データベース・システム 55－2
（1986. 9. 19）

先読みスケジューラを用いた分散型データベースシステムの並行処理制御（2）

原嶋 秀次・茨木 俊秀
豊橋技術科学大学　京都大学

先に我々は、分散型データベースシステムにおける、トランザクションの並行処理制御アルゴリズムを報告した。このアルゴリズムは、各サイトのスケジューラに先読みスケジューラを用いてサイト内での並行処理を行い、全域的なトランザクションの処理問題を時刻印を併用することによって解決している。この方法ではデッドロックが生じないため、一度入力されたトランザクションは拒否されず、再実行の必要がない等の特徴を持つ。

ところで、先の報告ではサイト間にデータのコピーがあるようなモデルを許していなかった。しかし、信頼性が高い、応答が速い等の分散型データベースシステムの特徴をいかすためには、コピーを許すようなモデルに理論を拡張しておくことが望まれる。

本報告では、上述のアルゴリズムをサイト間にコピーがある場合へ拡張し、先読みスケジューラの特徴を保持しつつ、直列可能なスケジュールを出力できる事を示す。

CONCURRENCY CONTROL OF DISTRIBUTED DATABASE SYSTEMS BY CAUTIOUS SCHEDULERS No.2

Shuuzi HARASHIMA
Department of Information and Computer sciences, Toyohashi University of Technology, Tempaku, Toyohashi, Japan 440

and

Toshihide IBARAKI
Department of Applied Mathematics and Physics, Kyoto University, Kyoto Japan 606

In the previous report, we proposed a concurrency control algorithm for distributed database systems. It makes use of time-stamping scheme under the assumption that the database system at each site uses a cautious scheduler. This algorithm enjoys nice properties such as it never deadlocks, and it does not require rollbacks of transactions. In the previous model, however, it is assumed that there are no multiple data copies in the system.

In this report, we remove the single copy assumption from our model, and show that the above nice properties are still preserved in our concurrency control.

1. はじめに

分散型データベースシステム（DDBMS)は、通信コストの低減化、ハードウエアや通信回線等の故障に対する信頼性の向上など、優れた特徴を有している。[BeG 81][Koh 81][CeP 85]。しかし、解決すべき課題も多く、その１つに、トランザクションの並行処理制御の問題がある。

先に我々は、各サイトのスケジューラに先読みスケジューラ[IKK 85]を用い、時刻印を併用する並行処理制御アルゴリズムを提案した[Hal 85]。このアルゴリズムは、入力されたトランザクションは、決して拒否されず、再実行の必要もないという特徴を持ち、並行処理の効果を高め得ると思われる。

しかし、[Hal 85]ではサイト間でのデータの重複のないモデルを対象としていた。このことは、参照主体のトランザクションが様々な場所で発生するようなシステムにおいては（データを複数のサイトに置くことによって効率を上げることが出来るので）、実用上大きな制約となる。そこで本稿では、サイト間にデータの重複がある場合について、分散型データベースシステムのモデル、スケジュールおよびその直列可能性を考察し、アルゴリズムの一般化を行う。

2. 分散型データベースシステムモデル

ここでは、本アルゴリズムが対象とする分散型データベースシステムについて述べる。次節で述べる先読みスケジューラを用いたｎ個のサイトのデータベースシステム（各データベースシステムを**サイト**と呼ぶ）が、通信制御機能を通じて結ばれている。各サイトは独立したデータベースシステムとして機能するが、必要に応じて他のサイトのデータにもアクセス出来るものとする。形式的に述べると、分散型データベースシステム（DDBMS)は**データ項目の集合**$D=\{X,Y,\ldots,Z\}$と外部からシステムへ到着するトランザクションの集合$T=\{T_0, T_1, T_2, \ldots, T_n, T_f\}$から成る。各データ項目Xについて、そのコピー（実データ）が適当なＫ個のサイトに配置されている（Ｋはデータ項目によって異なる）。簡単のためその様な実データの集合をデータ項目Xと同じ文字を用い、$X=\{x_1, x_2, \ldots, x_k\}$と記す。今後、データ項目の集合をＤ、そのコピー（実データ）の集合を$\overline{D}$と記す。各サイトａには、いくつかのデータ項目のコピーが配置されているが、そのデータ項目の集合をD_aとし、実データの集合を$\overline{D}_a$と記す。

トランザクション Tiは、いくつかの**読取りステップ**(read step) Ri[B]（$B \subseteq D$）と**書込みステップ**(wite step) Wi[B]（$B \subseteq D$）とからなる系列である。読取りステップ Ri[B]（書込みステップ Wi[B]）に対し、各 $X \in B$ の Ri[X]（Wi[X]）を**読取り操作**（**書込み操作**）という。ある $x_k \in X$に対して、$\overline{R}i[x_k]$（$\overline{W}i[x_k]$）は、実データx_kの値をTiに渡す（x_kの値を更新する）操作で、**実読取り操作**（**実書込み操作**）と呼ばれる。トランザクションTiの各ステップは、システムによって適当な実操作の集合に置き換えられた後（置き換えの詳細については後述）、実行に移される。Tiは、その１つのステップの実行に際し、置き換えられた全ての実操作の実行が終了後、次のステップの実行要求を発することが出来る。$T_0=W_0[D]$（$=\overline{W}_0[\overline{D}]$）および、$T_f=R_f[D]$（$=\overline{R}_f[\overline{D}]$）は初期値を書き、最終値を読取る架空のトランザクションで、各々**初期トランザクション**、**最終トランザクション**と呼ばれる。

