1H−7 入れ子型トランザクションの並行実行によるデッドロック解除法

安沢　伸二　　滝沢　誠　　大内　拓磨

東京電機大学

1. はじめに

情報システムが分散化される一方で、*CAD*等の応用では、複数のオブジェクトを操作するトランザクション応用が重要になってきている。分散型システムは、通信網によって結合された複数のオブジェクトから成り立っている。オブジェクトは、抽象データ型であり、データ構造とデータ構造を操作するための演算を提供する。トランザクションは、複数のオブジェクトの演算から成り立っている。

各オブジェクトoの演算opは、oの他の演算や他のオブジェクトの演算を呼び出すことができる。このようなトランザクションを入れ子型トランザクションと呼ぶ。オブジェクトrの演算aがオブジェクトpの演算bとオブジェクトqの演算cを呼び出すことを仮定する。もし、aがbとcを独立に呼び出すことができるなら、bとcは、並行に実行することができる。

トランザクションの同期機構にロック機構が用いられるとデッドロックが発生する。本論文では、入れ子型トランザクションを並行実行した場合に起こるデッドロック問題について考える。また、デッドロック解除に必要な演算を補償演算[1]を用いて部分的にアボートする方法を示す。

2章では、システムモデルについて述べる。3章では、入れ子型トランザクションを定義する。4章では、並行実行によるデッドロックについて考察する。5章では、補償演算によるデッドロック解除法について述べる。6章では、補償演算のデッドロックについて述べる。

2. システムモデル

分散型システムMは、高信頼な放送通信網である*SPO*通信網[2]によって結合されたオブジェクトから成り立っている。*SPO*通信網では、各オブジェクトは、送信順序を保ち、紛失することなくメッセージを送ることができる。

各オブジェクトoは、抽象データ型であり、データ構造D_oとD_oを操作する演算P_oを提供する。oは、演算opの補償演算$op^{\sim}$を持つ。$op^{\sim}$は、演算の意味によって定義される。システムの状態をsとし、sに演算opを実行して得られた状態を$op(s)$とする。補償演算$op^{\sim}$は、$op^{\sim}(op(s))=s$となる演算である。

[例1]分散型システムの例として、管理者 *manager*、従業員 *worker*、会議室 *room*などのオブジェクトから構成される*Office*システムを考える。管理者と従業員は、それぞれ予定表 *msch*と *wsch*をデータ構造として持ち、操作演算として予定の予約 *Book*と解約 *Cancel*を持つ。さらに管理者オブジェクトは、会議予約演算 *OpenM*を提供し、会議の予約を行う。会議室には、予定表 *rsch*があり、操作演算として予約と解約を提供している。予約 *Book*の補償演算は、解約 *Cancel*である。□

3. トランザクション

入れ子型トランザクションTは、オブジェクトの演算より構成される。各演算は、他の演算を呼び出すことができるので、トランザクションは木構造をしている。根を根トランザクション、葉でない節を副トランザクション、葉をアクションとする。

[例2]例1の*Office*において、入れ子型トランザクションの例を示す。管理者*mgr*は、従業員aとbと会議を開くために予定表に会議の予定を書き込み、会議の参加者と会議室*room*の予約を行なう。会議予約トランザクションを図1に示す。□

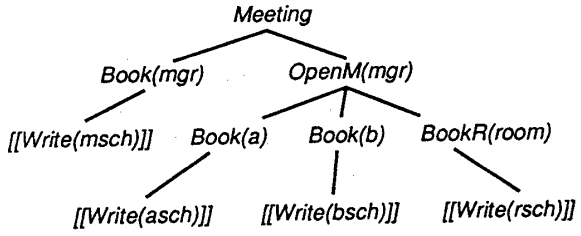

図1．会議予約トランザクション

オブジェクトoの演算opが、m個の演算op_1, ..., op_mを呼び出すとする。演算opによって呼び出される演算間の実行順序を関係→で表す。$op_1 \rightarrow op_j$は、op_jが、op_1のコミット後に実行されることを示す。もし、op_1とop_j間に順序関係が存在しなければ、op_1とop_jは、並行に実行される。*[op*と*op]*を演算の開始と終了とする。*op]*は、*op*が呼び出したすべての演算が終了するのを待ち、コミットする演算である。従って、並行実行する演算は、*AND*並行である。例えば、図1で、*Book(a)*、*Book(b)*、*BookR(room)*は、並行に実行できる。

4. 並行デッドロック

本論文では、デッドロックは、デッドロックマネージャによって検出される。デッドロックを定義するために、依存関係を定義する。

[定義4.1]次のいずれかの条件を満足するならば、演算op_1はop_2に依存している($op_1 \Rightarrow op_2$)という。(1)op_1は、op_2によって獲得されているオブジェクトを待っている、(2)あるトランザクションTにおい

Resolution of Parallel Deadlock by Partial Abortion

Shinji Yasuzawa, Makoto Takizawa, and Takuma Ouchi
Tokyo Denki University

て op_1が op_2より先行（$op_1 \rightarrow_T op_2$）して実行される、(3) $op_1 \Rightarrow op_3 \Rightarrow op_2$ となる演算op_3が存在する。□

