3Q-11

トランザクションの並行実行での後戻りデッドロック

宮島　勝己　　滝沢　誠

(東京電機大学理工学部)

1. はじめに

本論文では、複数の異種データベースシステム(DBS)に対する共通なデータ操作言語としてPrologを考え、分散型トランザクションをPrologで記述し、複数のDBSで並行実行する方法について考える。このとき、並行実行について、導出の前進時に、通信についてデッドロックを防止する方法は既に[TAK87]で論じてあるので、本論文では、後戻り時のデッドロック状態を検出し、解除する方法について論じる。

2. Prologトランザクション

本論文では、各DBSを、Prologの具象単位節の集合として事実ベース(FB)と、それらの操作演算の集合から成る事実ベースシステム(FBS)として仮想化し、これに対するトランザクションをPrologにより記述する方法を与える。まず、トランザクションを書くために、新たに述語記号Begin,Commit,Abort, Writeを導入し、それらの手続き的意味を以下に示す。

[手続き的意味]

(1)基本式Beginが選択されるとき、トランザクションの開始を意味する。

(2)基本式Commitが選択されるとき、トランザクションは終了を意味する。このとき、更新結果は永続され、他のトランザクションにより参照でき、FBSが障害を被っても生き延びる。その後、Beginの選択からこのCommitまで導出を後戻りできない。

(3)基本式Abortが選択されるとき導出は失敗し、BeginからAbortまでの更新結果は、FBから消去される。

(4)基本式Writeが選択されるとき、事実識別子と項が束縛されているか未束縛かによって、新しい事実を追加、置き換え、消去する。

3. 意味のある順序

Prologトランザクションを並行実行するために基本式の系列(<)の中から意味のある順序関係(<<)として、競合、流関係を定義する。ここで、基本式Aの後に基本式Bが現れるとする。

[定義]トランザクションTとは、基本式を全順序付けた集合T=(AA,<)である。AAは、基本式の集合$\{A_1,\ldots,A_m\}$を示し、<はA_kの全順序関係を示す。■

[定義]異なる基本式AとBが同一の事実に対する基本式で、どちらか一方がWrite基本式のとき、AとBは競合(A→B)するという。■

[定義]基本式AとBが変数を共有し、この変数はAより左の基本式に含まれないとき、AからBに流関係(A=>B)があるという。■

この二種の関係を保存するようにトランザクションを分割し、並行実行する。

Backward Deadlock in Concurrent Execution of Prolog
Katsumi MIYAJIMA, Makoto TAKIZAWA
Tokyo Denki University

4　Prologトランザクションの分割

Prologトランザクションを分割し、複数のFBSで並行実行する方法について考える。

4．1トランザクショングラフ

トランザクションの基本式の系列の中から、意味のある順序関係を表すためにトランザクショングラフ(TG)を考える。

[定義]トランザクションの基本式をノード、基本式内の競合と流関係を有向辺で示したグラフがトランザクショングラフである。■

以下のトランザクションのTGの例を図1に示す。

```
T?-Begin,AC1(A1,N,C1),AC3(A3,N,C3),
   Write(AC1,A1,N,C1/2+C3/3),AC2(A2,N,C2),
   Write(AC2,A2,N,C1/2+C2+C3/3),
   Write(AC3,A3,N,C3/3),Commit.
```

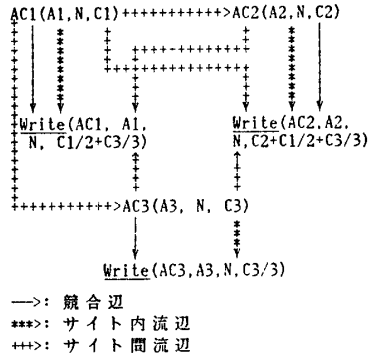

図1　トランザクショングラフ

4．2トランザクショングラフの分割

トランザクションを並行実行するためには、副トランザクション間で通信が必要となる。ここで、新たに通信述語SendとReceiveを導入し、それらの手続き的意味を示す。

[通信述語の手続き的意味]

(1)基本式Send(X,M)が選ばれたとき、導出は常に成功し、変数Mは変数Xで束縛される。

(2)基本式Receive(M,X)が選ばれたとき、導出は項Mが束縛されるまで遅延する。束縛されると導出は成功し、XはMで束縛される。ここで、Mは通信変数である。■

FBS間のXについての流辺A=>Bに対して、A→Send(X,CX),Receive(CX,X)→Bとして、得られた副TGを図2に示す。図2は、→が基本式間の意味のある順序(<<)を示すハッセ図である。

5．通信デッドロック

トランザクションの並行実行における通信についての前進時と後戻り時におけるデッドロックの防止方法と解除方法を示す。

5．1前進デッドロック

導出の前進時におけるデッドロック防止法として、

各副トランザクションT_K=(BB_K, $<<_K$)に対して、SendとReceiveの順序を考える。
(1) A$<<_K$Send, Receive$<<_K$Bかつ A<Bのとき、Send$<<_K$Receiveとする。
(2)基本式AとReceiveの間に意味のある順序関係がないとき、Receiveの選択を遅らせる。
(3)同様にSendの選択を促進する。
(2)と(3)は、待ち時間を減少するためである。
[命題]分割された副トランザクションの並行実行は、通信についてデッドロックを生じない。
[証明]詳細については[TAK87]を参照されたい。■

