分散データベースにおける トランザクション実行のプレアナリシス

山崎晴明　吉田勇　松下温
（沖電気工業株式会社）

1. はじめに

近年の通信ネットワーク技術の発展は情報処理の様々な分野に大きな影響を与えている。特にデータベース処理の分野においては、単一の大規模データベースシステムに比べて数多くの利点を持つ分散型データベースというアプローチを可能とするに到っている。しかしながら、このような分散データベースを構築するに当っては、解決すべき技術的問題も数多く残されている。特にデータのコンシステンシイ（一貫性）をいかに維持してゆくかが主要な技術課題のひとつである。

分散データベースあるいは広く分散データの処理という環境においては、このようなデータのコンシステンシイを保つためには、2つの問題を考慮する必要がある。詳細は後述するが、ひとつはトランザクションが入力されたとき、そのトランザクションをコンシステントに実行することであり、他のひとつはトランザクション群が入力されたとき、各トランザクション間の関係をコンシステントに保ちつつ実行させることである。

本稿の目的は、後者のトランザクション間コンシステンシイを分散環境下で維持するという問題に対し、トランザクションのプレアナリシスに基づくひとつの解を示そうとするものである。

一般に、多くのデータベース，ファイル処理においては、データの更新の形態（どのようなデータを読み、どのようなデータに書き込みを行うか）があらかじめ判明しているいわゆる定型業務が対象となることが多い。このような業務では入力され得るトランザクション間のコンフリクトの形態を前もって解析しておき、同時実行可能なトランザクションの集合を決定しておくことが、処理の並列性をあげるために極めて有効な方法となる。

これまでに、このようなプレアナリシス方式はSDD－1で提案されてはいたが、ここで述べたようなトランザクション内とトランザクション間の2つのコンシステンシイの違いを明確に区別していないため解析の手法が複雑であり、さらにタイムスタンプを用いた更新処理をベースとしている。ここで提案するプレアナリシス方式は、より簡単な代数演算を逐次実施していくのみのより簡潔なアルゴリズムを用いるもので、アップデート処理には必ずしもタイムスタンプを必要とするものではなく、この意味でより一般化されている。

2. データのコンシステンシイ維持

データのコンシステンシイを維持するには、次の2種の問題を考慮しなくてはならない。

(1) リード／ライトのアトミック性

たとえばT_aをファイルR_1，R_2を読み込んで更新データを作成し、それをファイルW_1，W_2に書き込むトランザクションとする。このときT_aのリード操作はコンシステントに行われなければならない。コンシステントでないリードとは、たとえばT_bをR_1，R_2にライトするトランザクションとしたとき、T_bが

R_1にライトし、次にR_2にライトする直前でT_aによるR_1, R_2のリードが行われたとすれば、T_aのリードはコンシステントでない。同様のことがライト操作についてもいえる。たとえばT_cをファイルW_1, W_2にライトするトランザクションとしたとき、T_cがW_1にライトした後、W_2にライトする直前でT_aによるW_1, W_2へのライトが行われると、ライト時のコンシステンシイが失われる。明らかに、リード／ライト実行の際、対象データに対し全く同時に操作を行うか、操作対象にロックをかけるかしてリード／ライト操作をアトミックなものとする機構があれば、このコンシステンシイは保証される。このような機構を設ける方式には、対象データにロックをかける方式，データとトランザクションにタイムスタンプを用いる方式等がある。さらには、ネットワークの遅延によるリード／ライト操作の時間差を避けるため、Ethernet のようなブロードカストタイプのネットワークを利用するという方法も有り得る。

いずれの手法を用いるにせよこれらの方式の詳細を述べることが本稿の目的ではないため、以降ではこのアトミック性を保証する機構は上述のいずれかの方式によって実現されていると仮定する。

