数理解析研究所講究録
第494巻 1983年 90-101

基本操作の依存性を利用した並行処理のための手法

京大工学部　近藤誠一 (Seiichi Kondo)
京大工学部　上林弥彦 (Yahiko Kambayashi)
京大工学部　矢島脩三 (Shuzo Yajima)

1. まえがき

データベースシステムの大きな目的のひとつは、データを多くの利用者が共有することにより、冗長度を減少させ、統一的な管理を可能にすることであると言える。この場合、複数の利用者の要求を並行処理することは、システムの効率向上のためには不可欠であると考えられる。本稿では、基本操作の依存性を考慮したトランザクションの等価変換について述べるとともに、それを利用した並行処理のための手法について考察する。

データベースに加えられる各トランザクション T_i は、$t_{i1}, t_{i2}, \cdots, t_{in_i}$ という基本操作の列で与えられる。複数のトランザクションを並行処理する場合、各トランザクションを構成する基本操作の列の間に他のトランザクションの基本操作の入った形の並行処理スケジュールが作られる。この場合、デッドロックによる処理の停止がないことと、直列可能性を

保証させることが重要な問題である。

従来、並行処理の問題を考える際、トランザクションの基本操作の実行順は利用者によって与えられたものであり、この順序を変換するという問題については考慮されていなかった。本稿では、基本操作の依存性を利用したトランザクションの等価性について考察する。また、この等価変換により、トランザクションの融通性が増加するため、並行処理に対しても、その処理効率向上に利用することができる。

2節では、並行処理に関する基本的事項について、3節では、基本操作の依存性を考慮したトランザクションの等価変換について述べる。また、4節では、その並行処理への応用について述べる。

2. 基本的事項

2.1. デッドロックと直列可能性

次のように記号 $\rightarrow$ を定める。

$T_i \xrightarrow{x_k} T_j$ ・・・トランザクションT_jが、項目x_kにロックをかけているとき、トランザクションT_iが、項目x_kに対するロックを待っている状態

$T_i \rightarrow T_j$ ・・・ $\exists x_k [T_i \xrightarrow{x_k} T_j]$

関係 $\rightarrow$ がサイクルを持つとき、デッドロックが生じているという。

個々のトランザクションが正当であっても、並行処理した場合、意味的な誤りが生じることがある。そこで、個々のトランザクションを独立にある順序で直列に処理した結果と等しい結果を得るとき、正当であるとする。これは、直列可能性と呼ばれているものである。次のように記号 $\Rightarrow$ を定める。

$T_i \overset{x_k}{\Rightarrow} T_j$ … トランザクションT_iが項目x_kのロックを解除した後、T_jが最初にロックをかける。

$T_i \Rightarrow T_j$ … $\exists x_k [\, T_i \overset{x_k}{\Rightarrow} T_j \,]$

関係 $\Rightarrow$ がサイクルを持つとき、そのスケジュールは、直列可能でないという。

ロックのモードとして、排他ロック、共有ロックのように二種類以上定めているものもあるが、結果は容易に一般化できるので、本稿では、ロックのモードとしては、排他ロック一種類のみであるとする。

2.2. 木プロトコル　デッドロックの回避と、直列可能性を保証するプロトコルとして、木プロトコルがある。木プロトコルは、各トランザクションがロックをかけていくデータの順を定める木によって定義される。木の節点には、変数名が対応し、ロックは次の順に行な

```
LOCK   A
LOCK   B
LOCK   C
UNLOCK B
LOCK   E
UNLOCK A
UNLOCK C
UNLOCK E
```

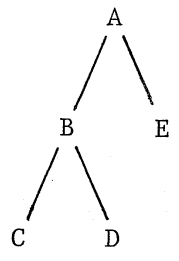

図1. 木プロトコルの例

われる。

(1) 最初は任意の節点にロックをかけることができる。

(2) その後、ロックをかけられた節点の子節点にのみロックをかけることができる。

(3) ロックの解除は、任意にできる。

図1にその例を示す。

3. 基本操作の依存性を考慮したトランザクションの等価変換　従来の並行処理制御の場合、トランザクションは利用者によって与えられた基本操作の列であり、その順序は固定されたものとしていた。ここでは、基本操作の依存性について考察する。

命令の依存関係を示す有向グラフを次のように定義する。節点は、各命令を示す。命令 tin が、命令 tim に依存しているとき、すなわち、命令 tin の実行のためには、命令 tim の終了が必要なとき、節点 tim から節点 tin へ有向枝をつける。たとえば、代入命令の場合、その代入すべき値の元になるデータを読み込む命令や計算する命令は、それに先がけて実行することが必要である。

