計算機アーキテクチャ 125－22
（1997．8．21）

投機的実行を用いたデータベース処理 ～マルチトランザクション環境下での高速化～

佐々木 敬泰　高山 毅　弘中 哲夫　藤野 清次

広島市立大学 情報科学部 情報工学科

本稿では，著者らが提案している投機的問合せ処理をマルチトランザクション環境下で高速に行う手法を提案する．投機的問合せ処理とは，ユーザが検索条件を入力する前の検索条件を考慮している間に，投機的に問合せ処理を開始することにより応答時間を短縮するものである．従来の実装方式では，投機のためのプロセス生成を動的に行っていたため、マルチトランザクション環境下において，通信/OSのオーバヘッドの累積が無視できず，応答時間が必ずしも十分短いとはいえなかった．本稿では，プロセスの起動を静的に行うことにより，上記オーバヘッドを低減することで高速化を行う．シミュレーション・プログラムを用いた評価によると，本稿の提案手法では，従来よりも応答時間の短縮が図れることがわかった．

Database Processing with Speculative Execution –Reduction of Response Time in Multi-transactions Environments–

Takahiro SASAKI　Tsuyoshi TAKAYAMA
Tetsuo HIRONAKA　Seiji FUJINO

Department of Computer Engineering, Faculty of Information Sciences,
Hiroshima City University

This paper proposes a methodology in order to reduce a response time of *speculative query processing* in multi-transactions environments. The speculative query processing is a technique, we propose, to reduce a response time. That is, the DB system starts to process, in parallel, some candidate queries corresponding to their distinct selection conditions before a single true selection condition is inputted. This paper proposes an effective algorithm for multi-transactions environments. With the algorithm, it is possible to keep the overheads down on communications and OS, and to reduce a response time. According to our experiments, this algorithm is more effective than the conventional method.

1　はじめに

近年，データベース（以後DBと略す）の大規模化が進行し，またマルチメディアなどの多様な形態の情報を多人数で共有・再利用することに対する要求が高まっている．これらの要求を満たすべく，オブジェクト指向データベース（Object-Oriented Database;OODB）が用いられている．応答時間の短縮のために問合せ処理の並列化は効果的であるが，OODBでは，負荷分散が容易ではなく，アイドル状態のプロセッサが発生することが少なくない．そこで，著者らはこれらのアイドル状態のプロセッサを用いて，入力される確率が相対的に高い複数の検索条件に対して，問合せ処理を投機的実行 [Yamana95] する手法を提案している [Takayama97-1].

しかし，文献 [Takayama97-1] では，プロセスの生成を動的に行っていたため，プロセスの生成にともなう通信/OSのオーバヘッドが大きく，マルチトランザクション環境下において十分な性能が得られなかった．そこで，本稿では上記オーバヘッドを低減し，マルチトランザクション環境下でも問合せ処理が高速化できる手法を提案する．シミュレーション・プログラムを用いた評価によると，ある環境下では問合せ処理時間が平均で4割程度短縮可能となった．

本稿は以降，次のように構成されている．まず，次節で著者らがこれまでに提案している投機的問合せ処理について概括する．3節で従来の実装方式とその問題点をまとめる．4節では，従来方式の問題を低減することを目指し，プロセスを静的に生成する方式を提案し，5節でシミュレーション・プログラムを用いて評価を行う．最後に6節で結論と今後の展望を述べる．