(2) 順序づけコンシステンシイ

トランザクション群が与えられたとき、各々のトランザクションの実行は前述のアトミック性が保証されていても全体としてはコンシステントな実行とならないことがある。たとえば T_a, T_b, T_c をトランザクションとし、T_aはF_1をリードし、F_2にライトする、T_bはF_2をリードし、F_3にライトする、T_cはF_3をリードし、F_1にライトするものとする。各トランザクションのリードおよびライト対象はひとつであるから、どのような実行を行おうともリード／ライトのアトミック性はこの場合保証される。しかしながら、たとえば3つのトランザクションのリードを一斉に行い、その後ライトを一斉に行った場合、T_aはT_cによってライトされる前のF_1を読んでいるため、T_aはT_cに先行したと考えられる。同様にT_bはT_aに先行し、T_cはT_bに先行したと考えられる。従って、T_a, T_b, T_cの間にどれが先に実行されたかを決定する順序関係が定義できなくなってしまう。このようなトランザクション群の実行形態を順序づけコンシステンシイの失われた形態と呼び、本稿ではこの問題に焦点をあてる。

3. 用語の定義とシステムモデル

システムは、サイトと呼ばれるローカルなデータベースシステムとそれらを結合する通信チャネルよりなる。各サイトに分散配置されるデータの単位をフラグメントと呼び、必要に応じていくつかのサイトに重複して保持されるものとする。また、トランザクションは、それが参照するフラグメント集合（リードセット）と、それが更新するフラグメント集合（ライトセット）とを持つ。また、すべてのトランザクションの実行は、必要なデータを読み、それによって更新データを一時記憶域に作成するリードアクションと、その更新データを二次メモリ上に実際に書き込むライトアクションとからなる。さらに、前述のアトミック性を保証する機構をシステムが持つことを仮定する。

また、同一のリードセットおよびライトセットを持つトランザクションの集合をトランザクションクラス（または単にクラス）と呼び、記号a, b…のようにあらわす。次に、クラスaのリードセットとクラスbのライトセット

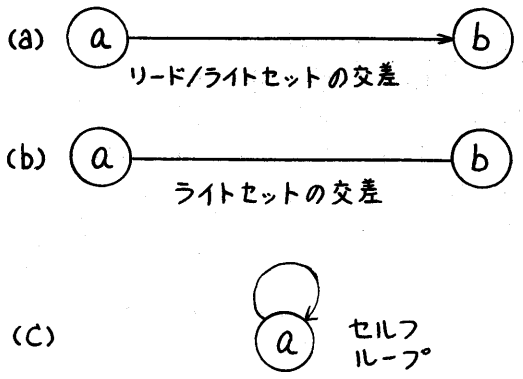

図1. クラスの交差に関する記法

とが空でない共通部分を持つとき、図1(a)のような方向付きエッジであらわし、これをaのリードエッジと呼ぶ。これは、クラスaおよびbに属するトランザクションが同時に実行されたとき、作り出される結果は、aを実行後bを実行することによって得られる結果と一致することを表わしている。また、クラスaとbのライトセットどうしが交差するときは、図1(b)のような方向を持たないエッジで表わし、これをaまたはbのライトエッジと呼ぶ。また、aのリードセットとaのライトセットとが交差するときは、図1(C)のようにあらわし、このエッジをaのセルフループと呼ぶ。

システムが扱うすべてのクラスをノードとし、それらの間の交差関係をエッジによりあらわしたものをクラス競合グラフと呼び、例を図3に示す。

4. トランザクションクラスの解析

以降では簡単のため、セルフループを含まないクラス競合グラフを対象とする。

4.1 トランザクション実行記述

aをトランザクションクラスとしたとき、aに属する任意のトランザクションのライトアクションをひとつのクラスのように扱いa_wと記述する。与えられたクラス競合グラフ上のクラスおよびそのライトアクションをあらわす記号の重複を許さない任意の組合せをトランザクション実行記述（または単に記述）と呼ぶ。ただし、ひとつのクラスとそのライトアクションをあらわす記号とは同一記述中には出現しないものとし、さらにライトアクションのアトミック性から同一記述中にはライトエッジを共有する2つのライトアクションは現われないものとする。