[例] 図2のようなトランザクションを考える。データベースからの Read 命令は、ローカル変数への代入を Write 命令は、ローカル変数から、データベースへの代入を意味する。

l_iは、ローカル変数、x_jはデータベース内の変数を、t_{ik}は、各基本操作を示すものとする。

t_{11} : Read l_1 := x_1
t_{12} : Read l_2 := x_2
t_{13} : Read l_3 := x_3
t_{14} : Read l_4 := x_4
t_{15} : l_5 := $f_1(l_1,l_2)$
t_{16} : l_6 := $f_2(l_2,l_3)$
t_{17} : l_7 := $f_3(l_4)$
t_{18} : Write x_1 := l_5
t_{19} : Write x_3 := l_6
t_{110}: Write x_5 := l_7

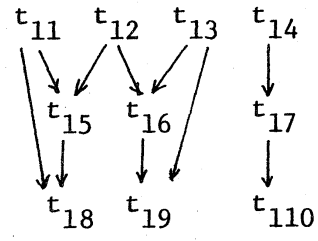

図2

これらの半順序に反しないような命令列は、すべて等価であることが、容易に示される。しかし、バッチ的なトランザクション（開始時にすべての命令が決定しているもの、すなわち問合わせは、唯一度だけであるもの）でしかも、静的なトランザクション（読むべき項目、書き込むべき位置が、トランザクション開始時にすべて決定することができるもの）のときのみ、トランザクション開始時に基本操作の依存関係を示すグラフを決定することができ、それを元にして、等価変換を行なうことができる。

4. 並行処理への応用

4.1. ロールバックへの応用

ロールバックする際、すべての書き込んだ値を前の値にもどし、はじめからやり直すといった方法が一般的であるが、依存関係を用いた等価変換を用いると部分的にロールバックすれば良いことがわかる。すなわち、原因となるデータに関する命令と、それに依存す

る命令にのみ、注目すればよい。この場合、実際に行なった操作を元にするので、動的なトランザクション、オンライン的なトランザクションでも問題にならない。

※ $t_{31} : l_1 := x_1$
$t_{32} : l_2 := x_2$
$t_{33} : l_4 := x_4$
※ $t_{34} : x_4 := f_1(l_1)$
$t_{35} : x_2 := f_2(l_2)$
→ $t_{36} : x_3 := f_3(l_2)$
$t_{37} : x_1 := f_4(l_1, l_4)$

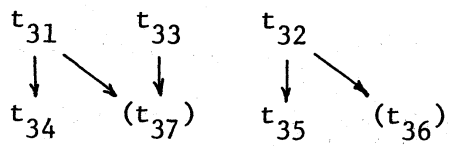

図 3

[例] 図3のトランザクションについて考える。t_{36}の操作の前にデッドロックが生じていることがわかり、x_1が原因となっているものとする。このとき、t_{31}とその子孫であるt_{34}のみをロールバックすればよいことがわかる。その結果、トランザクションは、"$t_{32}\ t_{33}\ t_{35}\ t_{31}\ t_{34}\ t_{36}\ t_{37}$"という順序に等価変換されたことになる。

以下のように処理を進める。

- 実行に従い、ログとともに、命令の依存関係を示すグラフを構成していく。
- ロールバックしなければならない原因をつくった命令及び命令の依存関係を示すグラフの上で、その子孫となる命令のうち、Write命令で書き換えた値を元にもどし、半順序に反しないように、関連する命令のみを再実行する。

4.2. 等価変換を利用したプロトコル　　基本操作の依存

関係を示す有向グラフを次の手順に従って変数の依存関係を示す対の集合に変換する。

①基本操作を示す節点をそこで使用しているデータベース内の変数で置き換える。もし、ローカル変数のみの命令のときは、'o'で置き換える。

②同じ変数が2回出現している場合は、最初が読み込み命令、後が書き込み命令なる矢印がついているが、後の方の節点を消去する。同時にその節点に向かう矢印も消去する。

③'o'の節点を消去する。ある'o'の親節点のすべてから、子節点のすべてに向かう矢印を加える。

④推移律によって導くことができる矢印を消去する。

⑤各矢印について、親を左に、子を右にする対をつくる。

図4にその例を示す。

トランザクションT_iで用いられるすべての変数の集合をX_iとする。また上で求めた変数の依存関係を示す対の集合をP_iとする。このP_iに矛盾しないようなホプロトコルであれば、

T_7: t_{71} : Read l_1 := x_1
t_{72} : Read l_2 := x_2
t_{73} : l_3 := $f_1(l_1, l_2)$
t_{74} : Write x_1 := l_3
t_{75} : Write x_3 := l_3