3. スケジュール

本節では、（分散型を含む）データベースシステムにおけるスケジュールの直列可能性について述べる。

Ｔ上のステップの集合をSTEP(T)とする。Ｔ上の**スケジュール**ｓとは、STEP(T)内のすべてのステップからなる系列である。系列は時間の経過に応じて左から右へ並べて記す。ただし、ｓの最初のステップは$W_0[D]$、最後のステップは$R_f[D]$に決められている。STEP(T)内のすべてのステップは、システムによって実操作の集合に置き換えられるが、得られた実操作の集合を$\overline{STEP}(T)$とする。Ｔ上の**実スケジュール**$\overline{s}$とは、$\overline{STEP}(T)$内の全ての実操作からなる系列である。

スケジュール ｓの読取り操作 Rj[X]に対して、Rj[X]より左に位置するXへの書込み操作の中で一番右のものが Wi[X]である時、Tjは Xを Tiから**読取る**と言い、RF_s（Rj[X]）＝iと記す。書込み操作 Wi[X]が更新した Xの値を読取る Rj[X]が存在しない時、Wi[X]は**無効書込み**であると言う。また、Ｔ上の２つのスケジュールｓ

とｓ'において、全ての読取り操作Rj[X]に対して

$RF_s(Rj[X]) = i \Leftrightarrow RF_{s'}(Rj[X]) = i$

が成立するならば、ｓとｓ'は 等価であると言う。

T上の実スケジュール$\overline{s}$の実読取り操作$\overline{R}j[x_k]$に対して、$\overline{R}j[x_k]$より左に位置するx_kへの実書込み操作の中で 一番右のものが$\overline{W}i[x_k]$である時、Tjはx_kを（または、Xを）**Tiから読取る**と言い、$RF_{\overline{s}}(\overline{R}j[x_k]) = i$と記す。実書込み操作$\overline{W}i[x_k]$に 対し、**無効実書込み操作**を上と同様に定義する。T上の２つの実スケジュール $\overline{s}$と$\overline{s}$' において、全ての実読取り操作$\overline{R}j[x_k]$に対して

$RF_{\overline{s}}(\overline{R}j[x_k]) = i \Leftrightarrow RF_{\overline{s}'}(\overline{R}j[x_k]) = i$

が成立するならば $\overline{s}$と $\overline{s}$' は**等価**であると言う。

直列なスケジュールとは、各ステップをトランザクション毎にまとめて、順番に並べたステップの系列である。ｓとｓ'が等価で ｓが直列なスケジュールであるとき、ｓ' を**直列可能なスケジュール**と呼ぶ。いうまでもなく、直列可能性は、データベースの一貫性を保証する条件であって、データベース・スケジューラの目的は、外部から入力されたステップ系列を直列可能なスケジュールに変換することにある。

T上の実スケジュール$\overline{s}$とスケジュールｓが以下の条件を満たすとき、**実スケジュール$\overline{s}$はスケジュールｓを実現する**という。

(1) $\overline{s}$は $\overline{W}_0[D]$で始まり、 $\overline{R}_f[D]$で終わる。
但し、 $\overline{W}_0[D]$（ $\overline{R}_f[D]$）はすべての$x_k \in D$に対する $\overline{W}_0[x_k]$（ $\overline{R}_f[x_k]$）を任意の順序で並べたものである。

(2) ｓ 中の、$j \neq 0,f$ なるトランザクションの各操作Rj[X]（Wj[X]）に対して、$\overline{s}$中に いくつかの $x_k \in X$に対する実操作 $\overline{R}j[x_k]$（ $\overline{W}j[x_k]$）が存在する。

(3) $\overline{s}$内の任意の $\overline{R}j[x_k]$に対し、

$RF(\overline{R}j[x_k]) = i \Leftrightarrow RF(Rj[X]) = i$

である。

$\overline{s}$がｓを実現するとき、$\overline{s}$とｓにおいてトランザクション間の 入出力関係は 等しい。即ち、各トランザクションが 読取る値と 書込む値は、$\overline{s}$とｓにおいて 等しい。

4. TIOグラフと実TIOグラフ

本節では、DDBMS の各実スケジュールにおいて、トランザクション間のデータの入出力関係を示す 実TIOグラフと呼ばれるグラフを定義し、それに基づいて実スケジュールの直列可能性の判定法を示す。

T上の スケジュール ｓに対して、次の様な**TIOグラフ** (Transaction Input/Output graph)、TIO(s)を考える。TIO(s)の節点はT内のトランザクションに対応し、ｓにおいて$RF_s(Rj[X])=i$の時、トランザクションTiの節点からTjの節点へXでラベル付けされた有向枝(Ti,Tj):Xを付す。TiがXに関する無効書込みWi[X]を有する時は、追加枝 (Ti,Ti'):Xを追加節点Ti'と共に導入する。 TIO(s)に 他の節点、枝は存在しない。TIO(s)の節点間に次の２つの性質を持つ様な全順序を定めることが出来るとき、TIO(s)は**DITS**を持つという[IKM 83]。