次に、ある記述$S=a_1a_2\cdots a_n$と、現在すでに同時実行を行っているトランザクション集合Tとが与えられたとき、すべてのiについてa_iであらわされるクラスまたはアクションがTに含まれていれば、Sは真であるといい、$T(S)=1$であらわす。もしTに含まれないa_iがひとつでもあれば$T(S)=0$であらわし、記述Sは偽であるという。同一クラス競合グラフ上の2つの記述S, S'が与えられたとき$T(S)=1$ならば$T(S')=1$が常に成り立つときSはS'をインプライすると呼ぶ。さらに、$T(S)=1$であって、Sをインプライするどのような$S'(\neq S)$についても、$T(S')=0$となるようなトランザクション集合Tの実行のことを、記述Sの実行と呼ぶ。

4.2 ラベル付グラフ

クラス競合グラフと記述$S=a_1a_2\cdots a_n$とが与えられたとき、すべてのa_iについて次の操作を行ったものをSによりラベル付けされた競合グラフと呼ぶ。

1) a_iがクラス記号であればa_iのすべてのリードエッジにラベル付けする。

2) a_iがライトアクション記号であれば、a_iのライトエッジにラベル付けを行う。このとき、まだ向きの定義されていないライトエッジにはa_iをソースとする向きを定義する。

このようにして、記述Sのラベル付

けされた競合グラフが与えられたとき、Sの任意の要素記号から出発して、Sでラベル付けされた各エッジの向きに従い、Sの各要素記号を1回ずつ通り、もとの要素記号に至る道が存在するとき、記述Sはループであるという。次に、記述Sの実行を行ったとき、その実行結果がSの各要素記号を（どのような順序でもかまわないが）逐次実行したときの結果に常に一致するとき、記述Sは順序付けコンシステンシイを満たす記述と呼ぶ。このとき、次の命題が成り立つ。

命題1　Sを与えられたクラス競合グラフG上の記述とする。SがG上のどのようなループもインプライしなければ、記述Sは順序付けコンシステンシイを満たす。

なお、アトミック性の保証されたシステムにおける順序付けコンシステンシイを満たす記述のことを同時実行可能な記述と呼ぶ。

4.3　ベーシックループ

クラス競合グラフGが与えられたとき、G上のループに対応する記述のうち他のループをインプライするループをすべて取り去った記述の集合をベーシックループ集合と呼び、その各々をベーシックループと呼ぶ。

命題2　記述Sが与えられたとき、Sがどのようなベーシックループもインプライしなければ、Sは同時実行可能な記述である。

こうして、クラス競合グラフとその上のすべてのベーシックループを導くことでトランザクションクラスの解析は終了する。なお、後述するように、トランザクションの制御の目的は、このような同時実行不可能な記述の実行を阻止することにある。

4.4　クラス解析の例

ある販売会社は図2のように、営業、購入、管理／検査の3部門から成っている。各部門の業務をそれぞれクラスa, b, cとしたとき、そのリードセット，ライトセットおよび処理の概要を表1に示す。

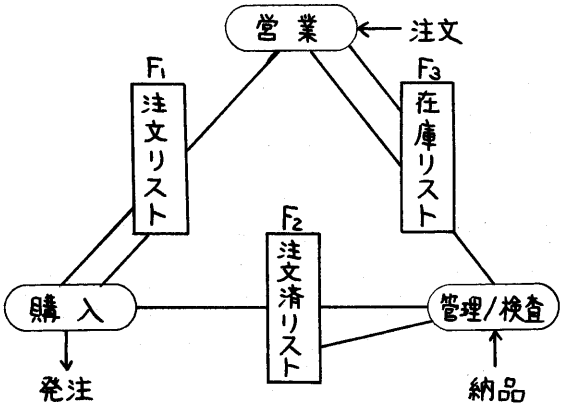

図2　注文処理システム

表1　各業務のリード／ライトセット

クラス	リードセット	ライトセット	動作
a（営業）	F_3	F_3 F_1	注文を受付け、注文品目がF_3中にあれば庫出し、販売する。F_3中になければ、その品目を注文リストF_1中に付け加える。
b（購入）	F_1	F_1 F_2	F_1中の品目をメーカ側に発注する。発注の済んだ品目については、それを注文済リストF_2に付け加え、F_1より取り除く。
c（管理／検査）	F_2	F_2 F_3	メーカ側より納品された品目の検査を行い、合格の品目についてF_2より取り除き、さらにその品目を在庫リストF_3に追加する。