2　投機的実行を用いた問合せ処理

本節では，本稿での議論に先立ち，文献 [Takayama97-1] で提案している『投機的問合せ処理』について概括しておく．

2.1 投機的問合せ処理導入の経緯

近年のマルチメディアDB，エンジニアリングDB，その他のいわゆる「DBの高度応用」は，扱うデータの複雑化，大規模化を進行させている．ここにおいて，OODBの利用が活発に行われており，また，大規模化したOODBに対して，検索処理の高速化が求められている．検索操作を高速に処理するために，問合せ処理の並列化は有効である．しかし，OODBではオブジェクトへのアクセスがオブジェクト識別子によってなされるために，オブジェクト数が均等になるように，あるクラスのオブジェクトをフラグメント化したとしても，すべてのオブジェクトをアクセスするために要するコストは全フラグメントについて均等になるとは限らない．そのため，OODBの問合せ処理の並列化における負荷分散はリレーショナルDBなどに比べ困難である．そこで，著者らは負荷分散の不均等などによって生じるアイドル状態のプロセッサを利用した，問合せ処理のもう一つの並列化手法として投機的問合せ処理を提案している．

2.2 投機的問合せ処理

投機的問合せ処理とは，検索画面を表示してからユーザが検索条件を入力するまでの間に，過去のDBのログ情報をもとに，入力されるであろう検索条件を予測し，投機的に問合せ処理を開始するものである．

一般に，DBの検索頻度にはデータによるバラつきが存在し，特定のデータに検索が集中することも少なくないという性質がある．DB検索における問合せ処理は，投機的実行導入の対象として適性がある．

2.3 対象とするOODB

本稿で用いるOODBについて文献 [Kim90] を基に概括する．図1は，OODBの具体例である．OODBでは図1のように現実世界を階層化して表現する．図の記法は先述の文献に基づく．図中の“A”～“D”はクラス名をあらわし，それぞれ局所的な属性“a”～“d”を持つ．各クラスは，自身より上位のクラスの属性をすべて継承する．すなわち，各クラスに所属するインスタンスはそれぞれ表1の属性に対する値を持つことになる．

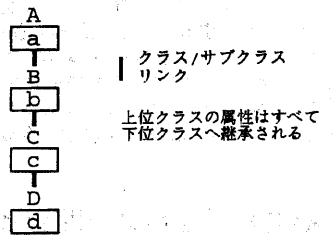


図 1: OODBの一例

また，文献 [Kim90] では，一つのオブジェクト

表 1: 各クラスに所属するインスタンスがもつ属性

クラス	属性
A	a
B	a,b
C	a,b,c
D	a,b,c,d

は，一つのクラスにしか属せないことを規定している．そのため，属性cに対する問合せが発生した場合，属性cに対する値をもつすべてのクラス，すなわちクラスC,Dを走査する必要がある．なお，本稿では簡単のためにクラス合成階層や多重継承については議論しない．

ここで本稿を通して共通する，二つの仮定を置く．

1. ユーザは検索条件の指定を，候補となる有限個の選択肢から選択することによって行う．
2. 検索処理のみを議論しており，ユーザが検索処理を行った場合のDBシステム側での問合せ処理過程においては，ロック操作を行わない．

3 投機的問合せ処理のアルゴリズム

投機的問合せ処理を実現するアルゴリズムとして，今まで著者らが提案してきた手法を述べ，その後，この方式の問題点をまとめる．

3.1 動的起動方式のアルゴリズム

投機的問合せ処理を行うアルゴリズムとして，著者らはこれまでに図2に示すものを提案している [Takayama97-2]．本稿では，図2のアルゴリズムを後述の方式と区別するため，「動的起動方式」と呼ぶ．

以下に動的起動方式のアルゴリズムの説明を行う．

step1 問合せ処理管理タスク（以後QMTと略す）は，検索条件となる属性群，すなわちキー属性群の確定時点で，問合せ処理施行タスク（以後QETと略す）を起動し，DBのログ情報を基に検索頻度が相対的に高い検索条件を送信する（<m1>-<m3>）．

step2 QMTから検索条件を受信したQETは検索を投機的に開始し，結果をファイルに出力する（<s1>-<s2>）．出力されたファイルは，（トランザクションID,検索条件ID）の組によって，一意に識別される．