(1) $Ti \ll Tj$ならばTjからTiへの有向路は、存在しない。

(2) **排除規則**： $g \neq i$ を満たし、同じラベルを持つ ２つの有向枝 (Tg,Th)と(Ti,Tj)に対し、$Tg \ll Tj$ならば $Th \ll Ti$ が成立する。

スケジュール ｓとTIO(s)に関する次の定理は重要である。

定理4.1[IKM 83] ｓが 直列可能であることの必要十分条件は、TIO(s)がT_0を最初としてT_fを最後とする、DITSを持つことである。□

T上の実スケジュール$\overline{s}$に対して、次の様な**実TIOグラフ**、$\overline{TIO}(s)$を 考える。$\overline{TIO}(s)$は やはりT内のトランザクションに対応する節点を持つ。$\overline{s}$において$RF_{\overline{s}}(\overline{R}j[x_k])=i$の時、トランザクションTiの節点からTjの節点へ$x_k$でラベル付けされた有向枝 (Ti,Tj):$x_k$ を付す。Tiがx_kに関する 無効実書込み $\overline{W}i[x_k]$を有する時は、追加枝(Ti,Ti'):x_kを追加節点Ti'と共に導入する。$\overline{TIO}(s)$に 他の節点、枝は存在しない。$\overline{TIO}(s)$の節点間に次の２つの性質を持つ様な全順序を定めることが出来るとき、$\overline{TIO}(s)$は**DITS**を持つという。

(1) $Ti \ll Tj$ならばTjからTiへの有向路は、存在しない。

(2) **排除規則**： x_m、$x_n \in X$および、$g \neq i$ を満たす２つの有向枝 (Tg,Th):x_m と (Ti,Tj):x_n を考える。このとき、$Tg \ll Tj$ならばTh

≪Tiが成立する。

スケジュールsの場合と同様、実スケジュール$\overline{s}$とその実TIOグラフ $\overline{TIO}$(s)の間に次の関係を示すことができる。

定理4.2 実スケジュール $\overline{s}$が実現するスケジュールsが直列可能であることの必要十分条件は、$\overline{TIO}$(s)がT_0が最初でT_fが最後となるような、DITSを持つことである。

証明（⇒） sが直列可能ならば TIO(s)はDITSを持つ。$\overline{s}$はsを実現しているから、TIO(s)における有向枝(Ti,Tj):Xに対し、$\overline{TIO}$(s)には有向枝（Ti,Tj):x_k（ただし、$x_k \in X$）が1本以上 存在する。また逆に、$\overline{TIO}$(s)の 各有向枝(Ti,Tj):x_k に対しTIO(s)に有向枝(Ti,Tj):Xが存在する。ところで、$\overline{s}$ では、Wi[X]に応じて複数の x_kに対して $\overline{W}i[x_k]$を実行し、その一部のみを読取ることがあるので、$\overline{TIO}$(s)には TIO(s)にない追加枝と 追加節点が存在する事がある。しかし、これら以外の有向枝は存在しない。従って、TIO(s)のDITSはそのまま$\overline{TIO}$(s)のDITSになっている。（余分な追加枝および追加節点は、その始点の節点にまとめておけばDITSの条件を乱すことはない。）

（⇐） $\overline{TIO}$(s)の各節点T_iに対し、データ項目X 毎に次の手続きを実行する。

① Tiからの、$x_k \in X$を 満たす全ての有向枝(Ti,T):x_kが 追加節点Ti'へ入っている場合は、それらを1本の有向枝（Ti,Ti'):Xに置き換える。

② Tj（≠Ti）からある$x_k \in X$をラベルに持つ有向枝（Tj,Ti):x_kが存在する場合は、Tiに入る全ての有向枝(T,Ti):x_{ℓ} ,$x_{\ell} \in X$を1本の有向枝(Ti,Tj):Xに置き換える。（この時、実現の条件(3)において、Th(≠Tj)から有向枝（Th,Ti）:x_{ℓ} が存在する事はないので、Tjを一意的に定めることが出来る。）

以上の手続きによってできたグラフは、$\overline{s}$が実現するスケジュール sのTIOグラフTIO(s)である。$\overline{TIO}$(s)がDITSを持てば TIO(s)も DITSをもつことは、手続きから明らかである。□

我々の目的は、直列可能なスケジュールを実現する実スケジュールを出力するアルゴリズムを構成することにある。直列可能なスケジュールを実現するという以上の概念は、1-SR(One-copy serializability) [BeG 83]と基本的に同一のものである。

5. 先読みスケジューラ

集中型データベースシステムの先読みスケジューラについては[IKK 85]等に詳しいが、ここでは、その動作の概要を述べ、8節で提案するアルゴリズムを特徴づけているいくつかの性質を示す。**先読みスケジューラ**(Cautious Scheduler)CS(C)は、全ての直列可能なスケジュールの集合SRのある部分集合Cに対し定義される。

各トランザクションTはシステム到着時に、Tの**読取り集合**（read set:Tが読むデータ項目の集合）と **書込み集合**（write set:Tが更新するデータ項目の集合）を宣言し、登録を行う。登録が済むと Tはステップの実行要求をスケジューラに発することが出来る。各トランザクションから発せられた実行要求に対して、スケジューラは最終的にクラス Cに属する直列可能スケジュールが得られるように、それらの順序を定めて出力する（実行を許可する）。各ステップの実行可能性の判定を、**完成テスト**と呼ぶ。