この例におけるクラス競合グラフは図3のようになり、また各ベーシックループとその関連フラグメントとの対応は表2で与えられる。

この例では、たとえばループ baw が実行されると、次のような矛盾が起ることがわかる。F_1 の内容がA、B、C、F_2 の内容が空、F_3 の内容がX、Yであったとする。さらにユーザから営業部門がX、Dなる品目の注文を受けたとする。発注業務 T_b が F_1 を読むことと、販売業務 T_a による品目Dの F_1 への追加が同時に行われると、次に T_b のライトアクション実行後では F_1 は空となってしまい、品目Dの発注という業務が消失してしまう。従って baw は実行されてはならないことがわかる。

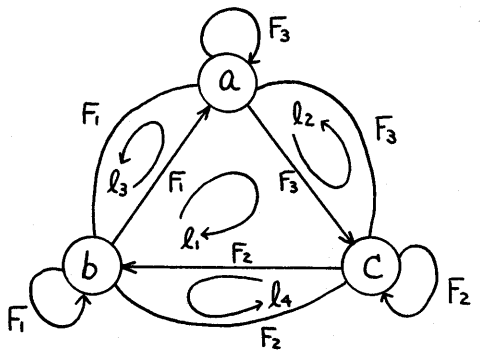

図3　クラス競合グラフ

表2　ベーシックループとその関連フラグメント

ベーシックループ	記述	関連フラグメント
ℓ_1	acb	F_1, F_2, F_3
ℓ_2	ac_w	F_3
ℓ_3	ba_w	F_1
ℓ_4	cb_w	F_2

5. 実行ステータスとその演算

5.1 実行ステータス

あるベーシックループ S を $S=a_1a_2\cdots a_n$ とする。いまクラス a_i に属するトランザクションが m_i 個実行中であるとすれば、非負の整数を要素とするn桁のベクトル $(m_1, m_2, \cdots, m_n)$ で、S に属する各トランザクションの実行状態をあらわすことができる。この実行状態を表現するベクトルをロックステータス（Lst）と呼ぶ。同様にして、実行終了したトランザクション数をベクトル $(m_1', m_2', \cdots, m_n')$ であらわすことができる。これをリリースステータス（Rst）と呼ぶ。さらにそのベクトル対（Lst, Rst）をベーシックループ S に対する実行ステータスと呼ぶ。従ってトランザクションの実行要求は Lst における該当桁への1の加算を、実行終了は Rst の該当桁に対する加算を意味することになる。さらに、Lst のすべての桁が非ゼロであったとき、その実行ステータスはコンフリクティングであると呼ぶ。またベーシックループの定義から、実行ステータスを巡回シフトしてもその意味は変わらない。従ってコンフリクティングでない実行ステータスはいつでも最下桁が0となるよう巡回シフトしておくものとする。

5.2 実行ステータスの変換

コンフリクティングでない実行ステータス：$Lst=(a_1, a_2, \cdots, a_n)$, $Rst=(b_1, b_2, \cdots, b_n)$ が与えられたとする（但し、$a_i \geq b_i$ for $\forall i$, $a_n=b_n=0$）。このとき変換 φ：$\varphi((Lst, Rst))=(Lst', Rst')$ を次のように定義する。

$Lst'=(a_1', a_2', \cdots, a_n')$, $Rst'=(b_1', b_2', \cdots, b_n')$

$$a_1' = a_1 - b_1 \qquad (1)$$

$$a_i' = \begin{cases} a_i - b_i & \text{if } a_{i-1}'=0 \\ a_i & \text{if } a_{i-1}'\neq 0 \end{cases} \qquad (2)$$

$$b_1' = 0 \qquad (3)$$

$$b_i' = \begin{cases} 0 & \text{if } a_i \neq a_i' \\ b_i & \text{if } a_i = a_i' \end{cases} \quad (4)$$

たとえば、

Lst = (4,5,3,0,1,2,0,0,1,3,1,6,0)

Rst = (2,2,3,0,0,1,0,0,1,3,1,5,0)

とすれば、

Lst' = (2,5,3,0,1,2,0,0,0,0,0,1,0)

Rst' = (0,2,3,0,0,1,0,0,0,0,0,0,0)