step3 QMTは，有限個の選択肢からなる検索画面を表示し，ユーザから検索条件の入力を受け付ける（<m4>-<m6>）．

step4 すべての投機が失敗した場合には入力された検索条件に対応するQETを新たに起動し，ここから検索を開始する（<m7>,<s3>）．

step5 QMTは，ユーザ入力の検索条件に見合う検

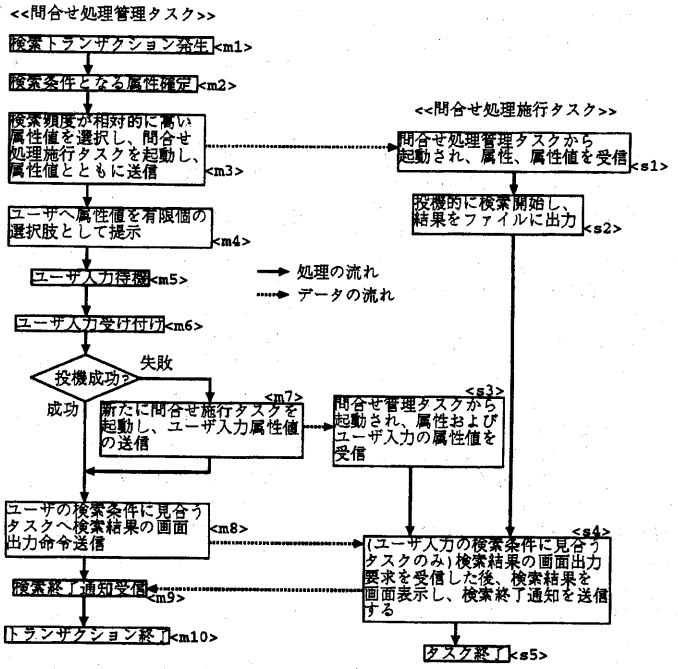


図 2: 動的起動方式のアルゴリズム

索をしているQETへ画面出力要求のメッセージを送る（<m8>）.

step6 QMTからこれを受信したQETは対応するファイルを走査して検索結果を画面出力する（<s4>）.

3.2 動的起動方式の問題点

動的起動方式では，以下にあげるような問題がある.

QET生成時の通信/OSのオーバヘッド

通常，プロセスの生成にはOSによる多大なオーバヘッドがかかる．しかし，問合せが発生するたびにメッセージ通信を用いて，リモートマシン上でQMT/QETの起動を行なっている.

投機失敗の場合の通信/OSのオーバヘッド

投機がすべて失敗した場合，新たにQETを生成し，発生させたQETに問合せ処理を行わせているため，ここでも通信/OSのオーバヘッドが生じる.

不要QETの放置

動的起動方式では，プロセス終了のオーバヘッドを回避するために，投機に失敗したQETを放置している．このままでは，DBシステムを長時間稼働させた場合，終了待ち状態のQETが蓄積されていき，DBシステムを含むシステム全体のトラブルの元となる危険性がある.

4 静的起動方式

本節では，前節で述べた問題点を解決することを目指し，プロセスの静的起動を用いた方式を提案する．その基本方針は以下の通りである.

- あらかじめQMT/QETのどちらにもなり得るプロセス起動しておく．トランザクションが発生したら，これに対し，QMT/QETのどちらかに機能させるためのメッセージを送り，各々の処理を開始する．QMT/QETを同一プログラムにすることはプログラムの巨大化を引き起こし，主記憶の不足による性能低下を引き起こす危険性があるが，最近のOSは仮想記憶方式を用いており，プログラムの必要な箇所のみを主記憶上にロードするため，主記憶の占有による性能低下は決定的打撃を与えるものとはならない.
- 投機がすべて失敗した場合，何もしていないQMT自身が問合せ処理を開始する.
- 処理の終ったQETは回収/再利用する.

