マルチメディア通信と分散処理　89－2
（1998．6．3）

逐次化グラフを用いた複合トランザクションの並行制御

多田 知正　　樋口 昌宏　　藤井 護

大阪大学 基礎工学部 情報科学科

近年のデータベースシステムで多く見られる処理時間の長いトランザクションに関しては，トランザクション内部における並行実行や，アボート時の再実行のオーバーヘッドの削減といった要求がある．このような要求を満たすトランザクションモデルとして，入れ子トランザクションモデルが提案されている．入れ子トランザクションは入れ子の形で複数の部分トランザクションを含む．各部分トランザクションは並行実行が可能で，アボート及び再実行を独立に行うことができる．しかし部分トランザクションはデータベースの無矛盾性を保存しないため，スケジューリングの際には独立したトランザクションとして扱うことはできず，これが並行性の向上を妨げる要因となることがある．そこで本稿では，複合トランザクションモデルを提案する．複合トランザクションは，データベースの無矛盾性を保存する複数の要素トランザクションから構成されている．要素トランザクションは独立にスケジューリングを行うことが可能であり，これにより高い並行性が期待できる．そこで並行性が高いことで知られる逐次化グラフ検査に基づいたスケジューリングアルゴリズムを複合トランザクションに適用した．また，シミュレーションにより入れ子トランザクションモデルとの比較を行ない，複合トランザクションモデルの有効性を確認した．

A Concurrency Control of Multitransaction using Serialization Graph Testing

Harumasa Tada　　Masahiro Higuchi　　Mamoru Fujii

Depertment of Information and Computer Science, Faculty of Engineering Science
Osaka University

In recent database systems, many transactions have long processing time. For such transactions, concurrent execution inside a transaction and suppression of the overhead for restart after abortion are desirable. In order to solve such problems, the nested transaction model is proposed. A nested transaction consists of several subtransactions and has a hierarchical structure. Subtransactions can be executed concurrently. Moreover, they can be aborted and restarted independently. However, subtransactions are not guaranteed to preserve database consistency. Hence they should not be scheduled independently and it damages concurrency of transactions. Therefore, we propose a new model which we call the multitransaction model. A multitransaction consists of several member transactions which are guaranteed to preserve database consistency. Member transactions can be scheduled as independent transactions. This fact causes, we consider, higher concurrency of transactions. We applied Serialization Graph Testing, which is a scheduling algorithm which achieves high concurrency of transactions. We compared the multitransaction model with the nested transaction model through simulations. Simulation results showed the usefulness of the proposed model.

1. まえがき

近年のデータベースシステムの特徴として，処理時間の長いトランザクションが存在することがあげられる．このようなトランザクションにおいて処理時間を短縮するためには，トランザクション内部において複数の操作を並行に実行することが有効な場合がある．また処理時間の長いことは中断された時の再実行のオーバーヘッドが大きくなることを意味し，これを削減することが望まれる．このような要求を満たすトランザクションモデルとして，入れ子トランザクションモデルが提案されている[1]．入れ子トランザクションは入れ子の形で複数の部分トランザクションを含むトランザクションであり，部分トランザクションの並行実行が可能である．また，部分トランザクション単位でのアボートおよび再実行を行うことで，オーバーヘッドを小さくすることができる．入れ子トランザクションのスケジューリングについてはこれまでもいくつか研究が行われている[2, 3, 4]．入れ子トランザクションモデルにおいては，各部分トランザクションの実行はデータベースの無矛盾性を保存することを保証していないため，スケジューリングの際には，入れ

子トランザクション全体を一つのトランザクションとして扱うことになる．このため，どのようなスケジューリングアルゴリズムを導入してもトランザクションの並行性が高くならないという問題がある．そこで本論文では，より高い並行性が得られる複合トランザクションモデルを提案する．複合トランザクションはデータベースの無矛盾性を保存する複数の要素トランザクションからなり，要素トランザクション単位でスケジューリングを行なうことが可能である．そこで，並行性が高いことで知られる逐次化グラフを用いたスケジューリングアルゴリズムを入れ子トランザクションと，複合トランザクションに適用し，シミュレーションにより，それぞれのスループットの比較を行なう．