となる。変換φは、原則としてLstとRstの対応桁ごとの減算によりLst'の対応桁を生成するが、Lst'の桁に0を生成したときのみ減算作用素が下位桁にもシフトし、そうでないときはもとのLstの桁がそのままLst'に引き継がれると考えると理解しやすい。明らかに、α = (Lst, Rst)とすると、

$\varphi(\varphi(\alpha)) = \varphi(\alpha)$ が成り立つ。

なお、このφはトランザクションの実行終了通知があったとき、実行ステータスをリセットするのに用いられる。一般に、トランザクションの実行終了通知があったとき、Lstの該当桁から1を減算するだけでは順序付けコンシステンシイが保たれない。

6. トランザクション実行時の制御

分散データベースシステムに伴う一般的問題点のひとつに、ネットワーク内の伝送遅延には、そのトポロジーに依存したばらつきがあるため、制御情報等の複数サイトへの伝送においても、それの各サイトへの到着時間に、ばらつきが生じてしまうということがある（Ethernetのようなブロードカスト伝送向きのネットワークではこの限りではない）。このため、トランザクション実行要求等の制御情報伝送時には、1対1通信を基本とする通常の通信プロトコルとは異なる（1対多通信を基本とする）プロトコルを導入する必要がある。本稿の方式では、以降に示すようにこのプロトコルは、多数決原理に基づくものとなっている。

6.1 トランザクション実行プロトコル

(1) リードアクション実行要求の送信

ユーザからトランザクションT_aの実行要求を受け付けたプロセスは、そのクラス（たとえばα）と、それを含むベーシックループとを判別し、関連フラグメントを持つすべてのサイトに（自分のサイトも含めて）リードリクエスト（REQ RD）を送信する。これを受信したサイトでは、Lstのαに対応する桁に1を加算し、その結果コンフリクティングとならなければACKを、そうでなければNACKを返送する。

(2) リードアクションの実行

もしプロセスが過半数からのACKを得られなかったようなベーシックループがひとつでもあった場合、ACKを応答したサイト（もしあれば）に対しABORTを送信し、T_aの実行を放棄する。なお、ABORT 受信サイトは、Lstの該当桁から1を減ずる。

一方、すべての関連ベーシックループに対し過半数のACKを受信したプロセスは、NACKを応答したプロセス（もしあれば）に、REQ RDを送信し続ける。すべてからACKが得られたとき、プロセスはT_aのリードアクションを実行する。

(3) ライトアクションの実行

読み込みを終了し、更新データを作成したプロセスは、次にα_wを含むベーシックループを判別し、関連フラグメントを持つすべてのサイトにREQ WTを送る。その後の処理は、(2)と同様であるが、ライトアクション実行要求時にABORTされたT_aは、実行中止とせず、一定の時間間隔をおいて再度実行が試みられる。

(4) 終了コマンドの送出

アトミック性を保証する機構により更新を終了したプロセスは、COMPLETE を、リード/ライトアクションに関与したすべてのサイトに送信する。これを受信したサイトではRstの該当桁に1を加え、変換φを実行する。

こうして、このプロトコルにより、すべてのベーシックループの実行は阻止され、順序付けコンシステンシイは保たれることになる。

7. 評価

本稿で提案したプレアナリシスによるトランザクション実行制御が、従来のリソースをロックする制御方式と比較してどのくらい効率が良いかを定量的に比較する。しかし、本格的な解析は他の機会にゆずるとし、ここでは図4で示されるようなクラス競合グラフを持つ場合について、任意の時点でトランザクション実行要求が発生したとき、コンフリクトのためにその実行が阻止される確率、すなわち呼損率を求めて比較する。