4.1 静的起動方式のアルゴリズム

静的起動方式のアルゴリズムを図3に示す．注意すべきこととして，図中のラベル<s1>,<m1>等は，動的起動方式の図2に対応させてあるため，ラベルが不連続になっている部分がある．これは，静的起動方式では不要な処理を意味している．また，<m2'>のようにラベルにダッシュ（'）のついたものは静的アルゴリズム固有の処理を，<s4-1>のようにマイナ

ス (-) のついたものは動的アルゴリズムでは一つの処理として行われていたものが分割されたことを表す.

以下に図3に沿って, 静的起動方式のアルゴリズムの説明を行う.

step1 実プロセッサ上であらかじめ有限個のQMT/QETのどちらにもなり得るタスク (以後, このタスクの数を「仮想プロセッサ台数」と表現する) を立ちあげておき, 必要に応じてQMTあるいはQETとして動作させる. また, これらのタスクの管理はすべてタスク管理機構 (以後TMと略す) で行う.

step2 QMTは, 検索条件となる属性群, すなわちキー属性群の確定時点で, TMに空きタスクの使用予約を出し, 割り当てられたタスクに対し, QETとして動作するようメッセージを送り, DBのログ情報を基に検索頻度が相対的に高い検索条件を送信する (<s0'>,<m1>-<m3>,<t1>,<t2>).

step3 QMTから検索条件を受信したQETは検索を投機的に開始し, 結果をファイルに出力する (<s1>-<s2>). 出力されたファイルは, (トランザクションID,検索条件ID) の組によって, 一意に識別される.

step4 QMTは, 有限個の選択肢からなる検索画面を表示し, ユーザから検索条件の入力を受け付ける (<m4>-<m6>).

step5 すべての投機が失敗した場合には, 入力された検索条件に対してQMT自身が検索を始める. (<m7'>).

step6 投機が成功した場合には, QMTはユーザ入力の検索条件に見合う検索をしているQETへ画面出力要求のメッセージを送る. (<m8>).

step7 QMTからこれを受信したQETは対応するファイルを走査して検索結果を画面出力し, 新たなメッセージを待ち続ける (<s4-1>,<s4-2>).

step8 QMTは画面出力要求を出したQETから終了通知を受けとる (<m9>).

step9 QMTは自分に割り当てられたQETをTMに返却する (<m9'>,<m10>,<t3>,<t4>).

図3では, 説明を簡単にするため, QETの使用時のみTMにタスク使用予約を出しているが, 実際にはQMT自体もTMから割り当てられたタスク上で実行される.

5 静的起動方式の有効性に関する第一次評価

動的起動方式と静的起動方式をシミュレーション・プログラムを用いて評価する.

5.1 評価に用いるOODB

評価に用いるDBは, 図1のような構造を持つ. 本稿では, OODBの種々の概念のうち, クラス階層, 属性の継承, メッセージパッシングの3つを考慮する. また, オブジェクトは各クラスに10,000オブジェクトあり, 単一の属性について5つの値が等確率であらわれる. ただし, 本稿ではクラスCの属性cに対する検索についての評価のみを行う. これは文献 [Takayama97-2] から, 検索を行うクラスによる応答時間等の特性はクラス階層を伝播すると判断したためである.

5.2 パラメータ

各計測に用いるパラメータ, およびそのデフォルト値を以下に示す. 各パラメータについて, 変数名に続き, 括弧内にデフォルト値, そしてパラメータの説明を行う.

c(3種類): 投機を行う属性値の場合の数.

t(10秒): トランザクション発生周期.
マルチトランザクション環境下でのトランザクションの発生時間間隔.

w(5秒): 考慮時間.
ユーザが与えられた選択肢の中から検索属性値を入力するまでの時間.

m(10,000オブジェクト/クラス): DBサイズ.
単一のクラスの持つオブジェクト数.