本稿では以降，2. で一般的なデータベースシステムの概要と逐次化グラフスケジューリングについて説明する．3. では入れ子トランザクションモデルについて述べ，4. では複合トランザクションモデルについて説明する．5. では複合トランザクションモデルにおけるスケジューリングアルゴリズムを示す．6. では入れ子トランザクションと複合トランザクションのシミュレーションによる比較を行い，7. でまとめを行う．

2. 準備

2.1 データベースシステム

データベースシステム (DBS) はデータベースを操作する命令を持つハードウェアおよびソフトウェアモジュールの集合である．データベースは，データ項目 (data item) の集合であり，それぞれのデータ項目は値を持っている．データ項目の値の集合がデータベースの状態を表している．データベースの状態集合のある部分集合は矛盾のない状態と定義される．

分散データベースシステムとは複数のデータベースシステムがネットワークで結合されたものである．分散データベースに含まれるそれぞれのデータベースをサイトという．

トランザクションとはデータベースを操作するプログラムの実行のことである．トランザクションは読み出し操作と書き込み操作によってデータベースを操作する．トランザクションの実行の開始は開始操作を実行することでによってデータベースシステムに伝えられる．またトランザクションはコミット操作またはアボート操作によって実行の終了を伝える．コミット操作により，トランザクションが正常に終了し，その結果を保存することをデータベースシステムに伝える．アボート操作は，トランザクションが異常な状態で終了したため，その結果をすべて無効にしなければならないことを伝える．

2.2 トランザクション

トランザクションは，読み出し，書き込み操作によってデータベースを操作するプログラムの実行である．トランザクションは読み出し，書き込み操作の実行される順序を表し，コミットもしくはアボートで終わる．

トランザクション T_i によるデータ項目 x の読み出し (書き込み) を $r_i[x]$ ($w_i[x]$) で表し，T_i の開始操作を s_i，コミット (アボート) を $c_i(a_i)$ で表す．1 つのトランザクション中には c_i と a_i はどちらか一方しか存在しない．トランザクションの並行実行において，開始されたがコミットもアボートもされていないトランザクション Tは実行中 (active) であるという．また，実行中のトランザクション Tの全ての読み出し操作および書き込み操作がすでに実行されているとき，Tはコミット待ち状態 (waiting commit) であるという．

トランザクションは単独で実行された場合に，データベースの無矛盾性を保存する．これをトランザクションの一貫性という．複数のトランザクションを並行に実行する場合，それらのデータベース操作の相互作用 (interfere) により，データベースの無矛盾性が破壊されることがある．これを避けるためにトランザクションの実行を操作の実行順序を制御する必要がある．これをトランザクションの並行制御，またはスケジューリングという．

定義 1: 異なるトランザクション T_i, T_j 中の操作 o_i, o_j が，同じデータ項目に対する操作で，少なくとも一方が書き込みである場合，o_i, o_j は衝突するという． □

定義 2: トランザクション T_i が書き込んだデータ x の値を T_j が読み出したとき，T_j は T_i を待つ (T_j waits for T_i) という． □

トランザクション Tが T'を待っているとき，T'がアボートされると，Tもアボートされることになる．トランザクション Tがコミット待ち状態であり，かつ Tが待つ全てのトランザクションがすでにコミットしているとき，Tはコミットされる．

2.3 アトミック性

トランザクションが正常終了した場合，そのトランザクションがデータベースに及ぼした影響は全て保存されなければならず，また異常終了したトランザクションはデータベースには全く影響を及ぼしてはならない．これをトランザクションのアトミック性という．

トランザクション T_iがコミットされると，T_iが更新したデータはすべて不揮発性の記憶に保存される．またトランザクション T_iがアボートされると，T_iが更新したデータはすべて更新前の値に戻され，T_iを待つトランザクションは全てアボートされることになる．このように，あるトランザクションのアボートが他のトランザクションのアボートを引き起こすことを連鎖アボート (cascading abort) という．

2.4 逐次化可能性

トランザクションの並行実行の正当性の基準として逐次化可能性 (serializability) と呼ばれる性質が一般に用いられる [5]．並行実行 Eについて，ある逐次的実行 E_sが存在し，E中の全ての衝突する操作の対 (o_i, o_j) について，o_iと o_jの実行順序が Eと E_sで同じであるとき，Eは逐次化可能であるという．並行制御を行う際には，その結果得られる実行が逐次化可能性を満たすように行われる．