7.1 評価モデル

評価の対象とするのは、図4で示されるクラス競合グラフを持つシステムとする。すなわち、

(1) 3つのトランザクションクラス a, b, c がある。

(2) それぞれのクラスはセルフループを持つ。従って同一クラスに属するトランザクションはひとつずつしか実行できない。

(3) a のライトセットが c のリードセットと、b のライトセットが a のリードセットと、c のライトセットが b のリードセットと交差する。

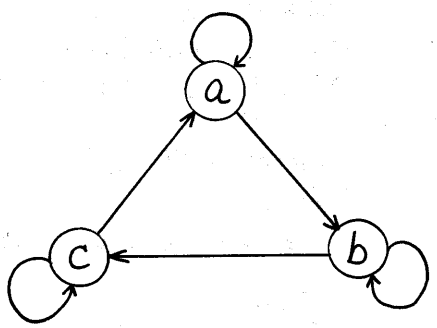

図4 評価対象のクラス競合グラフ

さらに、以下のことを仮定する。

(4) 各トランザクションクラスにおけるトランザクション実行要求の発生分布は同一であり、平均入のポアソン分布に従う。

(5) 各トランザクションクラスにおけるひとつのトランザクションの処理に要する時間、すなわちサービス時間の分布は同一であり、それぞれ平均 μ の指数分布に従う。

7.2 リソースロック方式の呼損率

リソースをロックする方式では、ひとつでもトランザクションが発生して実行を開始すると、他のトランザクションは一切実行できなくなってしまう。従ってこの場合には、3つのトランザクションクラスのトランザクションの発生はすべて同じものとみなし、入力が平均3入のポアソン分布、サービス時間が平均 μ の指数分布の場合の M/M/1 (1) モデルと考えることができる。

従って、$\lambda/\mu = \rho$ とおき、リソースロック方式の呼損率を $f_\ell(\rho)$ とすると、

$$f_\ell(\rho) = \frac{3\rho}{1+3\rho} \qquad (5)$$

となる。

7.3 プレアナリシス方式の呼損率

本稿で提案したプレアナリシスによるトランザクション実行制御の方式は図5で示すS_0～S_9の10ヶの状態をもつContinuous-Time Markov Chainとしてモデル化できる。図5で示された各状態の上段はロックステータス(Lst)を、下段はリリースステータス(Rst)をあらわすが、前述の場合と違って、ここでは左から順にa, b, Cの状態を固定的に表現しており、最下桁が0となるようなシフトは行わない。

Continuous-Time Markov Chainの理論に従い、以下の記号を定義する。

π_i：定常状態における任意の時点でシステムの状態がS_iである確率

$p_{ij}(t)$：今システムがS_iにあるとして、t時間後にS_jとなる確率

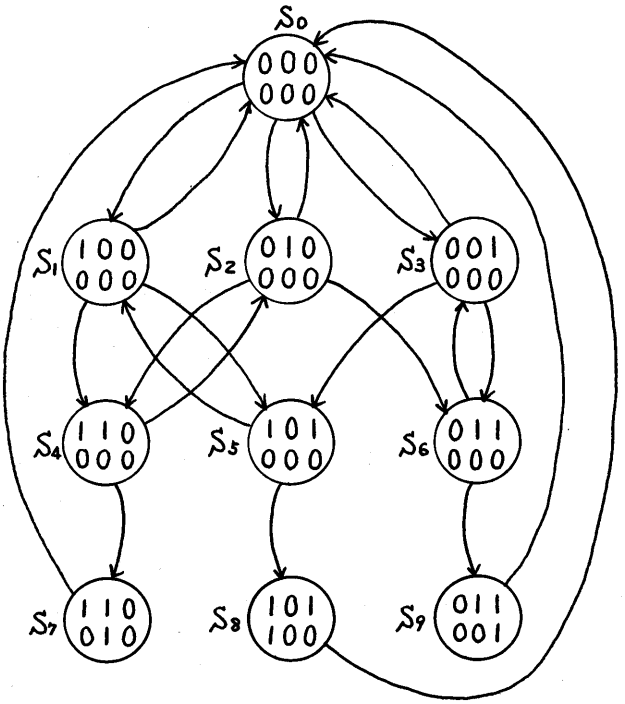

図5　システムの状態とその遷移

q_{ij}：transition rate，すなわち、

$$q_{ii} = \lim_{\Delta t \to 0} \frac{p_{ii}(\Delta t) - 1}{\Delta t} \qquad (6)$$

$$q_{ij} = \lim_{\Delta t \to 0} \frac{p_{ij}(\Delta t)}{\Delta t} \qquad i \neq j \qquad (7)$$

