データベースシステム　109－30
（１９９６．７．２５）

並行制御アルゴリズムのシミュレーション

中津　楢男
愛知教育大学総合科学課程情報科学コース

あらまし　構文情報だけを利用した並行制御アルゴリズムうちで、２相ロックアルゴリズムと直列化可能性グラフを用いたＢＢアルゴリズムの性能比較をシミュレーションによって行った。

まず、アルゴリズムの不動点集合（並行制御アルゴリズムによって直列化可能と判断される履歴集合）の大きさを比較し、次に、Agrawalらのモデルに基づいて並行処理のシミュレーションを行った。不動点集合には顕著な差がみられ、処理結果では、競合の少ない場合には両者に差がなく、競合が増えるにつれＢＢアルゴリズムが高いスループットを示すことが確かめられた。

Simulation Results of Database Concurrency Control Algorithms

Narao NAKATSU
Department of Information and Computer Sciences, Aichi University of Education

Abstract　Two kinds of concurrecy control algorithms using only syntactical information are compared. One is the twp phase lock algorithm and the other is a BB algorithm, which is one of serializability test algorithms. The BB algorithm outperforms a 2PL algorithm in highly conflicted cases.

１．はじめに

並行制御のアルゴリズムはデータベースシステムの効率に大きな影響を及ぼすため、これまでも多くの研究が継続して行われており、そうしたアルゴリズムの比較研究も数多く報告されている。並行制御アルゴリズムは、直列化可能な履歴を生成するものであり、２相ロックに代表されるロックを用いるアルゴリズム、時刻印を用いるアルゴリズム、楽観的アルゴリズムなどが代表的なものとして知られている[2,4]。またこうしたアルゴリズムにはさまざまな変形が存在し、細部にわたる記述が不足しているため、それらを共通の土俵で比較することは困難である。

さらに並行制御のアルゴリズムが利用できると仮定している情報にもさまざまな仮定があって、一般にトランザクションに関する情報が多ければ多いほど能率の良いアルゴリズムが得られるものと考えられる。たとえば情報量の少ない順に、(1)トランザクションによって発行される命令の種類がその都度わかる場合、(2)トランザクションがアクセスする項目とアクセスする形態が前もってわかっている場合、(3)トランザクションがアクセスする項目とアクセスする形態を及びその順序が前もってわかっている場合　と分類することができる。さらに、トランザクションをREAD/WRITEの列ではなく、意味をもつ小命令の列として考えたり[3]、トランザクションをより小さいトランザクションに分解する[5]などのように、トランザクションの持つ意味を利用して並行性を高める研究も広く行われている。また、直列化可能性という条件はかなり厳しい条件であるため、アルゴリズムの適用分野を限定して、直列化可能性の定義を拡張する試みがなされている[6,8]。

並行制御アルゴリズムの性能は、各アルゴリズムの不動点集合（そのアルゴリズムによって実行順を変更せずに通過させられる履歴の集合）の大

きさを比較したり、シミュレーションによって比較される場合が多い。これまでの比較結果では、時刻印方式よりは2相ロック方式の方が有効であると報告されている。

本稿では十分なディスクキャッシュがある場合において、2相ロックアルゴリズム[2,4]とSGTアルゴリズムの1種であるBBアルゴリズム[9]をシミュレーションによって比較する。

2．並行制御アルゴリズム

本稿で用いるアルゴリズムを説明する。ここではトランザクションに関して最小の情報だけを利用するアルゴリズムを考える。2相ロックアルゴリズムではトランザクションは共有ロック、排他ロック、アンロックの列と考えられている。その詳細は省略するが、情報量の仮定から、2相ロックアルゴリズムの中でもaggressive2相ロックを用いる。このアルゴリズムはREADやBLIND WRITEを行う直前にロックを行い、コミットと同時にアンロックを行うアルゴリズムである。また2相ロックにおいて、デッドロックが生じた場合のアボート処理の方法によって効率が変化する。2相ロックにおけるデッドロックはwait-forグラフに閉路ができることで検出できる。できた閉路にを構成する頂点に対応したトランザクションの内より1つ選んで後退復帰させるのが通常のやり方である。本稿では、後退復帰させるトランザクションの選び方として、1）そのトランザクションの終了を待っているトランザクション数が最大のもの、2）これまで発行したI／O命令が最小のもの、の2種類を考えた。後退復帰するトランザクションの選び方として1）、2）の方法を採用した2相ロックアルゴリズムをそれぞれ2PLv1,2PLv2と呼ぶ。