2.5 逐次化グラフ

トランザクションの並行実行の逐次化可能性は，その実行から得られる逐次化グラフ (serialization graph) を検索することで確かめられる．逐次化グラフとは並行実行から導かれる次のような有向グラフである．逐次化グラフの頂点は並行実行に含まれるトランザクションである．また，2 つのトランザクション T_i,T_jの操作 o_i,o_jが衝突しており，o_iが o_jより先に実行されているとき，頂点 T_iから頂点 T_jへの有向辺 (衝突辺と呼ぶ) が存在する．

逐次化グラフについて次の定理が証明されている．

定理 1: 並行実行が逐次化可能であるための必要十分条件は，その実行から得られる逐次化グラフが閉路を持たないことである [5]． □

2.6 逐次化グラフスケジューリング

トランザクションのスケジューリングアルゴリズムの一つに，逐次化グラフ検査を用いたスケジューリングが提案されている．逐次化グラフ検査を用いたスケジューリングは従来のデータベースで多く用いられている二相ロックや時刻印を用いたスケジューリングよりも高い並行性を実現できることが知られている．

逐次化グラフスケジューリングは以下のようにして行なわれる．

- スケジューラはトランザクション T_iの開始操作 s_i を受けとると，トランザクション T_i に対応する頂点を逐次化グラフに加える．
- トランザクション T_iの読みだし操作または書き込み操作 $p_i[x]$ を受けとると，$p_i[x]$ と衝突していてすでに実行されている操作 $q_j[x]$ を含むトランザクション T_jに対して，逐次化グラフに衝突辺 (T_j, T_i) を加える．

ここで逐次化グラフの状態により２つの場合が考えられる．

1. 逐次化グラフが閉路を含まない場合．
   スケジューラは $p_i[x]$ を実行する．
2. 逐次化グラフが閉路を含む場合．
   このとき，並行実行は逐次化可能ではないので，スケジューラは閉路を形成しているトランザクションのうちいくつかのトランザクションをアボートし，それらに対応する頂点と隣接する辺を取り除くことで逐次化グラフを閉路のない状態にする．

- スケジューラはトランザクション T_iのコミット操作 c_iを受けとると，T_iの待つトランザクションが全てコミットするのを待って，T_iをコミットする．

3. 入れ子トランザクションモデル

3.1 入れ子トランザクション

近年のデータベースでは処理時間の長いトランザクションが見られる．そのようなトランザクションにはトランザクション内で並行実行を行うことで処理時間を短縮できるものがある．また，あるトランザクションがアボートされた場合，そのトランザクションのそれまでの処理を全て無効にし，必要ならば最初から再実行を行う．処理時間の長いトランザクションでは，この再実行のオーバーヘッドが大きくなるという問題があり，これを削減することがのぞまれる．これらの要求を満たすために提案されたモデルが入れ子トランザクションモデル (nested transaction) である．

入れ子トランザクションは部分トランザクションの集合である．部分トランザクションはその内部に部分トランザクションを生成することが出来る．すなわち入れ子トランザクションは階層構造を持つ．図 1に入れ子トランザクションの例を示す．

部分トランザクション Tが，部分トランザクション T' を生成したとき，Tは T'の親，T'は Tの子であるという．またこのとき，Tは T'の先祖であり，T'は Tの子孫である．トランザクション Tの先祖の親も Tの先祖であり，トランザクション Tの子孫の子も Tの子孫である．

親が存在しない部分トランザクションをトップトランザクションという．

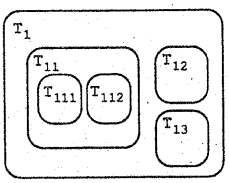

図 1　入れ子トランザクションの例

入れ子トランザクションでは，ある部分トランザクションがアボートした際に，その部分トランザクションとその子孫のみを再実行するだけでよい．その他の部分トランザクションは影響を受けず，これにより再実行のオーバーヘッドを小さくすることができる．

またトランザクション内部において衝突する操作を並行に実行することは通常許されないが，入れ子トランザクションでは，部分トランザクションに分割することで衝突する操作の並行な実行が可能になり，トランザクション内部の並行性を高めることができる．