完成テストは、定理4.1に基づいて なされるが、用いられるクラスCによってはNP-完全になる場合もある。実用的には、完成テストが多項式時間で実行できるクラス Cが重要であって、例えば、C＝WW をあげることが出来る[IKK 85][CaB 80][KaP 85]。このとき、具体的な手順は以下のようになる。

ある時点で 既に出力されたステップの 系列（部分スケジュール）をP、CS(C)に登録されているがまだ実行されていないステップの集合をPEND、判定すべきステップを qとする。このとき、まずPqと PENDに関する**活性TIOグラフ**(active TIO graph)を構成する。活性TIOグラフとは TIOグラフと類似の方法で以下のように定義されるグラフである。既に登録済みのトランザクションTiを節点とし、Pq部分においてTjが XをTiから読取っているならば有向枝（Ti,Tj):Xを付す。Pq部分の無効書込み Wi[X]は追加節点Ti'への有向枝（Ti,Ti'):Xで示される。PEND内の Wi[X]は、追加節点Ti'への有向枝(Ti,Ti'):Xで示される。PEND内のRi[X]は、節点Tiへの端を持たない有向枝で示される（完成テストの手順では無視される）。得られたグラフをATIO(Pq,PEND)と記す。

つぎに、ATIO(Pq,PEND)に 以下の有向枝を加え $ATIO_{ww}$(Pq,PEND)を 構成し、最後に その排

除閉包 （加えるべき 排除枝がなくなるまで、排除規則を繰り返し適用してできるグラフ）を $ATIO^{*}_{ww}(Pq,PEND)$と記す。

（ⅰ）PqにおいてWi[X]がWj[X]に先行するとき枝(Ti,Tj)を付す（ww-制約枝）。

（ⅱ）Wk[X]を Pqにおける Xへの最後の書込み操作とする。このとき、すべての Wj[X]∈PENDに対し枝 (Tk,Tj)を付す（ww-制約枝）。

（ⅲ）Wk[X] を(ⅱ')と同様に定める。すべての Rj[X]∈PEND に対し(Tk,Tj)を付す（wr-制約枝）。

定理5.1[IKK 85] 与えられた PqとPENDに対し、CS(WW)の完成テストは、$ATIO^{*}_{ww}(Pq,PEND)$が有向閉路を持たないとき、かつその時に限り成功する。□

有向グラフが有向閉路を持つかどうかは多項式時間で判定できるので、C＝WW における完成テストは多項式時間で実行できる。完成テストが成功すれば、CS(C)はqの実行を指示し、失敗すれば qの実行を遅延し、遅延待ち行列の最後に入れられる。なお、完成テストが成功した場合は、その都度遅延待ち行列内のすべての操作に完成テストを試みる。その結果、遅延待行列に置かれたステップも最終的に必ず実行される。

完成テストが 多項式時間で実行可能な クラスとしては、他にも WRW [IKK 85]、MWW、MWRW [IKK 86]などが知られており、これらでは、適当に定義された 活性TIOグラフが有向閉路を持たないことが完成テスト成功の必要十分条件となっている。以後、これらのクラスにもとづく先読みスケジューラを仮定して議論を進める。

単一のサイトに用いられる集中型先読みスケジューラは、次の性質を持つ。

（1）システム中でデッドロックを生じない。

（2）ステップの再実行の必要がない。

（3）要求されたステップは、必ず有限時間内で実行される。

6. 分散型データベースシステムモデルの仮定

2節でその概略を述べた DDBMSをここで具体化する。

（1）サイト数＝n。各サイトには1からnまでのサイト番号が割り当てられている。

（2）各サイトaは先読みスケジューラ $CS_a(C)$ によって制御され、同一の機能をもつ。

（3）各サイトはクロックをもち、それに基づいて図6.1のような 時刻印を発行することが出来る。但し、複数のトランザクションに同じ時刻印は与えないものとする。

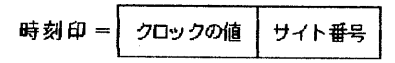

図6.1 時刻印

（4）各サイトは、他のサイトがどの様な実データを持っているかを知っている。

（5）サイト aで受け付けた各実操作は、それが実行されれば有限時間で終了する。実行が終了すると、それが他のサイト bから送られてきたものであるならサイト bにその結果を伝える。

（6）サイトa≠bに対し、$D_a \cap D_b \neq \phi$を許す。

（7）各サイトaの構成は、図6.2の通り。

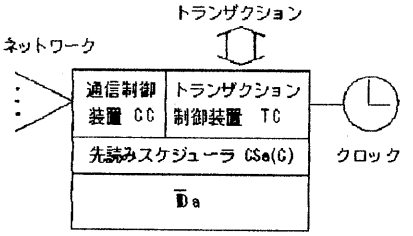

図6.2 各サイトaの構成

7.トランザクションの受付けとその処理

前節のモデルにおいて、ユーザは次のようにトランザクションTiを実行する。

① まず最寄りのサイトへ トランザクションTiをその読取り集合と書込み集合と共に登録する。

② トランザクション Tiの各ステップの実行要求を出す。この時、登録が完了して初めて最初のステップの実行要求を、また前のステップの実行が終了してはじめて、次のステップの実行要求を出すことが出来る。