π_i, q_{ij}を要素とする行列を

$$\Pi = [\pi_i]$$

$$Q = [q_{ij}]$$

とすると、

$$\Pi Q = 0 \qquad (8)$$

が成り立つ。

従って、各q_{ij}を求めればよいことになる。各q_{ij}は例えば以下のようにして求められる。

$$q_{00} = \lim_{\Delta t \to 0} \frac{p_{00}(\Delta t) - 1}{\Delta t}$$

$p_{00}(\Delta t)$は今システムがS_0(アイドル状態)にあるとしたとき、Δt時間後もS_0のままである確率であるから、いいかえれば、Δtの間にaもbもCも発生しない確率である。従って、

$$p_{00}(\Delta t) = e^{-\lambda \Delta t} \cdot e^{-\lambda \Delta t} \cdot e^{-\lambda \Delta t}$$

であり、

$$q_{00} = -3\lambda \qquad (9)$$

となる。

同様に、例えば$p_{22}(\Delta t)$はΔtの間にbの処理が終了せず、かつaもCも到着しない確率であるから、

$$p_{22}(\Delta t) = e^{-\mu \Delta t} \cdot e^{-\lambda \Delta t} \cdot e^{-\lambda \Delta t}$$

であり、

$$q_{22} = -(2\lambda + \mu) \qquad (10)$$

となる。

同様にして、すべてのq_{ij}が求められ、Qは以下のようになる。

$$Q=\begin{bmatrix}
-3\lambda & \lambda & \lambda & \lambda & 0 & 0 & 0 & 0 & 0 & 0 \\
\mu & -(2\lambda+\mu) & 0 & 0 & \lambda & \lambda & 0 & 0 & 0 & 0 \\
\mu & 0 & -(2\lambda+\mu) & 0 & \lambda & 0 & \lambda & 0 & 0 & 0 \\
\mu & 0 & 0 & -(2\lambda+\mu) & 0 & \lambda & \lambda & 0 & 0 & 0 \\
0 & 0 & \mu & 0 & -2\mu & 0 & 0 & \mu & 0 & 0 \\
0 & \mu & 0 & 0 & 0 & -2\mu & 0 & 0 & \mu & 0 \\
0 & 0 & 0 & \mu & 0 & 0 & -2\mu & 0 & 0 & \mu \\
\mu & 0 & 0 & 0 & 0 & 0 & 0 & -\mu & 0 & 0 \\
\mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu & 0 \\
\mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu
\end{bmatrix}$$

式(8)より、以下の式を得る。

$$\begin{aligned}
&-3\lambda\pi_0+\mu(\pi_1+\pi_2+\pi_3+\pi_7+\pi_8+\pi_9)=0\\
&\lambda\pi_0-(2\lambda+\mu)\pi_1+\mu\pi_5=0\\
&\lambda\pi_0-(2\lambda+\mu)\pi_2+\mu\pi_4=0\\
&\lambda\pi_0-(2\lambda+\mu)\pi_3+\mu\pi_6=0\\
&\lambda(\pi_1+\pi_2)-2\mu\pi_4=0\\
&\lambda(\pi_1+\pi_3)-2\mu\pi_5=0\\
&\lambda(\pi_2+\pi_3)-2\mu\pi_6=0\\
&\mu\pi_4-\mu\pi_7=0\\
&\mu\pi_5-\mu\pi_8=0\\
&\mu\pi_6-\mu\pi_9=0
\end{aligned}$$