入れ子トランザクションは分散データベースシステムにおいてとくに有効である．分散データベースシステムにおいて，複数のサイトで実行されるトランザクション Tを考える．あるサイトで Tの処理が失敗した場合，T はアボートされ，他のサイトで実行中の Tの結果も全て無効にしなければならない．これはアボートがサイト間の通信を伴うことを意味し，大きなオーバーヘッドとなる．そこで Tを入れ子トランザクションとして実装し，それぞれのサイトでは部分トランザクションが実行されるようにすると，あるサイトで部分トランザクションがアボートした場合，他のサイトで実行中の部分トランザクションは影響を受けないので，通信を行わず局所的にアボートと再実行を行うことができる．

3.2 部分トランザクション

入れ子トランザクションにおいて，部分トランザクションは一貫性を満たす必要がない．例えば，口座 xから口座 yへ 1000 円移すトランザクション Tを考える．Tを xから 1000 円引き出す部分トランザクション T_1と yへ 1000 円振り込む部分トランザクション T_2 を含む入れ子トランザクションとして実装することができる．この場合，入れ子トランザクション T全体では，データベースの無矛盾性は保存されるが，T_1もしくは T_2 のみが実行された場合，データベースの無矛盾性は破壊されてしまう．このように部分トランザクション自体は一般に一貫性を満たさないので，厳密にはトランザクションではない．したがってスケジューリングの際には，部分トランザクションを独立に扱う事はできず，入れ子トランザクションの一部であるとみなされる．

3.3 コミットとアボート

入れ子トランザクションは，部分トランザクション単位のアボート，再実行が可能である．部分トランザクションがアボートされるとその部分トランザクションとすべての子孫がアボートされ，必要ならば再実行される．

ただし入れ子トランザクションでは部分トランザクション単位のコミットは行うことはできない．コミットは以下のような手順で行われる．

定義 3: 子を持たない部分トランザクション Tがコミッ

ト待ち状態であり，かつTが待つ全ての部分トランザクションがすでにコミットされているとき，Tはコミット可能 (committable) であるという．□

定義 4: 部分トランザクションTがコミット待ち状態であり，かつTが待つ全ての部分トランザクションがすでにコミットされており，Tの子が全てコミット可能であるとき，Tはコミット可能 (committable) であるという．□

コミット可能である部分トランザクションTの親がアボートされると，Tはアボートされる．入れ子トランザクションはその中の全ての部分トランザクションがコミット可能になった後，全体としてコミットされ，全ての部分トランザクションはその時点でコミットされることになる．

4. 複合トランザクションモデル

4.1 目的

Tを分散データベースシステムで複数のサイトで実行されるトランザクションとする．3.1で述べたように，Tを入れ子トランザクションとして実装し，それぞれのサイトで部分トランザクションが実行されるようにすることで，アボートの際のオーバーヘッドを削減できる．ここで，Tの部分トランザクションがすべて一貫性を満たす場合を考える．この場合，各部分トランザクションを別々のトランザクションとしてスケジューリングすることで，高い並行性を得ることができる．しかし3.2で述べたように，入れ子トランザクションではこのようなスケジューリングは不可能である．一方，部分トランザクションはすべて一貫性を満たしているので，入れ子トランザクションを用いず，Tを複数のトランザクションの集合として実装することも可能である．しかしこの場合，あるサイトのトランザクションはコミットされたにもかかわらず，他のサイトのトランザクションはアボートされ，永久にコミットされないという状況が起こり得る．これは，Tの実行結果の一部分のみが保存されるという状態を意味し，Tのアトミック性が破壊される．これらの問題を解決するには，トランザクションに含まれる各部分トランザクションを別々にスケジュールでき，かつトランザクションのアトミック性を保証するようなトランザクションモデルが必要となる．

そこで，一貫性を保証するトランザクションの集合からなる複合トランザクションモデルを導入する．

4.2 複合トランザクションモデル

複合トランザクションは一貫性を満たすトランザクションの集合である．複合トランザクションに含まれるトランザクションを要素トランザクションという．また，複合トランザクションはアトミック性を満たす．すなわち，複合トランザクションに含まれるある要素トランザクションがコミットされたとき，他のすべての要素トランザクションもコミットされる．要素トランザクションは他の要素トランザクションを生成することが出来る．要素トランザクションTが，要素トランザクションT'を生成したとき，TはT'の親，T'はTの子であるという．またこのとき，TはT'の先祖であり，T'はTの子孫である．トランザクションTの先祖の親もTの先祖であり，トランザクションTの子孫の子もTの子孫である．要素トランザクションは子を生成する時，パラメータを与えることができる．パラメータは子の生成時にのみ渡すことができ，それ以外の要素トランザクション間の通信は許されない．