これに対して、トランザクションTiを受付けたサイト aは、その読取り集合と書込み集合を実データの集合へ次のように置き換える。

（1）Ri[X]は$x_k \in X$なる適当な実データx_k 1個を選び、割り付ける（例えば、$x_k \in X$なる

実データをもつサイトの中で、通信費用最小のものを選ぶ）。

（2）書込みデータXについてはすべての$x_k \in X$を割り付ける。

このようにして得られた実データの集合を$\mathbb{D}(Ti)$とする。あるbに対し$\mathbb{D}(Ti) \subseteq \mathbb{D}_b$の時（a＝bの場合もある）、Tiを局所的なトランザクション、そうでなければ全域的なトランザクションと呼ぶ。なお、このとき実データの集合$\mathbb{D}(Ti)$をうまく定めれば、局所的トランザクションと成り得るものに付いては、必ずその様な選択を行うものとする（局所的トランザクションは並行処理上有利であるため）。その後、ユーザからの①、②の要求に対してシステムは次節のように動作する。

8.DDBMSの並行処理制御アルゴリズム

A．各サイトaにおけるトランザクション制御装置TCの動作

A-1 新しいトランザクションTiの受付け：

TCは7節のような 実データへの割り付けを行った後、以下の動作を行う。

（1）局所的なトランザクションの場合：

$\mathbb{D}(Ti) \subseteq \mathbb{D}_b$を満たすサイトbの$CS_b(C)$にTiを登録する。

（2）全域的なトランザクションの場合：

（2-1）Tiにサイトaの時刻印 t'を与え、関係するサイトへTiと共に t'を送り、登録可能性を問い合わせる。

（2-2）関係する 全サイトからの 返答のなかで最大の時刻印を見つける。それを t_{max}とすると、t_{max}をTiの登録時刻印と 定める。この時点で、サイト aの クロックの 値が t_{max}による値より小さければ、そこまでクロックを進める。

A-2 トランザクションからのステップの実行要求の受付け：

先に定めた実データへの割付に従って、実操作への変換を行った後以下の動作を行う。

（1）局所的なトランザクションの場合：

そのステップの実操作のすべてを関係するサイトbへ送る。

（2）全域的なトランザクションの場合：

そのステップの実操作それぞれを関係するサイトへ送る。この時、それがそのサイトへ送る初めての実操作であるならば、t_{max}を共に送る。

A-3 他のサイト bからのトランザクションTiの登録可能性の問い合わせの受付け：

自己のサイトaのクロックによる時刻印tとTiに付与された時刻印t'を比べる。

（1）t'＞tのとき： 自己のサイトのクロックを t'による値まで進め、実行可能であると返答し、$CS_a(C)$に(Ti,t')で仮登録。

（2）t'＜tのとき： 自己の $CS_a(C)$に t'で登録可能かどうか 問い合わせる。可能ならその旨返答し、(Ti,t')で$CS_a(C)$に仮登録する。不可能ならt以上の時刻印であれば可能であると 返答し、(Ti,t)で$CS_a(C)$に仮登録する。

A-4 他のサイトからの仮登録済みトランザクションTi の 実操作の 実行要求の受付け：

実操作と共に時刻印t_{max}が送られてくれば (Ti,t_{max})で本登録し、自己のサイトの$CS_a(C)$に渡す。実操作のみ送られてくれば本登録は済んでいるので、そのまま$CS_a(C)$に渡す。

B．先読みスケジューラ$CS_a(C)$の動作

基本的には 集中型データベースの 場合と同一。但し$CS_a(C)$が処理しているトランザクションのうち、全域的なトランザクション (Ti,t')と(Tj,t'')に対しては、t'<t''ならば、TiをTjの前に直列化するという制約を付した上で（具体的には $CS_a(C)$が保持する活性TIOグラフにおいて節点TiからTjへ制約枝を付す）、次のように扱う。

B-1 トランザクション(Ti,t')の （他サイトあるいは自己のサイトからの）登録要求の処理：

Tiの実操作のうち自己のサイト aに関するものを、$CS_a(C)$ の PENDに入れる。このとき、活性TIOグラフに 有向閉路ができなければ(Ti,t') のままで仮登録しておく。そうでなければ、時刻印t'での登録を拒否し、サイト aの現在のクロックによる時刻印tを用いて、(Ti,t)を仮登録する。

B-2 実操作qの実行要求の受け付け：

先読みスケジューラ$CS_a(C)$の通常の手順に従い、４節で述べた完成テストを qに関

して行い活性TIO(s)グラフに有向閉路ができなければ、q を実行する。そうでなければ、即時の実行を拒否し qを遅延操作の待ち行列の最後に入れる。

9. アルゴリズムの正当性

8節の手順を6、7節で定義した分散型データベースのモデルに適用したとき、システムはデッドロックに陥ることなく直列可能なスケジュールを実現することを示す。

9.1 直列可能なスケジュールsの実現について

上述の DDBMSで 実現される 実スケジュール $\overline{S}$とは、システム内で実行されたすべての実操作をそれが実行された絶対時刻に従って並べた系列で あると 考えられる。このとき、複数の（異なるサイトで実行された）実操作が、同じ時刻の所に 並ぶならば、それらの順序は 任意に定めて一列に並べる。これは、分散型データベースシステムを、全ユーザから 1つの データベースシステムとして ながめることに 相当する。この実スケジュール $\overline{S}$中の全ての 実読取り操作 $\overline{R}i[x_k]$に 対して2節の定義に従って、$RF_{\overline{S}}(\overline{R}i[x_k])$ の値を 定めることができる。同時刻に 実行された複数の 実操作は、すべて異なるサイトで 実行されたもので あることから、このように 定められた RFの値は、各サイトaにおける$CS_a(C)$が定めた値に一致する。RFが定まると、$\overline{S}$に対する$\overline{TIO}(s)$を描くことができる。この$\overline{TIO}(s)$がDITSを持つことを示せば、上述のアルゴリズムが出力する実スケジュールは、直列可能なスケジュールを実現する事がいえる。