BBアルゴリズムではトランザクションは、READonly,READ followed by WRITE およびWRITEの列と考える。READを2種類に区別しているが、いずれの種類のREADであるかはアルゴリズムにわかっているものと仮定している。2相ロック方式においても、共有ロックと排他ロックの区別がアルゴリズムにわかっているものと仮定しており、両者が利用する情報は同じである。

SGTアルゴリズムは競合する命令のすべての組み合わせに対して、その出現順をトランザクション間の順序として利用するものである。トランザクション間の順序を、トランザクションを頂点とするグラフで表現したとき、そのグラフが閉路を持たない限り、その履歴は直列化可能であることが知られている。SGTアルゴリズムでは、1つの命令を実行許可した場合に、そのグラフを更新し、グラフに閉路ができなければその命令を許可し、閉路ができればどれか1つのトランザクションをアボートする。

BBアルゴリズムは、各実体ごとに、トランザクション間の順序を示すグラフを管理し、こうしたグラフによって規定される順序に閉路ができるかどうかを調べる。実体ごとに順序関係を表現するためより細かな制御ができ、blind WRITE間の出現順序を利用するためオンラインでグラフの更新ができる。また、実体毎のグラフは実際には特殊な形のグラフになるため、比較的単純な処理で実現できる。以下にBBアルゴリズムを示す。

データベースに初期値を書き込む仮想的トランザクションT_0の存在を仮定する。

［定義1］与えられた履歴hにおいて、特に実体xにアクセスする命令だけを考える。read-fromグラフG(h,x)=(N,E)を次のように定義する。Nは頂点集合で、1つの頂点はxにアクセスするトランザクションに対応している。Eは有向枝で、hが$W_i[x]\cdot\cdot R_j[x]$を部分列として含む（ただし、その間にW[x]はない）とき、T_j read-from T_iといい、(T_i,T_j)がEの要素になる。以後、read-fromグラフの頂点とトランザクションを区別しない。

一般にG(h,x)はいくつかの連結成分から構成され、各連結成分の根はxにblind WRITEを行ったトランザクションになる。このとき次の補題が成立する。

［補題1］hが直列化可能ならば、任意のxに関するread-fromグラフG(h,x)の連結成分C_iは木であり、C_iの同一レベルにある頂点のうちxにWRITEする頂点は高々1つしかない。また、頂点T_jの子を$T_{j1},T_{j2},\ldots,T_{jk}$とし、その内、$T_{j1}$だけがxにWRI

TEしているとする。このときｈと等価な直列履歴において、$T_{j2},\ldots,T_{jk}$はどれもT_{j1}に先行する。

［補題２］履歴ｈが直列化可能とする。任意のG(h,x)とその２つの連結成分C_i、C_jを考える。ｈと等価な直列履歴h'において、$T_m \rightarrow T_n$（$T_m \in C_i$、$T_n \in C_j$）であれば、任意の$T_u \in C_i$と任意の$T_v \in C_j$に対して$T_u \rightarrow T_v$が成り立つ。

実体ｘに関する直列化可能性判定を行うためのＢＢ判定グラフを定義する。

［定義２］履歴hと実体ｘに対して、read-fromグラフG(h,x)の連結成分間の順序を、各成分の根に対応したトランザクションが発行したblind WRITEのｈにおける出現順とする。このG(h,x)において、補題１、２によってえられる枝集合を付加して得られるグラフをｘのＢＢ判定グラフといいBB(h,x)と書き、ｈに出現するすべての実体ｘに関してｘのＢＢ判定グラフの和集合をｈのＢＢ判定グラフという。（グラフG(N,E)とG'(N',E')の和集合は(N∪N',E∪E')というグラフである）。