4.3 コミットとアボート

複合トランザクションのコミットとアボートは，入れ子トランザクションの場合とほぼ同様である．複合トランザクションにおいても，要素トランザクション単位のアボートおよび再実行が可能である．

複合トランザクションは要素トランザクション単位のスケジューリングが可能であるが，要素トランザクション単位のコミットは出来ない．コミットは入れ子トランザクションと同様に以下のような手順で行われる．

定義 5: 子を持たない要素トランザクションTがコミット待ち状態であり，かつTが待つ全ての要素トランザクションがすでにコミットされているとき，Tはコミット可能 (committable) であるという．□

定義 6: 要素トランザクションTがコミット待ち状態であり，かつTが待つ全ての要素トランザクションがすでにコミットされており，Tの子が全てコミット可能であるとき，Tはコミット可能 (committable) であるという．□

要素トランザクションTが待つ全ての要素トランザクションがコミット可能になった後，Tはコミット可能になる．要素トランザクションはその中の全ての要素トランザクションがコミット可能になった後，全体としてコミットされ，全ての要素トランザクションはその時点でコミットされることになる．

5. 複合トランザクションのスケジューリングアルゴリズム

5.1 逐次化グラフスケジューリングの適用

複合トランザクションに逐次化グラフを用いたスケジューリングを適用することを考える．要素トランザクションは一貫性を満たすので，一つのトランザクションとしてスケジューリングを行なうことが可能である．したがって，逐次化グラフには各要素トランザクションに対応した頂点が存在する．操作の衝突する要素トランザクション間には，通常のトランザクションの場合と同様に衝突辺を引く．また，4.2で述べたように，要素トランザクションTが子T'を生成する時にパラメータを与える場合がある．このとき，実際にデータベースへの読み書きは行わないがT'はTの処理したデータを利用しているので，T'がTより前に実行されるような逐次的実行は，もとの並行実行とは等価ではない．このことを反映して，Tからその子T'にパラメータを渡す場合は，TからT'へ衝突辺を引く必要がある．逐次化グラフの探索は通常のトランザクションの場合と同様に行われる．

5.2 アボートの減少

入れ子トランザクションの逐次化グラフスケジューリングでは，入れ子トランザクションが逐次化グラフの頂点に対応する．一方複合トランザクションでは，要素トランザクションごとに頂点が存在する．これにより，複合トランザクションではアボートの回数が減少する．たとえば，図2のような例を考える．入れ子トランザクションT_1,T_2と，複合トランザクションM_1,M_2は全く同じ動作を行うとする．今部分トランザクションT_{11}とT_{21}，T_{12}とT_{22}がそれぞれ衝突している．この場合，入れ子トラ

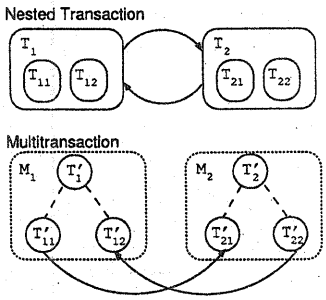

図 2　アボートの減少

ンザクションでは，図に示したように入れ子トランザクション T_1 と T_2 の間に閉路が存在することになる．すなわち T_1 と T_2 のいずれかはアボートされなければならない．一方，複合トランザクションでは，T'_{11} から T'_{21}，T'_{12} から T'_{22} に辺が引かれ，この場合閉路はできない．このように，複合トランザクションでは，要素トランザクション単位でスケジューリングを行えるために，トランザクションのアボート回数は減少する．このことはすなわち，複合トランザクションが入れ子トランザクションよりも高い並行性を得られることを意味している．

5.3　待ちの循環

トランザクションは別のトランザクションを待つことがある．T が T' を待っている時，T' がコミットするまでは，T はコミットできない．もし T' が T を待っていれば，T は永久にコミットできない事になる．このような状態を待ちの循環と呼ぶ．