定理9.1 8節のアルゴリズムによって 制御されている分散型データベースシステムが実スケジュール $\overline{S}$を出力したとすれば、その実TIOグラフ$\overline{TIO}(s)$は、T_gを最初としT_fを最後とするDITSをもつ。

証明 上述の議論から、$\overline{TIO}(s)$は、各サイトaの実TIOグラフ$\overline{TIO_a}(s_a)$のすべてを、各全域的トランザクション Tiの（いくつかのサイトに 存在する）節点を一つにまとめつつ重ね合わせることで実現されることがわかる。ただし、$\overline{S}_a$はサイト aの実スケジュールである。各サイトaの $\overline{TIO_a}(s_a)$は$CS_a(C)$の動作からT_gを最初に、T_fを最後に置くDITSをもつことが保証されている。これらのDITSと矛盾せず、かつ全域的トランザクションの順序は時刻印にしたがって並ぶように、$\overline{TIO}(s)$の全節点上に全順序≪を定めよう。これが可能なことは、

(1) 全域的な トランザクション間には 時刻印による順序が全サイト共通に定められていて、各サイト aでは、a に関与する全域的トランザクション間に、時刻印による制約枝を付してスケジュールしていることから、その順序に矛盾しないことが保証されていること、

および

(2) 各局所的トランザクションは一つのサイトにのみ登場すること、

を考えると明らかである。この≪がT_gを最初に、T_fを最後におくことも明らかである。そこで≪が$\overline{TIO}(s)$のDITSであることを示そう。

定理4.2の前に示したDITSの性質(1)は明らかに成立する。なぜなら、$\overline{TIO}(s)$の枝は時刻印による制約枝の一部を除いて全てどれかの$\overline{TIO_a}(s_a)$に存在していたものであり、≪はその方向に矛盾しないように定められている。また、時刻印による制約枝は、上述のように必ず左から右へ向かうことが保証されているから、これも問題ない。

つぎに、DITS の性質(2)を示すために、$\overline{TIO}(s)$ 内の有向枝(Tg,Th):x_aと(Ti,Tj):x_bを 考える。ただし、$x_a, x_b \in X$ かつx_a,(x_b)は サイトa（サイトb）の実データとする。a＝bであれば、これら2本の有向枝は$\overline{TIO_a}(s_a)$に存在し、$\overline{TIO_a}(s_a)$のDITSに矛盾しないように≪が定められていることから、性質(2)が示せる。a≠b であれば、書込み集合の実データへの割り付け方法（7節参照）からわかるように、データ項目 Xへの書込みを行っているTgとTiは全域的トランザクションであり、どちらもサイト aとサイト bの両方に関係する。従って、$\overline{TIO_a}(s_a)$と$\overline{TIO_b}(s_b)$のどちらにおいてもTgとTiの間には時刻印による制約枝がすでに付されている。一般性を失う事なく制約枝は（Tg,Ti)であるとしよう。すなわち、$\overline{TIO}(s)$においてTgからTjへ有向路が存在し、Tg≪Tjである。さて、Tg とTiの両節点は $\overline{TIO_a}(s_a)$に存在するから、DITS の性質(2)より、$\overline{TIO_a}(s_a)$ではTg,Th,Tiの順に左から右へ並んでいる。同様に$\overline{TIO_b}(s_b)$ではTg,Ti,Tjの順に左から右へ並んでいる。 従って、≪はこの 順序に矛盾しない

ので、(Tg≪) Th≪Ti (≪Tj) となり、$\overline{TIO}$(s) において 性質(2)が成立する。□

9.2 デッドロックについて

各サイト個々の動作については、先読みスケジューラの性質として、デッドロックが生じないことが示されているので省略する。ここではシステム全体としてのデッドロックのみを考える。

いま、トランザクションTiと Tj が Ti=0i1 ... 0ir...0iq...、Tj=0j1...0jp...0js...であるとする。0jk 等は実操作を表す。このとき、サイトa において 0jpが 0iqの実行を待っていたとする。すなわち、サイトa の先読みスケジューラが 0jpの実行の前に 0iqの実行を要求した。(換言すると、0jpの前に0iqを置かないと、先読みスケジューラの完成テストを満足しない)しかし、Tiは0iqの 前に位置する0ir (つまりr<q) の実行をサイトbで要求中であり、それが済まなければ 0iqの実行要求を出せない。そして、サイトb ではこの 0irが何等かの実操作の実行を待ち、その実操作はまた別の実操作の実行を待つというふうにしてその先頭が 0jsであったとする。ここで、p<sを仮定すると、Tjの 0jsは0jpが実行されないと実行できず、結局、図9.1に 示すようにこれらの実操作のそれぞれは自分自身の実行を待っていることになり、デッドロックを生じている。確かに、一般の先読みスケジューラでは、このような状況が考えられ、デッドロックが生じることがある。