［例１］$h=R_1[x]R_2[y]R_3[x]W_2[y]W_1[x]R_5[y]R_2[x]R_4[y]R_5[x]R_6[y]W_4[x]R_6[x]W_6[y]$とする。read-fromグラフG(h,x)、ｘのＢＢ判定グラフBB(h,x)、およびｈのＢＢ判定グラフを図１～３に示す。

このとき次の補題が成り立つ。

［補題３］履歴ｈのＢＢ判定グラフが閉路を持たなければ、ｈは直列化可能である。ＢＢ判定グラフによる直列化可能性の判定は多項式時間で実行でき、ＢＢ判定グラフによって直列化可能と判定される履歴はクラスＣＳＲ[2,4]を含む。

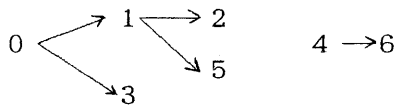

図１ 例１の履歴に対するG(h,x).

Fig.1 The read-from graph on x for the history of Example1.

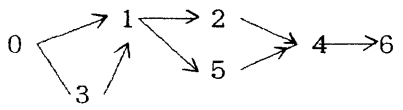

図２ 例１の履歴に対するBB(h,x).

Fig.2 The BB decision graph on x for the history of Example1.

ＢＢ判定グラフによる並行制御アルゴリズムは次のようである。

1) 実体ｘへのアクセス要求があればBB(h,x)に必要があれば枝を追加し、それに応じてＢＢ判定グラフにも枝を追加する。

2) ＢＢ判定グラフに閉路ができなければその要求を許可し、閉路ができれば、条件によって、その要求を待たせるか、あるいは、その要求を許可しかつ閉路に含まれるトランザクションの１つを後退復帰させる。

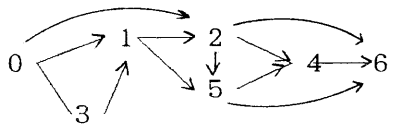

図３ 例１の履歴に対するBB判定グラフ.

Fig.3 The BB decision graph for the history of Example1.

３．不動点集合の比較

任意に発生させた実行履歴のうち、２相ロックアルゴズム、ＢＢアルゴリズムによってそれぞれ直列化可能と判定された履歴の割合を測定した。直列化可能と判定された履歴はこれらのアルゴリズムによってそのまま実行されるが、そうでないと判定された履歴は（実際には直列化可能であっても）、実行順序を変更して実行される。並行制御アルゴリズムによって、実行順を変更せずに実行される履歴集合をそのアルゴリズムの不動点集合と呼ぶ。

シミュレーションパラメータは次の通りである。

トランザクションの平均長：AV_LENGTH

トランザクション長の標準偏差：AV_LENGTH*0.2

トランザクション数：TR_NO

読出し専用トランザクションの比率：RATE_OF_R

読み書きを行うトランザクションにおける

READアクションの比率：　RATE_OF_R_IN_RW

blind WRITEの比率：　RATE_OF_B_IN_RW

データ数は1000、その内ホットスポットデータを200、ホットスポットデータのアクセス割合を80%とし、1000の履歴を発生させ、不動点集合に入った履歴の数を数えた。

AV_LENGTHを変化させた場合の実験結果を表1に示す。この実験ではTR_NO=10, RATE_OF_R=50%, RATE_OF_R_IN_RW=30%, RATE_OF_B_IN_RW=20%とした。

次にTR_NOを変化させた場合、各クラスのサイズがどの程度になったかの実験結果を表2に示す。表2の実験ではAV_LENGTH=10, RATE_OF_R=50%, RATE_OF_R_IN_RW=30%, RATE_OF_B_IN_RW=20%とした。

不動点集合の大きさで比較する限りでは、ＢＢアルゴリズムの方が２ＰＬより格段に優れている。しかし競合が激しくなるにつれ、不動点集合は急速に小さくなることがわかる。

表1　トランザクションの平均長を変化させた場合の不動点集合の大きさ
（トランザクション数＝１０）

Table1　Size of fixed points sets.
(The number of transactions is 10).