しかし，通常のトランザクションの逐次化グラフスケジューリングでは，トランザクション T が T' を待つ場合，常に T' から T へ衝突辺が引かれる．したがって，待ちの循環が生じるとき，逐次化グラフでは閉路が出来ることになる．閉路が検出されると，閉路に含まれるいずれかのトランザクションがアボートされるので，待ちの循環は問題にならない．

一方，入れ子トランザクションや複合トランザクションの場合は，部分トランザクションや要素トランザクションは他の全てのトランザクションがコミット可能になるまでコミットできないという制限がある．入れ子トランザクションの逐次化グラフスケジューリングでは，部分トランザクションは入れ子トランザクションの一部とみなされるために問題とならないが，要素トランザクションが独立したトランザクションとして扱われる複合トランザクションの逐次化グラフスケジューリングでは，上の制限は逐次化グラフには反映されないために，逐次化グラフでは検出できない待ちの循環が生じる可能性がある．

例えば，図2の例で，T'_{12} が T'_{22} を待っており，T'_{21} が T'_{11} を待っているものとする．この例の場合，T'_{11} は T'_{12} がコミット可能になるまでコミットできず，T'_{22} は T'_{21} がコミット可能になるまでコミットできない．しかし，T'_{12} は T'_{22} がコミットしなければ，コミット可能にはならず，T'_{21} は T'_{11} がコミットしなければコミット可能にならない．このように，複合トランザクションにおいては，待ちの循環によりコミットができなくなる場合が生じる．

5.4　待ちグラフ

5.3で述べた問題を解決するために，待ちグラフ (Wait-for Graph) というグラフに基づいてコミットを行う．待ちグラフは複合トランザクション間の待ちの関係を表す．

待ちグラフの頂点は並行実行中の複合トランザクションである．複合トランザクション M の要素トランザクション T が別の複合トランザクション M' の要素トランザクション T' を待つとき，頂点 M から頂点 M' に有向辺が存在する．待ちグラフはスケジューリングには影響しない．すなわち，待ちグラフに閉路が存在しても，複合トランザクションをアボートする必要はない．

定義 7: 複合トランザクション M の全ての要素トランザクションがコミット待ち状態であるとき，M はコミット待ち状態であるという．□

複合トランザクション M がコミット待ち状態になると，待ちグラフを探索し，M から到達可能なすべての複合トランザクションを調べる．もしそれらすべてがコミット待ち状態であるならば，それらは M を含めてすべてコミットされる．M を含む閉路が存在する場合，閉路中の複合トランザクションもすべてコミットされることになる．もし到達可能なコミット待ち状態でないトランザクションが一つでも存在すれば，M はコミットされない．

6.　シミュレーション

複合トランザクションモデルと提案するスケジューリングアルゴリズムの有効性をシミュレーションによって検証する．

全く同じデータにアクセスする入れ子トランザクションと複合トランザクションに対して，ともに逐次化グラフを用いたスケジューリングを行い，スループットを比較する．シミュレーションにおいては，各トランザクションは同期して実行されるものとし，この同期の単位をステップと呼ぶ．

スケジューラが並行に処理するトランザクションの数をマルチプログラミングレベルという．このマルチプログラミングレベルをパラメータとして変化させる．シミュレーションは 1 回につき 50000 ステップとし，1000 ステップ当たりのコミットしたトランザクションの数の平均をスループットとして求める．スループットの他に，トランザクションがアボートした回数を求める．

6.1　シミュレーションの仮定

シミュレーションにおいて生成される入れ子トランザクションは，トップトランザクションと 2 つの部分トランザクションから構成されている．トップトランザクションと各部分トランザクションはそれぞれ，2000 のデータ項目の中からランダムに選んだ 10 のデータ項目にアクセスする．

複合トランザクションは対応する入れ子トランザクションと同様の構造をもつものとする．すなわち，入れ子トランザクションのトップトランザクションと同じデータにアクセスする要素トランザクションが，2 つの部分トランザクションと同じデータにアクセスする 2 つの要素トランザクションを生成する．

データの読み出しおよび書き込み操作はそれぞれ 10 ステップかかるものとする．トランザクションのアボートのオーバーヘッドは 50 ステップとする．また，アボートされたトランザクションは必ず再実行を行なうものとする．

