクションの実行を再開できる状態になったことを伝えるメッセージで、tsは送信時刻のタイムスタンプである。

6) UP_i ------ TM_iがトランザクションの実行を再開することを伝えるメッセージである。

7) $TRY_j(v,t,d)$--- R_{ij}に対するDMの応答メッセージである。vはデータの値，tはデータアクセスに要した時間，dはT_iが実行順番を逆転させるために、一時的な実行を行わねばならない時間である。

8) $WAIT_j$----- R_{ij}に対するDMの応答である。これを受信したTM_iは待ち状態になる。

9) ACK_j ------ $READY_i$に対する応答で、T_iが実行を再開することを認めるメッセージである。

5.2 TMのアルゴリズム

T_iを管理するTM_iの処理を示す。すべてのTMは標準時刻に同期した時計を持っていると仮定する。TM_iは，空き状態(S0)実行状態(S1)，待ち状態A，B(S2，S4)及び仮実行状態(S3)の五つの状態を繰り返す。実行状態では$T_{i'} > T_i$のような$T_{i'}$は存在しないが，待ち状態Aではこのような$T_{i'}$が存在する。更に仮実行状態では$T_{i'} > T_i$である任意の$T_{i'}$に対し、$T_{i''} > T_{i'}$のような$T_{i''}$が存在する。待ち状態Bは，$T_{i'} > T_i$を$T_{i'} < T_i$に変更するために必要な状態である。

TM_iが使用するローカルな変数$t_{j,j+1}$は，r_{ij}発生からr_{ij+1}発生までに要した処理時間，f_1,f_2はそれぞれ実行状態，仮実行状態で発生したr_{ij}に対するjの最大値である。f_jは、$d(r_{ij})$から最後に受信したメッセージがTRY_jであるか，$WAIT_j$であるかを表わすブーリアンである。前者の場合はf_j=true，後者の場合はf_j=falseとなる。

図5.1に、TMの状態遷移図を示す。

《TMのアルゴリズム》

0° 空き状態

T_iが発生すれば，$s_i \leftarrow 0$，$j_1 \leftarrow 1$，$j_2 \leftarrow 1$ $t_s \leftarrow$ (T_i発生時刻)，$t_{01} \leftarrow 0$として1°へ移る.[T_{01}].

1° 実行状態

1.1° T_iが終了すれば，$d(r_{ik})$，($k=1,2,\cdots n_i$)に$END_i(v)$を送信して1°へ戻る。vは、$d(r_{ik})$を変更すべき値である[T_{01}].

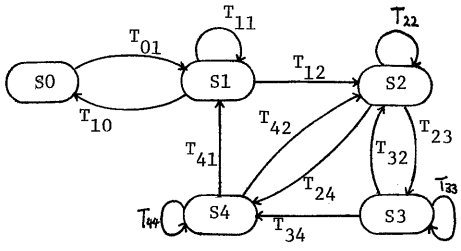

図5.1 TMの状態遷移図

1.2° ローカルな処理を繰り返し，r_{ij_1}が発生すれば $t_{j_{l-1}j_l} \leftarrow$(($r_{ij_1}$の発生時刻) $-t_s + t_{j_{l-1}j_l}$，$\alpha_{j_1} \leftarrow \infty$ とし，$d(r_{ij_1})$に $R_{ij_1}(s_i=0)$を送信して応答を待つ．

1.2.1° 応答として $TRY_{j_1}(v, t, \alpha)$を得れば，$v_{j_1} \leftarrow v$，$t_{j_1 j_{l+1}} \leftarrow t$，$\alpha_{j_1} \leftarrow \alpha$，$f_{j_1} \leftarrow$ true，$t_s \leftarrow$(TRY受信時刻)，$j_l \leftarrow j_{l+1}$，$j_2 \leftarrow j_1$として1°を続ける〔T_{11}〕．