これらの式と

$$\sum_{i=0}^{9}\pi_i=1 \qquad (11)$$

とから、

$$\pi_0=\frac{1+\rho}{1+4\rho+6\rho^2}$$

$$\pi_1=\pi_2=\pi_3=\frac{\rho}{1+4\rho+6\rho^2}$$

$$\pi_i=\frac{\rho^2}{1+4\rho+6\rho^2} \qquad i=4\sim9$$

を得る。

クラスaのトランザクションが呼損となるのは、状態がS_1および$S_4\sim S_9$のときであるから

$$\pi_1+\sum_{i=4}^{9}\pi_i=\frac{\rho+6\rho^2}{1+4\rho+6\rho^2}$$

クラスb, cについても同様であるから、結局システム全体の呼損率を$f_p(\rho)$とすると、

$$f_p(\rho)=\frac{\rho+6\rho^2}{1+4\rho+6\rho^2} \qquad (12)$$

リソースロック方式の場合の呼損率$f_l(\rho)$と、プレアナリシス方式の場合の呼損率$f_p(\rho)$とを比較したグラフを図6に示す。

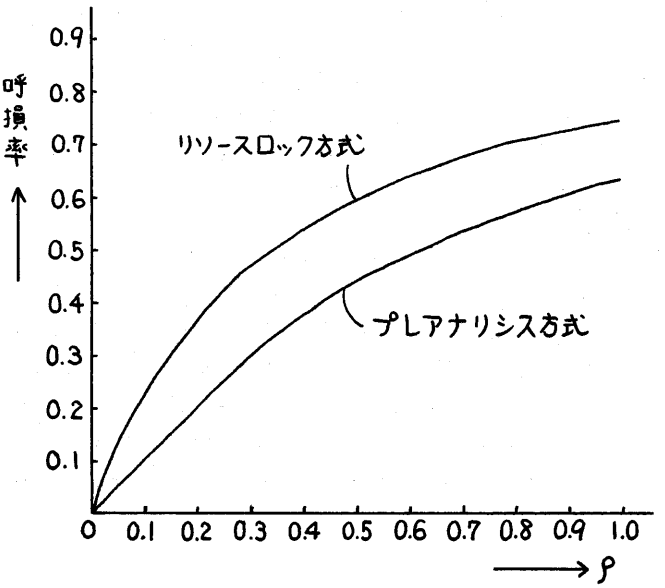

図6　呼損率の比較

8. おわりに

本稿では分散データ処理におけるデータの一貫性維持のため、入力トランザクションを事前に解析し実行を制御する手法を提案した。特に筆者らは、本方式の分散型オフィスシステムへの適用を検討しているが、もちろんこの方式は、分散データ処理のみならず、複数ユーザがデータを共有するときすべてに適用可能である。なお、リソースデータをロックする従来の方式では、ロック要求を、リソースを保有するサイトにのみ送れば良いのに対し、本稿で提案したトランザクションによるトランザクションのロック方式は、ロック要求を関連するすべてのサイトにブロードカストしなければならないため、伝送コストが増すことが考えられる。一方、リソースロックの方式では、一旦ロックを行えばそのリソースを用いるすべてのトランザクションがブロックされるのに対し、本方式では実行ステータスがコンフリクティングとならない限り、リソースを共有する多数のトランザクションが同時実行できるため、処理の並列性は向上する。このため、たとえばEthernetのように、ブロードカスト伝送のコストが1対1通信のそれとあまり変わらないようなネットワークには、本方式は最適な方式となっている。

［参考文献］

1) Rothnie, J.B. and Goodman, N.: A Survey of Research and Development in Distributed Database Management, Proc. of Int. Conf., 3rd VLDB, pp. 48-62 (1977)
2) Rothnie, J.B. et. al.: Introduction to a System for Distributed Database (SDD-1), ACM Trans. on Database System, Vol. 5, No. 1, pp. 1-17 (1980)
3) Bernstein, P.A. et. al.: The Correctness of Concurrency Control Mechanisms in a System for Distributed Database (SDD-1), ACM Trans. on Database System, Vol. 5, No. 1, pp. 52-68 (1980)
4) Stonebraker, M.: Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES, IEEE Trans. on software engineering, Vol. SE-5, No. 3 (May 1979)
5) Metcalfe et. al.: Ethernet: Distributed packet switching for Local Computer Networks, CACM, Vol. 19, No. 7 pp. 395-404 (July 1976)
6) 山崎他：分散データベースにおける同期制御のための階層型プロトコル，情報処理学会、分散システム研究会 2-1 (1979年9月)
7) Yamazaki, H. et. al.: A Graph Theoretic Approach for Fault Detection and Recovery in a Distributed Database, Proc. of ICCC, pp. 237-242 (1980)
8) Kleinrock, L.: Queueing Systems Volume I: Theory, John Wiley & Sons (1975)