AV_LENGTH	5	10	15	20	25
２ＰＬ	819	373	53	6	2
ＢＢ	957	729	330	79	13

表2　トランザクション数を変化させた場合の不動点集合の大きさ
（トランザクション平均長＝１０）

Table2　Size of fixed points sets.
(The average length of transactions is 10).

TR_NO	10	20	30	40
２ＰＬ	373	18	0	0
ＢＢ	729	223	38	1

４．シミュレーションモデル

Agrawal[1]らはシミュレーションモデルによって全く逆の結果が生じ得ることが指摘し、並行制御アルゴリズムのためのシミュレーションモデルを提案し、その後このモデルが事実上の標準モデルと考えられてきている。ここでも原則的にこのモデルに従ってシミュレーションを行った。

トランザクション：WRITEを伴わないREAD、WRITEを伴うREAD、WRITEの列であり、それぞれの命令はある平均時間間隔をおいて発行されるものとした。新しいトランザクションは、同じ端末で実行された以前のトランザクションが終了した後、ある時間間隔をおいて発生するものとしている。

このシミュレーションで用いている論理モデルを図４に示す。物理モデルとしてＣＰＵは１台と仮定した。従って、ﾄﾗﾝｻﾞｸｼｮﾝの処理は逐次的であり、１つの処理を終えた後次の処理が行われる。I/Oサーバは無限と仮定した。時刻t_1で$A_i[x]$、時刻t_2で$A_j[y]$が実行されるとき、ﾄﾗﾝｻﾞｸｼｮﾝT_iの次の命令はt_1＋I/O時間以後に実行可能となり、T_jの次の命令はt_2＋I/O時間以後に実行可能となる。つまり２つの入出力は重畳して実行される。ただしｘとｙが同じ場合は、$t_1 \leqq t_2 \leqq t_1$＋I/Oであれば、T_jの実行はt_1＋I/Oまでblockされる。ここでI/Oとはデータの入出力にかかる時間である。

またマルチプログラミングのレベルも無限と仮定

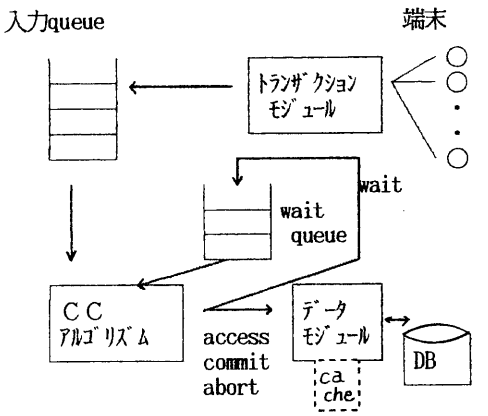

図４　論理モデル

Fig.4　The logical simulation model

した。従って、すべてのトランザクションは同じ優先度で並行して実行される。

5．実験結果

前節のモデルに基づいて、シミュレーションを行った。端末数とﾄﾗﾝｻﾞｸｼｮﾝの平均長以外のパラメータの値は次の通りである。

○端末当りのﾄﾗﾝｻﾞｸｼｮﾝの平均発生時間間隔 10秒
○１つのﾄﾗﾝｻﾞｸｼｮﾝ内での、平均I/O発生時間間隔
　３ミリ秒

ﾄﾗﾝｻﾞｸｼｮﾝに関する上記以外のパラメータは３．と同じである。ﾄﾗﾝｻﾞｸｼｮﾝはその値の書き込みをキャッシュに対して行い、コミットした時点でデータベースに一括して書き込むことにした。また、ＢＢアルゴリズムでは、どのﾄﾗﾝｻﾞｸｼｮﾝもそれに先行するﾄﾗﾝｻﾞｸｼｮﾝがコミットするまでコミットできないものとした。ＢＢアルゴリズムにおけるアボートは2PLv2と同じく、ループに含まれるﾄﾗﾝｻﾞｸｼｮﾝのなかで、もっとも処理の進んでいないトランザクションをアボートさせることとした。

○キャッシュサイズ　300
○ディスクI/O時間　100ミリ秒
○キャッシュI/O時間　１ミリ秒
○ﾄﾗﾝｻﾞｸｼｮﾝﾓｼﾞｭｰﾙ処理時間　１ミリ秒
○2PLｱﾙｺﾞﾘｽﾞﾑの処理時間　２ミリ秒
○BBアルゴリズムの処理時間　３ミリ秒