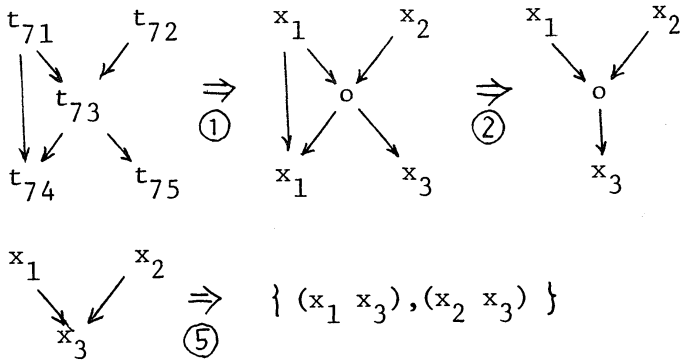

図4

このトランザクションは効率良く処理できる。この対の集合を用いた並行処理のための手法を次の2節で述べる。

4.3. 木の動的作成　　ここでは、トランザクションは、静的であり、かつ、バッチ的なトランザクションに限ることにする。すなわち、トランザクション開始時に順序の対 P_i が決定できるものとする。

次の順に処理を進める。有向グラフとして、全体の状況を示すGと、個々のトランザクションを示す g_i を用いる。

① X_i と P_i を求める。

② Gの節点の集合と X_i との交わりの節点は、Gにおける共通祖先から順序を決定する。P_i の順に反するように、ロックをかける必要性が生じる場合もある。

③ それ以外の節点については、P_i に反しないように任意に順を付けることができる。

②③により、もとのトランザクションを等価変換する。このとき、もとの順序を基本にして、それを変更する形で行なう。

④ 変換したトランザクションにより g_i を構成する。これは、木プロトコルの規則に反しないように構成する。

⑤ g_i をGに組み込む。

⑥ トランザクションを実行する。

[例] 元のトランザクションは、$x_1\ x_2\ x_3\ x_4\ x_5$ の順に参照

するものとする。

$X_8 = \{x_1, x_2, x_3, x_4, x_5\}$

$P_8 = \{(x_1\ x_2), (x_2\ x_3)\}$ とする。

現在Gは、図5のようになっているものとする。②の条件により、X_8とGとの交わりである$\{x_2, x_3, x_5\}$については、その取得順は$x_3x_2x_5$となる。このときP_8と矛盾するので、x_3はまずロックをかけるだけで実際の操作は、x_2の操作終了後となる。③により、x_1とx_4の実行順を決定する。この場合もとのトランザクションに最も近い$x_1x_3x_2x_4x_5$の順が適当となる。この後、g_iを決定するが、これは、ロックを解除する時点によって変わる。たとえば図6のようになる。⑤により、図7のようにGを変更する。

トランザクションの等価性を利用した順序づけを行なうとその分、融通性が増し、より木に適合したものが得られる。その反面、次のような欠点がある。

・有向グラフGの参照、変更に対するコストが問題となる。

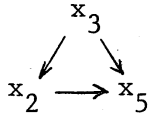

図5

```
LOCK   x1
LOCK   x3
UNLOCK x1
LOCK   x2
LOCK   x4
UNLOCK x2
UNLOCK x4
LOCK   x5
UNLOCK x5
UNLOCK x3
```

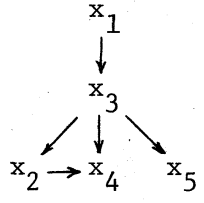

図6

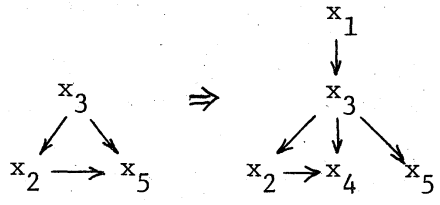

図7

しかも、Gの参照、変更は並行処理することができない。

・静的でありかつバッチ的であるトランザクションに限る。

4.4. トランザクションの定形的性質を利用した木の作成

ここでは、オンライン処理を含めたトランザクションを対象とするが、各トランザクションは、その参照するデータの順序がある程度固定しているものを扱う。4.6節では、その都度、Gを変更していくが、ここでは、統計情報によりGを決定し、Gの変更は、挿入，削除があった場合を除いて、大きな周期で変更するものとする。

{$T_1, T_2, \cdots T_n$}を典型的なトランザクションの集合とする．各トランザクションT_iの出現頻度をr_iとする。また、P_iに含まれる変数の集合をV_iとする。以下の手順で木を構成する。

①すべてのP_iをもとにして、重み付き有向グラフをつくる。
重みはその対の出現頻度を示す。すなわち、そのトランザクションの出現頻度を示す。

②重み付き有向グラフを正規化する。すなわち、各矢印についてその重みの合計をとる。逆方向の矢印がある場合は、その重みの差をとり、小さくない方の矢印を採用することにより．重みはすべて負でない数になるようにする。また、矢印がない場合も、すべて重み0の矢印があるものと仮定