1.2.2° 応答として $WAIT_{j_1}$を得れば，$d(r_{ik})$，$(k=1,2,\cdots,j_{l-1})$に $DOWN_i(\alpha = \sum_{j=k}^{j_{l-1}} t_{jj+1})$を送信し $s_i \leftarrow 1$とした後2°へ移る〔T_{12}〕．

2° 待ち状態A

2.1° $TRY_j(v, t, \alpha)$を受信すれば，$f_j \leftarrow$ true とする．v_jが未定又は $v_j \neq v$ であるなら，$\alpha_j \leftarrow \alpha$，$t_{jj+1} \leftarrow t$，$t_s \leftarrow$(TRY受信時刻)とし，$d(r_{ik})$，$(k=j_l+1, j+2, \cdots, j_2)$に $CANCEL_i(b=0)$を送信し，$j_2 \leftarrow j$ として r_{ij}までロールバックした後 $v_j \leftarrow v$とする．引き続き2°を繰り返す〔T_{22}〕．

2.2° $WAIT_j$を受信すれば，$f_j \leftarrow$ false とし，$j \leq j_1$なら $d(r_{ik})$，$(k=j+1, j+2, \cdots j_1)$に $CANCEL_i(b=1)$を送信し $j_1 \leftarrow j$として2°を繰り返す〔T_{22}〕

2.3° 次の条件C1又はC2のいずれかを満たす ℓ $(j_1 \leq \ell \leq j_2)$が存在すれば $t_s \leftarrow$(条件成立時刻)として以下の処理を行う．

$$C1 \quad \bigwedge_{k=j_1}^{\ell} f_k = \text{true}$$

$$C2 \quad \sum_{k=m}^{\ell} t_{k,k+1} > \alpha_m \quad (m=j_1, j_1+1, \cdots, \ell)$$

2.3.1° C1，C2を共に満たす最大の ℓに対し，$d(r_{ik})$，$(k=1,2,\cdots,\ell)$に $READY_i(t_s)$，$d(r_{ik'})$，$(k'=\ell+1, \ell+2, \cdots, j_2)$に $CANCEL_i(b=0)$を送信する．$j_2 \leftarrow \ell+1$とし $r_{i\ell+1}$までロールバックした後 4°へ移る〔T_{24}〕．

2.3.2° ℓが，C1のみ満たし $\ell = j_2$ なら 3°へ移る〔T_{23}〕．

3° 仮実行状態

3.1° T_iの終了点に達したなら2°に戻る．ただし，ENDメッセージは送信しない〔T_{32}〕．

3.2° ローカルな処理を繰り返し，r_{ij_2}が発生すれば，$t_{j_{2-1}j_2} \leftarrow$(($r_{ij_2}$の発生時刻) $-t_s + t_{j_{2-1}j_2}$，$\alpha_{j_2} \leftarrow \infty$とし，$d(r_{ij_2})$に $R_{ij_2}(s_i=1)$を送信して応答を待つ．

3.2.1° $TRY_{j_2}(v, t, \alpha)$を得れば，$v_{j_2} \leftarrow v$，$t_{j_2 j_2+1} \leftarrow t$，$\alpha_{j_2} \leftarrow \alpha$，$f_{j_2} \leftarrow$ true，$t_s \leftarrow$(TRY受信時刻)とする．j_2に対しCが成立すれば，$d(r_{ik})$，$(k=1,2,\cdots,j_2)$に $READY_i(t_s)$を送信し $j_2 \leftarrow j_2+1$として4°に移る〔T_{34}〕．そうでなければ，$j_2 \leftarrow j_2+1$として3°を続ける〔T_{33}〕．

3.3.2° $WAIT_j$を受信すれば，$f_j \leftarrow$ false として2°に戻る．このとき $j < j_1$なら $d(r_{ik})$，$(k=j+1, j+2, \cdots, j_1)$に $CANCEL_i(b=1)$を送信し $j_1 \leftarrow j$とする．〔T_{32}〕．