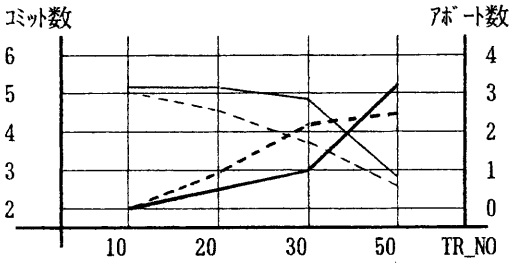

図５　１端末あたりのコミットしたﾄﾗﾝｻﾞｸｼｮﾝの数
　　１つのﾄﾗﾝｻﾞｸｼｮﾝ当たりのアボート回数
　　（平均ﾄﾗﾝｻﾞｸｼｮﾝ長=20）

Fig.5 The number of committed transactions and aborted tansactions (average length=20)

この仮定のもとで、あらかじめ各端末ごとに発生させておいたﾄﾗﾝｻﾞｸｼｮﾝ集合に対して、並行制御ｱﾙｺﾞﾘｽﾞﾑを適用し、３０秒間のシミュレーションを行って、その間にコミットされたﾄﾗﾝｻﾞｸｼｮﾝ数とアボートの回数を測定した。ＢＢｱﾙｺﾞﾘｽﾞﾑではアボートの連鎖が起こりえるが、その回数も算入している。以下で破線は2PLを実線はBB、細線はｺﾐｯﾄ数、太線はｱﾎﾞｰﾄ数を示す。

図５は平均ﾄﾗﾝｻﾞｸｼｮﾝ長が２０で端末数を変化させた場合、１端末あたりのｺﾐｯﾄしたﾄﾗﾝｻﾞｸｼｮﾝの数と１つのﾄﾗﾝｻﾞｸｼｮﾝ当たりのｱﾎﾞｰﾄ回数を示している。図６は端末数を３０とし、平均ﾄﾗﾝｻﾞｸｼｮﾝ長を変化させた場合、１端末あたりのコミットしたﾄﾗﾝｻﾞｸｼｮﾝの数と１つのﾄﾗﾝｻﾞｸｼｮﾝ当たりのアボート回数を示している。図７は端末数を変化させた場合の、システム内のｺﾐｯﾄﾄﾗﾝｻﾞｸｼｮﾝ数を示している。

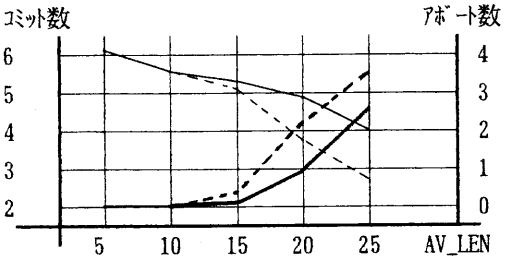

図６　１端末あたりのコミットしたﾄﾗﾝｻﾞｸｼｮﾝの数
　　１つのﾄﾗﾝｻﾞｸｼｮﾝ当たりのアボート回数
　　（端末数=30）

Fig.6 The number of committed transactions and aborted tansactions (terminals=30)

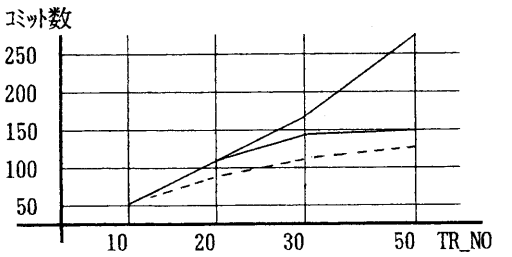

図７　システム内のｺﾐｯﾄﾄﾗﾝｻﾞｸｼｮﾝ数

Fig.7 Total number of committed transactions

図５、６から明らかなように、競合の少ない場合には2PLアルゴリズムもBBアルゴリズムもその効率に大差がない。むしろアルゴリズムの処理時間の仮定から2PLｱﾙｺﾞﾘｽﾞﾑの方がほんの少し良いといえる。しかし、競合が激しくなるにつれ、2PLアルゴリズムの効率が急速に低下するのに比べて、ＢＢｱﾙｺﾞﾘｽﾞﾑは比較的良い効率を示し、アルゴリズムの差が顕著に現れてくる。2PLV1と2PLv2とでは、個別のケースでは１長１短ではあるが、概して2PLv2の方が良い効率を示している。