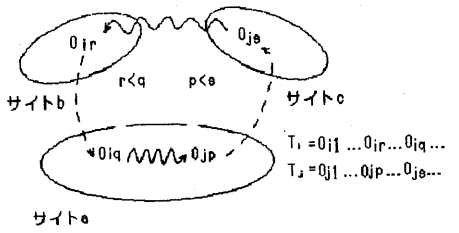

図9.1 システム全体にかかわるデッドロック

ところで、一般に先読みスケジューラでは、ユーザが宣言した書込み集合や読取り集合の一部を途中で取り消すと、スケジューラ自体がデッドロックに陥るという取り消し異常を引き起こすことがある。そこで、実用的な先読みスケジューラとして、取り消し異常を生じないものが提案されており、先に言及したCS (WW)、$\overline{CS}$ (WRW)、 CS_m (MWW)、CS_m (MWRW) 等は、そのような性質を持っている。これらの、先読みスケジューラでは、補題9.1に 示すように(Ti, t')、(Tj,t")、t'<t" を考えたとき、Tiの実操作0iはTjの実操作0jの実行を決して待たないことがいえる。従って、各サイトに 補題9.1の条件を満たす先読みスケジューラを用いれば、時刻印の小さなトランザクションの実操作がより大きな時刻印を持つトランザクションの実操作の実行を待つことはなく、デッドロックを生じないこと (定理 9.2) が証明できる。

補題9.1 Pを現在までにスケジューラCS(C)によって、出力された部分スケジュール、また、現在のATIOグラフにおけるDITSの順に従って、PEND中のステップをトランザクション毎に直列に 並べた系列を s'とする。このとき、Ps'∈C が常に成立するような先読みスケジューラでは、(Ti,t')、(Tj,t")、t'<t"なる２つの全域的トランザクションを 考えたとき、Tiの ステップ 0iはTjのステップ0jの実行を待つことはない。

証明 TiからTjへ時刻印による制約枝が付されているので、ATIOグラフのどのようなDITSにおいても、Ti≪Tjが成立する。従って、s'では 0i はOjの前に 位置する。Ps'∈Cの性質によって、先読みスケジューラはs'の順序で実操作を処理でき、これは0iが0jの実行を待たないことを意味する。□

この補題9.1の性質 Ps'∈Cは、すべての先読みスケジューラで 成立するわけでは ないが、上記のCS(WW)、$\overline{CS}$(WRW)、CS_m(MWW)、CS_m(MWRW) 等では成立することが 知られている [IKK 85] [IKK 86]。

定理9.2 各サイトに補題9.1の条件を満足する先読みスケジューラを用いるとき、システムにデッドロックは生じない。

証明 上述の様なデッドロックが生じたとすれば、閉路中に存在する全域的トランザクション(Ti,t')と(Tj,t")で、t'<t"にもかかわらず、TiがTjを待つ (Tiの実操作がTjの実操作の実行を待つ) という事態が必ず起こっている。これは、補題9.1に示した スケジューラの動作に矛盾する。□

9.3 時刻印の付与について

同一時刻印が、複数のトランザクションへ割当てられることはない。しかし、同一トランザクションへ複数の時刻印が割当てられることは起こり得る。この状態は、仮登録の手続き中に生じる。しかし、実行時（本登録）の時刻印はそのトランザクションを最初に受け付けたサイトにおいて決定され、各サイトでの実操作実行時にはその時刻印が知らされるので、実操作実行時の時刻印は各サイトで同一である。

各サイトaの $CS_a(C)$は、トランザクションTiの登録可能性を問われたとき、サイト aのその時点でのクロックによる時刻印をtとすれば、必ず(Ti,t)を登録する事ができる。これはTiを受付けたサイトでも同様である。なぜなら、サイトaにおけるクロックの修正法(A-1(2-2),A-3(1))から、時刻印 tを発行したとき、$CS_a(C)$内の全てのトランザクションは tより小さな時刻印を持つので、$CS_a(C)$内では、Tiを最後の位置に直列化するようなDITSが必ず存在するからである。既に登録されているトランザクションの実操作を全て実行してから、Tiの実操作を実行すれば必ず直列可能なスケジュールを実現する実スケジュールを出力できるからである（ただし、実際にその様にスケジュールするとは限らない）。

次に、t'なる時刻印を持つ全域的なトランザクションTiを既に仮登録している$CS_a(C)$に対し、時刻印を t（＞t'）に修正し、本登録を要求することがある (A-1(2-2), A-2(2))。このとき $CS_a(C)$は (Ti,t) を受理する事を示す。

定理9.3 $CS_a(C)$ として CS(WW)、$\overline{CS}$(WRW)、CS_m(MWW)、CS_m(MWRW)を 考えるとき、t'なる時刻印を持つ全域的なトランザクションTiを既に受理している$CS_a(C)$に対し、時刻印をt（＞t'）に 修正し、本登録を 要求するとき、$CS_a(C)$は (Ti,t) を受理する。

証明 最初に、これらのスケジューラにおける完成テストの方法を示す。

５節に CS_a(WW)の場合を述べたが、他の場合も含めて、ATIOグラフの 排除閉包 ATIO*c(Pq,PEND)が 閉路を持たないことで完成テストの結果を判定できる。