4° 待ち状態B

4.1° $WAIT_j$を受信すれば，$d(r_{ik})$，$(k=1,2,\cdots j_2)$に $CANCEL_i(b=3)$を送信し $f_j \leftarrow$ false として2°に戻る．このとき，$j < j_1$なら $d(r_{ik'})$，$(k'=j+1, j+2, \cdots, j_1)$に $CANCEL_i(b=1)$を送信し $j_1 \leftarrow j$とする〔T_{42}〕．

4.2° ACK_j受信時，$d(r_{ik})$，$(k=1,2,\cdots,j_2)$すべてからACKを得たことになればこれらに UP_iを送信し，$t_s \leftarrow$(UP_i送信時刻)，$s_i \leftarrow 0$として1°へ移る〔T_{41}〕．そうでなければ4°を続ける〔T_{44}〕

5.3 DMのアルゴリズム

DMも，TMと同様に標準時刻に同期した時計を持っているとする．DMは，空き状態(SO)，データ変更待ち状態(S1)実行順変更待ち状態(S2)の三つの状態を繰り返す．DMは，TMからの $R_{ij}(s_i)$に対応する要求 r_{ij}を，$s_i=0$ ならQ1，$s_i=1$ ならQ2に入れ管理する．また $READY_i(t_s)$は ts_iをQ3に入れて管理する．Q1はFIFO，Q2は単なるリスト，Q3は tsによる優先順位キューである．これらの構造を図5.2に示す．

DMがデータ変更待ち状態にあるとき，Q1の先頭にある要求 r_{ij}を発したトラン

ザクション T_i のみが実行状態にあり，他は待ち状態Aにある．DMが実行順変更待ち状態にあるとき，Q1，Q2 中の r_{ij} を発したトランザクション T_i は、仮実行状態かまたは待ち状態Aにある．図5.3 にDMの状態遷移図を示す．

《DMのアルゴリズム》

0° 空き状態

$R_{ij}(s_i)$ を受信すれば，TM$_i$ に TRY$_{ij}$(v, t, α) を送信する．v はデータの値，t はデータアクセスに要した時間，$\alpha=0$ である．
$s_i=0$ なら、r_{ij} をQ1に追加し1°へ移る〔T_{01}〕．
$s_i=1$ なら，r_{ij} をQ2に追加し2°へ移る〔T_{02}〕．

1° データ変更待ち状態

1.1° CANCEL$_i$(b) を受信した場合，b=0 なら r_{ij} をQ1またはQ2から除く．b=1 なら r_{ij} をQ1からQ2に移す．b=2 なら，ts_i をQ3から除く．
Q1の先頭 r_{ij} に対し，END$_i$(v) を受信した場合，データの値を v に変更しQ1から r_{ij} を除く．
CANCEL，ENDメッセージによりQ1の先頭が変化した場合，以下の処理を行う．そうでなければ1°を続ける．

1.1.1° Q1≠φ なら，Q1の新しい先頭 r_{ij} に対し、TM$_i$にTRY$_i$(v, t, α=0) を送信し．Q3を空として1°を続ける〔T_{11}〕．

1.1.2° Q1=φ，Q2≠φ なら，Q2内の r_{ij} すべてに対しTRY$_i$(v, t, α=0) を送信して2°へ移る〔T_{12}〕．

1.1.3° Q1=Q2=φ なら0°へ戻る〔T_{10}〕．

1.2° Q1の先頭を r_{ij} とする．DOWN$_i$(α) を受信した場合，RC←α とし．r_{ij} 以外のQ1及びQ2の要素に r_{ij} に対しTRY$_{ij}$(v, t, α=RC) を送信しQ3を空とした後．2°へ移る〔T_{12}〕．

1.3° Q1の先頭を r_{ij} とする．READY$_i$(ts) を受信すれば，Q3に ts_i=ts を追加しTM$_i$にACK$_{ij}$を返信して1°を続ける〔T_{11}〕．