ＢＢと２ＰＬを比較すれば、２ＰＬの方がロックを用いてトランザクションの実行をブロックする場合が多いので、２ＰＬの方がアボートが少ないと考えていたが、結果はＢＢアルゴリズムの方が少なかった。

平均ﾄﾗﾝｻﾞｸｼｮﾝ長を固定して、端末数を変化させてｺﾐｯﾄしたﾄﾗﾝｻﾞｸｼｮﾝの総数の変化を見てみると、初めは端末数の増加に比例して増加してゆくが、次第に増加率が鈍り、ついには飽和状態になるように見受けられる（図７）。

６．まとめ

２．で示したＢＢアルゴリズムはアクセスする各実体毎にＢＢ判定グラフを作成するため、２相ロックアルゴリズムに比べて実体毎にＮバイトほど余分の記憶量が必要となる（Ｎはﾄﾗﾝｻﾞｸｼｮﾝ数）。従って、実体として余り小さな単位を仮定するのは現実的ではない。

並行制御ではどの単位で並行制御を行うかという議論があるが（粒度という）、粒度が小さい場合にＢＢアルゴリズムを適用するには多少工夫をする必要がある。１つはホットスポットデータについてはＢＢを、他の実体については2PLアルゴリズムを適用するというハイブリッドアルゴリズムである。この方法の効率はシミュレーションによれば両者の中間的な性能を示し、取り立てて見るべき結果にはならなかったが、実現面からみれば実際的かもしれない。もう１つの方法はアクセス頻度の小さい実体集合を分割し、その実体集合に対して１つのＢＢ判定グラフを用いることである。この場合、１つの集合に属する実体へのアクセスは実際には競合していなくても競合しているように扱われるため、１つのﾄﾗﾝｻﾞｸｼｮﾝでともにアクセスされる確率の高い実体同士を同じ集合に含めるように分割することが必要である。

分散ﾃﾞｰﾀﾍﾞｰｽに対してＢＢアルゴリズムを適用することも可能である。特にＢＢ判定グラフは１次元で表現できるため、全域的な競合情報を１箇所で管理しても通信量は大きくならない。分散ﾃﾞｰﾀﾍﾞｰｽでの効率測定は残されている。

［参考文献］

[1] Agrawal,R, Carey,M.J. and Livny,M.:Concurrency Control Performance Modeling:Alternatives and Implications, ACM TODS, vol.12, No.4,609-654, 1987.

[2] Bernstein,P.A.,Hadzilacos,V. and Goodman, N.:Concurrency Control and Recovery in Database Systems, Addison Wesley,1987.

[3] Frrg,A.A. and Ozsu,M.T.:Using Semantic Knowledge of Transactions to Increase Concurrency, ACM TODS, vol.14, No.4,503-525,1989.

[4] Papadimitriou,"The Theory of Database Concurrency Control",Computer Sci. Press,1986.

[5] Shasha,D., Llirbat,F., Simon,E. and Valduriez,P.:Transaction Chopping:Algorithms and Performance Studies, ACM TODS, vol.20, No.3,325-363,1995

[6] Wu,K-L., Yu,P.S. and Pu,C.:Divergence Control for Epsilon-Serializability, IEEE 9th Inter. Conf. on Data Engineering, 506-515,1992.

[7] 加藤、「分散データベース・システムにおける同時実行制御方式の性能評価」、情報処理学会論文誌、３３巻１１号、1361-1371,1992年

[8] 徐、古川、史、「一貫性情報を用いたデータベースの並行処理制御」、情報処理学会論文誌、３５巻１２号、2752-2761,1994年12月

[9] 中津、「直列化可能性の新しい判定法」信学論 vol. J76-D-I, no.6, 1993.