する。

③この重み付き有向グラフにおいて、重みの合計が最大となる全域木を求める。ただし、ここでは、重みの合計は次のようにして求める。

- 親子関係が一致する場合、また、祖先と子孫の関係が一致する場合は、和をとる。
- 親子関係が一致しない場合、また、祖先と子孫の関係が一致しない場合は、差をとる。

④$(\bigcup_{i=1}^{n} X_i - \bigcup_{i=1}^{n} V_i)$に含まれる変数は、その順序は何ら規定されないので、適当な位置に加えられる。たとえば、最も出現頻度の高いトランザクションの元の順序に従うように加える。

⑤典型的なトランザクションでないものの場合は、ロールバックの手法を繰り返しながら、結果として木に従うように、実行を進めていく。

図8にその例を示す。

5. おわりに　　4節での手法では、各項目にアクセスする最初の命令の順にのみ着目したが、読みのみ、書きのみの場合は、ロックをかけるのは一時的で良いのに対して、読み書きする場合は、ある程度の時間、ロックを持続することが必要となる。このような命令をも考慮したスケジューリング

に拡張することができると考えられる。

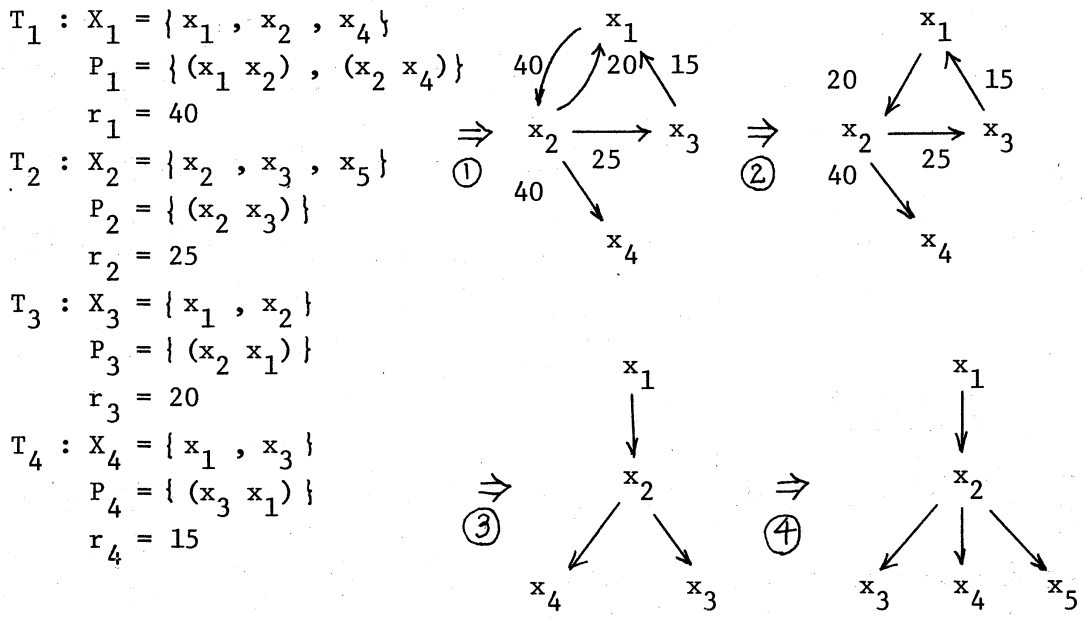

図 8

参考文献

[ESWAG76] Eswaran,K.P., Gray,J.N., Lorie,R.A. and Traiger,.I.L. "The Notion of Concurrency and Predicate Locks in a Data Base System" CACM Vol.19,No.11 (1976)

[FUSSK81] Fussell,D., Kedem,Z. and Silberschatz,A. "Deadlock Removal Using Partial Rollback in Database Systems" SIGMOD(1981)

[KEDES79] Kedem,Z. and Silberschatz,A. "Controlling Concurrency Using Protocols" JACM (1979)

[KEDEM82] Kedem,Z., Moham,.C. and Silberschatz,A. "An Efficient Deadlock Removal Scheme for Non-Two-Phase Locking protocols" VLDB (1982)

[SILBK80] Silberschatz,A. and Kedem,Z. "Consistency in Hierarchical Database System" JACM (1980)

[ULLM 80] Ullman,J.D. "Principles of Database Systems" (1980)

[YANNP79] Yannakakis,M., Papadimitriou,C.H. and Kung,H.T. "Locking Policies : Safety and Fredom from Deadlock" FOCS(1979)

[YANN 82] Yannakakis,M. "Freedom from Deadlock of Safe Locking Policies" SIAM J.COMPUT. Vol.11,No.2 May (1982)