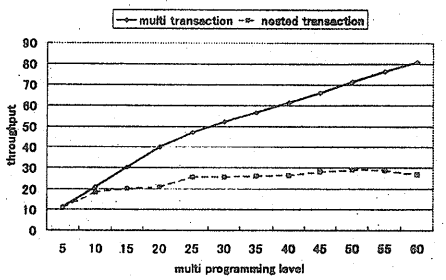

図 3　スループット

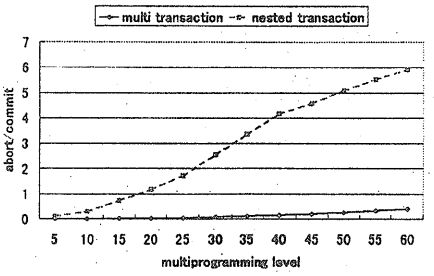

図 4　トランザクションあたりのアボート回数

6.2　シミュレーション結果

スループットを図3に示す．両者ともマルチプログラミングレベルの増加に伴って，スループットは上昇して行くが，上昇は次第に緩やかになる．これは，マルチプログラミングレベルの増加により，トランザクションの衝突が頻繁に起こり，その結果トランザクションのアボートが増加するためであると考えられる．とくに入れ子トランザクションにおいては，マルチプログラミングレベルが 25 を超えると，スループットはほぼ横ばいとなり，複合トランザクションと比べてかなり低いものとなっている．

アボートの回数をコミットしたトランザクションの数との比の形で表したものが図 4である．入れ子トランザクションと複合トランザクションでアボートの回数に著しい差が生じていることがわかる．5.2で述べたように，複合トランザクションでは，入れ子トランザクションに比べて，トランザクションのアボートが減少することは予想されたが，実際にはアボート回数の差は予想以上のものとなった．これは連鎖アボートの影響によるものであると考えられる．トランザクションがアボートされる際，そのトランザクションを待つすべてのトランザクションもアボートされる．複合トランザクションの場合では個々の要素トランザクションを待つ要素トランザクションはそれほど多くないが，入れ子トランザクションの場合はすべての部分トランザクション同士の待ちの関係が，入れ子トランザクション同士の待ちの関係となるため，入れ子トランザクションを待つ入れ子トランザクションはかなり多くなる．したがって，入れ子トランザクションがアボートされると，多くの入れ子トランザクションが連鎖アボートされる．連鎖アボートされたトランザクションがさらに連鎖アボートを引き起こすため，結果的に非常に多くの入れ子トランザクションがアボートされることになる．本シミュレーションにおいては，部分トランザクションの数が 2 という比較的単純な構造の入れ子トランザクションを取り上げたが，図 4のようにアボート数の差は非常に大きなものとなった．より複雑なトランザクションにおいては，この差はさらに大きくなるものと考えられる．

7.　まとめ

本稿では，一貫性を満たすトランザクションの集合からなる複合トランザクションモデルと，逐次化グラフを用いたスケジューリングアルゴリズムを提案した．入れ子トランザクションに含まれる部分トランザクションは一貫性を満たすとは限らないため，すべての入れ子トランザクションが複合トランザクションとして実装できるわけではないが，可能な場合には，より高い並行性を得るために有効であると考えられる．

シミュレーションの結果，マルチプログラミングレベルが高い状況において，入れ子トランザクションより非常に高いスループットが得られるということがわかり，複合トランザクションの有効性が確かめられた．

References

[1] J. Moss, B. Eliot: “Nested Transaction: An approach to reliable distributed computing”, MIT Press, (1985).

[2] L. Daynes, O. Gruber and P. Valduriez: “Locking in OODBMS Client Supporting Nested Transaction ”, IEEE the 11th intl. conf. on Data Engeneering, pp.316-323 (1995).

[3] S. Ben-Hassen and M. Rusinkkiewicz: “On Serializability of Distributed Nested Transaction”, IEEE The 12th intl. conf. on Distributed Computing Systems, pp.152-159 (Jun. 1992).

[4] C. Beeri, P. A. Bernstein and N. Goodman: “A Model for Concurrency in Nested Transactions Systems” Journal of the Association for Computing Machinary, Vol.36, No.2, pp.230-268 (Apr. 1989).

[5] P. A. Bernstein, V. Hadzilacos and N. Goodman: “Concurrency Control and Recovery in Database Systems”, Addison-Wesley, (1987).