このようなテストにおいて、(Ti,t')の (Ti,t)への 変更を考える。ただし、t＞t'である。時刻印の変更によって、全域的トランザクション Tgとの順序が Ti≪Tgから、Tg≪Tiになったとする。時刻印変更前の状態を、図9.2(a)に示す。ただし、TiとTgはともに実データx_mへの書込み操作を持つ（その様なx_mがなければ、証明は簡単）。Tiの時刻印が変更されたことは、その実操作がどれも未実行であることを意味する。また、ATIO*c(Pq,PEND)に 閉路が存在しないことから、Tgは $\overline{W}g[x_m]$を 未実行（C=WWの場合）、あるいは実行済みであるが、それを読んでいるものは ない（C=WRWの場合）。すなわち、Tgから出ているx_mの枝は追加節点に終わっている。

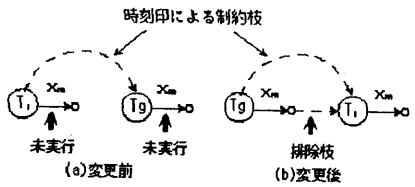

図9.2 (Ti,t')から(Ti,t)への変更

時刻印をtへ変更すると、 Tiから出る枝は、追加節点へのものと時刻印によるもののみであるから、変更以前に比べると減少する。Tiへ入る枝は (Tg,Ti)（時刻印による制約枝）が増える。その結果、図9.2(b)に示すように、排除枝でTiに入るものが出来ることがある。しかし、同図に示したように、この排除枝はTgの追加節点からのものであるので、TgからTiへの経路が出来たにすぎず、制約枝 (Tg,Ti)と同じ効果を与えるだけである。 tへの時刻印変更によってATIO*c(Pq,PEND)に 閉路が出来たとする。上の議論から、この閉路はTiに入る時刻印による制約枝とTiから出る時刻印による制約枝を含んでいる。そこで、この閉路中、Tiの前にTg、後ろにThがあるとする。つまり閉路は時刻印による制約枝(Tg,Ti)と(Ti,Th)を持つ。しかし、これは (Tg,Th)が、Tiの時刻印の変更以前から存在していた（つまり閉路が存在していた）ことになり矛盾が生じる。よって、$CS_a(C)$は(Ti,t)を必ず受理する。□

10. むすび

本報告ではサイト間にデータの重複がある場合について、分散型データベースシステムモデルの構築、そこでのスケジュールの直列可能性

の意味づけ、およびトランザクションの並行処理制御アルゴリズムの提案を行った。

本稿の冒頭で述べたように、データのコピーを複数のサイトにおく目的として、参照主体のトランザクションが多数生ずるような環境の下での効率向上、サイト故障からのデータの保護等を上げることが出来る。これらのに対して、ここで提案したアルゴリズムは

(1) 全域的なトランザクションが生起しない場合、各サイトは独立に集中型の場合とほぼ同様の効率で動作する、

(2) デッドロックを生じないため全域的なトランザクションの介入による効率の低下が比較的少ない、

といった望ましい特徴を有する。

今後、スケジュールの効率評価や障害時における対処、回復アルゴリズムを含めて、実用システムへの適用性の検討が必要と思われる。

謝辞

本研究を行うに当たり、熱心に御討論頂いた山下雅史助教授（広島大）、滝沢誠助教授（東京電機大）、橋口攻三郎助教授（豊橋技科大）、今井正治講師（豊橋技科大）、平田富夫講師（名大）に深謝します。なお、本研究は一部文部省科学研究費によるものである。

参考文献

[BeG 81] P.A.Bernstein, N.Goodman: Concurrency control in distributed database systems, ACM Computing Surveys, vol.13 No.2, June 1981, pp.185-224.

[BeG 82] P.A.Bernstein, N.Goodman: Multiversion concurrency control: Theory and Algorithms, ACM TODS, vol.8 No.4, December 1983, pp.465-483.

[CaB 80] M.A.Casanova, P.A.Bernstein: General purpose schedulers for database systems, Acta Informatica 14, 195-220, 1980.

[CeP 85] S.Ceri, G.Pelagatti: Distributed Databases Principles & Systems, McGraw-Hill International Student Editions, 1985.

[Hal 85] 原嶋秀次，茨木俊秀： Cautious Scheduler（先読みスケジューラ）を用いた分散型データベースシステムの並行処理制御， 情報処理学会データベース研究会 1985年7月.

[IKK 85] T.Ibaraki, T.Kameda, N.Katoh: Cautious transaction schedulers for database concurrency control, Technical Report LCCR TR85-6, Simon Fraser University, 1985.

[IKK 86] T.Ibaraki, T.Kameda, N.Katoh: Multiversion cautious schedulers for database concurrency control, Technical Report LCCR TR86-2, Simon Fraser University, 1986.

[IKM 83] T.Ibaraki, T.Kameda, T.Minoura: Disjoint-interval topological sort: A useful concept in serializability theory, Proc. 9th International Conf. on VLDB, Florence Italy, 89-91, Oct/Nov 1983.

[KaP 85] P.C.Kanellakis, C.H.Papadimitriou: The complexity of distributed concurrency control, SIAM J.Comput, Vol.14, No.1, Februrary 1985.

[Koh 81] W.H.Kohler : A survey of techniques for synchronization and recovery in decentralized computer systems, ACM Computing Surveys, vol.13 No.2, June 1981, pp.149-183.