システムの状態を拡張待ち(*EWF*)グラフ[1]で示す。*EWF*グラフが巡回閉路を含むならば、システムは、デッドロック状態にある。

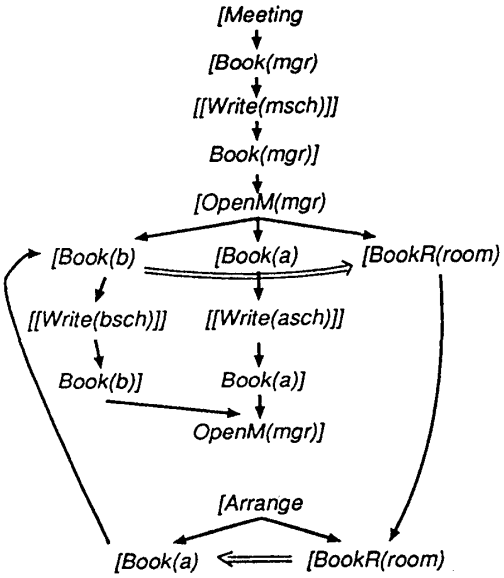

図2．並行実行におけるデッドロック

[例3]図2の*EWF*グラフにおいて、トランザクション*Arrange*は、従業員*b*と会議室の予約を行なうトランザクションとする。*Arrange*は、*Meeting*に獲得される*b*を待っており、*Meeting*は、*Arrange*に獲得される*room*を待っていることを仮定する。このとき、*EWF*グラフは、巡回閉路を含まない。しかしながら、*AND*並行によって、二つのトランザクションは、デッドロック状態にある。□

例3で示したデッドロックを定義するために、以下の定義を定義4.1に加える。

[定義4.2]トランザクション*T*によって op_1と op_2は、呼び出されている。op_1は、オブジェクトを獲得し、op_2は、オブジェクトを待っているとき、$op_1 \Rightarrow op_2$である。□

定義4.1と4.2の依存関係によって、図2の*EWF*グラフは巡回閉路を含む。

5. デッドロック解除法

デッドロックが生じるとデッドロックしているトランザクションの一つを選択し、アボートすることによってデッドロックを解除する。従来の方法では、実行された演算のすべてをアボートしていたが、本論文では、補償演算を用いてデッドロック解除に必要な演算のみを部分的にアボートをする。デッドロック状態のトランザクション*T*において，*T*内のどの演算をアボートするかは、他のトランザクションを待たせている演算*op*が実行中であるか、すでにコミットしているかによって異なる。*AO*(*T*)を*T*においてアボートされる演算の集合とする。

[アボートされる演算]
(1) *op*が実行中であるなら、*op*と*op*によって先行されるすべての演算は、アボートされる。
(2) *op*がすでにコミットしているなら、*op*を呼び出す*op'*と*op'*によって先行されるすべての演算が、アボートされる。

デッドロックマネージャが、デッドロックを検出すると、アボートするトランザクション*T*を決定し，メッセージ*Pabort*(*T*,*AO*(*T*))を放送する。各オブジェクトの以下の動作によって、*AO*(*T*)内の演算は、部分的にアボートされる。

[オブジェクト*o*の動作]
(1)[*Pabort*(*T*,*AO*(*T*))の受信]*o*が*AO*(*T*)内の演算によって呼ばれる演算*op*を実行中ならば、*o*は、*op*の実行を中止し、親の演算*op'*を実行するオブジェクトにメッセージ*Compensate*(*T*,*op'*)を放送する。
(2)[*Comensate*(*T*,*op*)の受信]*o*は、*op*の補償演算*op~*を実行する。もし、*op*が並行に演算を呼び出すならば、すべての*Compensate*(*T*,*op*)を受信してから*op~*を実行する。次に、親の演算に対して*Compensate*を送る。もし、*op*が*AO*(*T*)内の演算であるなら、*op*と*op~*によってロックされたすべてのオブジェクトを解放する。□

6. 補償演算のデッドロック

補償演算*op~*は、*op~*が用いるオブジェクトのロックを要求する。もし、*op~*が*op*の獲得したオブジェクト以外のオブジェクトを要求すると*op~*を実行することでデッドロックが発生する場合がある。補償演算を並行に実行することで起きるデッドロックは、定義4.1、4.2によって同様に定義される。補償演算を実行することによって解除できないデッドロックを解除不能なデッドロックという。解除不能なデッドロックは、新たなオブジェクトを要求しない補償演算によってのみ解除される。

7.まとめ

本論文では、分散型システムにおいて、トランザクションが並行実行されたときにおこるデッドロックと補償演算を用いたデッドロック解除法を示した。

参考文献

[1] Yasuzawa, S., Takizawa, M., and Ouchi, T., "Resolution of Parallel Deadlock by Partial Abortion," *Proc. of International Symposium on Communications (ISCOM)*, Tainan, 1991, pp.708-711.
[2] Nakamura, A. and Takizawa, M., "Reliable Broadcast Protocol for Selectively Partially Ordering PDUs (SPO protocol)," *Proc. of the IEEE 11th ICDCS*, Arlington, 1991, pp.239-246.