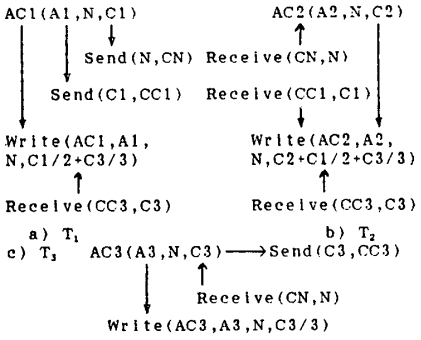

図２　副トランザクショングラフ

５．２後戻りデッドロック

前進時の導出が失敗すると後戻りが行われるが、このときデッドロックが生じる可能性がある。図３で、T_2とT_3はCommitで他の副トランザクションの終了を待ち、T_1はWriteで導出に失敗したとき、Receiveに後戻りすることによってデッドロック状態となる。このトランザクションの後戻り時におけるデッドロック状態を検出し、解除する方法について考える。ここで、T_3をCC3を束縛させるAC3まで後戻りさせることにより、デッドロックを防げる。このT_3での後戻りを強制後戻り[図４]という。

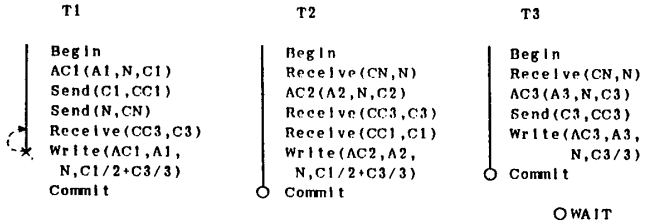

図３　後戻りデッドロック

図５は、T_1とT_2のWriteで導出に失敗した例である。T_1では、C1,C3の組で導出に失敗したが、お互いがReceiveのSenderに強制後戻りし、新たな値C1',C3'の組で導出を行うと、T_1を強制後戻りした時に得られる可能な組C1,C3'とT_2を強制後戻りした時に得られる組C1',C3について導出を行えないという問題が生じる。その解除方法としてReceiveに後戻りしたとき、WAITを早く送信した副トランザクションをSenderに強制後戻りさせ、トランザクションを再実行する。高信頼放送通信網を用いると、同時にWAITを放送しても各トランザクションに一定の順で到着する。この順序で早く届いたものについて強制後戻りを行う。

デッドロック状態を解除する方法として、Send,Receiveの手続き的意味を以下に示す。

[後戻り時の手続き的意味]
(1)後戻り時
(a)Receive(C,X)に後戻りしたとき、WAIT(C)を放送し、Cの到着を待つ。
(b)Send(X,C)に後戻りしたとき、FAIL(C)を放送し、導出は失敗する。
(2)強制後戻り時
ReceiveのSenderに後戻りしたとき、そのSenderを含む副トランザクションを、そのSenderを束縛している基本式（ホスト）まで強制的に後戻りさせる。
このとき、
(a)Receive(C,X)では、Cをクリアしない。
(b)Send(X,C)では、FAIL(C)を放送する。
(3)WAIT(C)の受信
ここで、副トランザクションT_Kの現在の位置をcurrent(T_K)とすると、Send(X,C)$<_K$current(T_K)ならば、Send(X,C)のホストまで強制後戻りし、DO(C)を放送する。
(4)FAIL(C)の受信
Receive(C,X)まで強制後戻りする。
(5)UNDO(C)の受信
current(T_K)=Receive(C,X)ならば、WAIT(C)を放送する。
(6)precommitの放送
current(T_K)=commitならば、precommitを放送する。

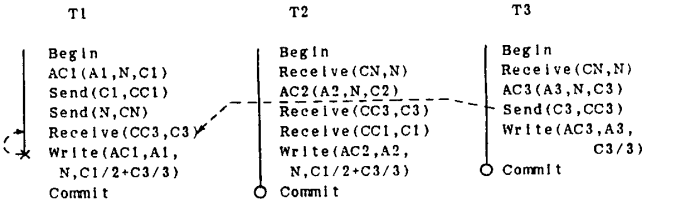

図４　強制後戻り

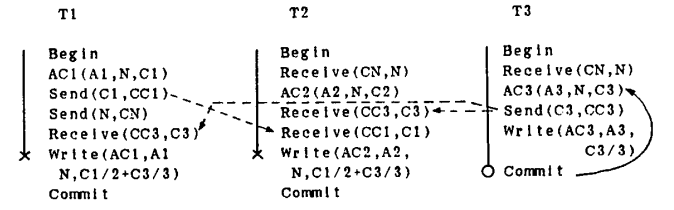

図５　導出が複数で失敗したとき

６．おわりに

本論文により、Prologによるトランザクションの記述と分割方法を示した。また、副トランザクションの並行実行において、前進時に通信についてデッドロックの防止方法と、後戻り時のデッドロックの検出と解除方法を示した。今後の課題として、並列な後戻りと冗長な実行を行わない後戻りについて検討している。

参考文献

[TAK87] Takizawa, M., "Transaction Management by Prolog," Proc. of Logic Conf., ICOT, 1987.
[TAK88] 滝沢 誠、宮島 勝己 "Prologトランザクションの並行実行," 情報処理学会(MDP)研究会, Vol.88, No.35, pp.73-80, 1988.