1.4° Q1の先頭 r_{ij} に対し，UP$_i$ を受信すれば、Q3から ts_i を除き1°を続ける〔T_{11}〕．

2° 実行順変更待ち状態

2.0° $R_{ij}(s_i)$ を受信した場合，$s_i=0$ なら，r_{ij} をQ1に追加して2°を続ける．
$s_i=1$ なら r_{ij} をQ2に追加し、TM$_i$にTRY$_i$(v, t, α=RC) を返信して2°を続ける〔T_{22}〕．

2.1° CANCEL$_i$(b) を受信した場合，b=0 ならQ1またはQ2から r_{ij} を除く．b=1 なら r_{ij} をQ1からQ2に移す．b=3 ならQ3から ts_i を除く．2°を続ける．

2.2° READY$_i$(ts) を受信すれば，ts_i をQ3に追加し、ソートし直す．2°を続ける．

2.3° 条件に応じて以下の処理を行う．

2.3.1° Q1≠φ で、Q3の先頭にACKを出していなければ、Q1の先頭 r_{ij} に対しTRY$_{ij}$(v, t, α=0) を，r_{ij} 以外のQ1，Q2の要素に対しWAITを送信する．1°に戻る〔T_{21}〕．

2.3.2° Q1=φ，Q3≠φ でQ3の先頭 ts_i に対し(現在時刻)≧ts_i+γ となった時点で、TM$_i$にACK$_i$を送信し2°を続ける．ただし、γ は、TM，DM間の最大通信遅延とする〔T_{22}〕．

2.3.3° Q1=Q2=φ となれば0°に戻る〔T_{20}〕．

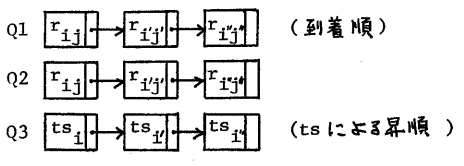

図5.2 DMの使用するデータ構造

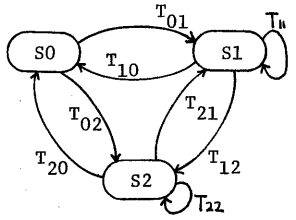

図5.3 DMの状態遷移図

2.4° Q3の先頭 ts_i に対し，UP_i を受信すれば，r_{ij} 以外のQ1，Q2の要素にWAITを送信する．r_{ij} をQ1の先頭に移し，Q3を空にして1°へ戻る．[T_{21}]

6. むすび

トランザクションの実行を任意の地点から再開できるという前提のもとで，トランザクション間に発生する待ちの連鎖を短くおさえようとする，同時実行制御アルゴリズムを提案した．データの変更はTMが実行状態にある時に限り発生し，DMは一時にひとつのTMに対してのみ実行状態にあることをゆるすといった点から，提案したアルゴリズムがデーター貫性を保つことがいえる．

今後の課題としては，

i) アルゴリズムの効率化
ii) 2相ロック手法を用いた場合の，トランザクションのスケジューリング問題の解析
iii) シュミレーション等による他方式との比較
iv) トランザクションを部分的にロールバックする方式の詳細な考察

などが急務である．

参考文献

[1] Date, C.J. : An Introduction to Database Systems Volume II, Addison Wesley Publis-ing Co.

[2] Bernstein,P.A. and Goodman,N.: Concurrency Control in Distributed Database Systems, ACM Computing Surveys, Vol.13, No.2,pp.185-221(1981).

[3] 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).

[4] Badal, D.Z. : Correctness of concurrency control and implications in distributed databases, in Proc. COMPSAC 79 Conf., Chicago, III., Nov. 1979.

[5] Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M. : System Level Concurrency Control for Distributed Database Systems, ACM Trans. Database Systems, Vol. 3, No. 2, pp. 178-198(1978).

[6] Isloor, S.S. and Marsland, T.A. : The Deadlock Problem:An Overview, IEEE Computer (1980).

[7] Obermarck, R. : Distributed Deadlock Detection Algorithm, ACM. Trans. Database Systems, Vol. 7, No. 2, pp. 187-208(1982).