ℓ(100キロバイト/ヒット): 出力負荷.
検索条件に一回適合するにつき, 結果を出力するのに必要となる負荷のことである. マルチメディアデータベースでは, 記憶媒体から読み出してきたデータを展開することなどに相当する. 出力負荷のシミュレートをするために, テキストの出力負荷ℓキロバイト/ヒットを用いる.

n(10個): トランザクション数.
マルチトランザクション環境下でのトランザクションの総数.

s(50%): 投機効率.
投機的実行を行った検索条件が実際に入力される確率.

p(16台): 実プロセッサ台数.
並列環境の要素プロセッサとして使用する要素プロセッサの数.

v(16台): 仮想プロセッサ台数.
静的起動方式において, あらかじめ立ちあげておくQMT/QETの数. p<vの場合, 一台の実プロセッサ上で複数のQMT/QETが動いていることを表す.

5.3 実装環境

現存するDB管理システムを用いる代わりに, 本稿では評価用のシミュレーション・プログラムを実装して評価を行う. 評価環境は, SPARCstation 10(Solaris 2.5.1)がイーサスイッチにより30台つながっているワークステーション・クラスタを使用した. また, DBは同イーサスイッチ上にあるNFSサーバ(Ultra-Enterprise 3000)から提供する. メッセージ・パッシング・ライブラリとしては, PVM3(Version 3.3)を用いた.

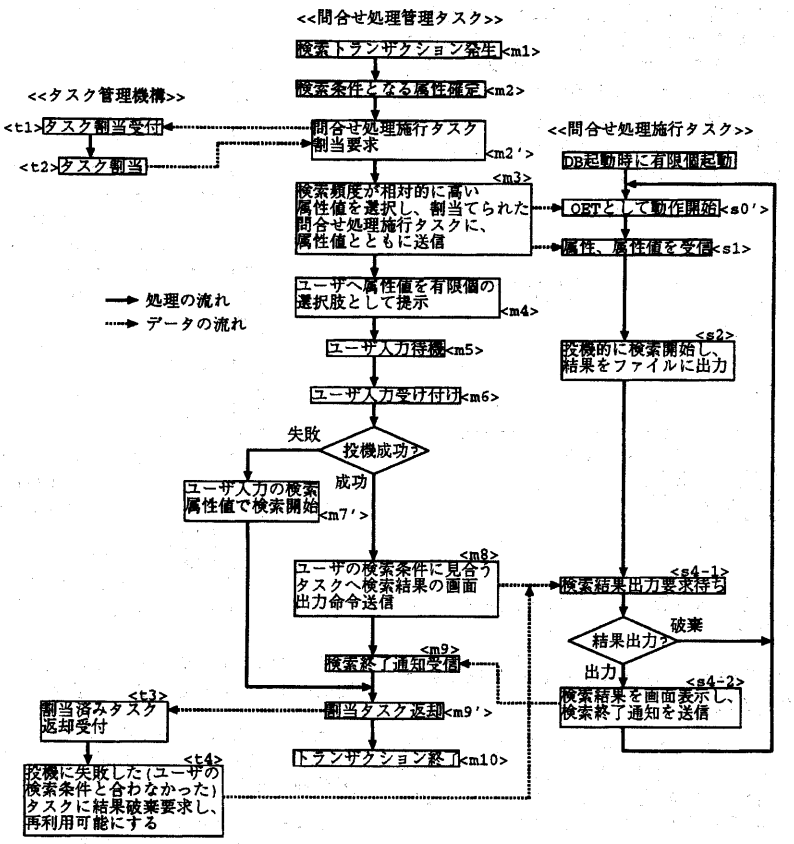


図 3: 静的起動方式のアルゴリズム

5.4 評価方法と結果

5.4.1 ユーザの考慮時間 w に対する応答時間 r の短縮

ユーザの考慮時間 w を変化させたときの応答時間 r を計測する.

図 4に計測結果を示す. 同図より, ユーザの考慮時間 w に関わらず, 静的起動方式の方が平均で 4 割程度高速なことをがわかる. これは, 動的起動方式がトランザクションが発生するたびに投機成功の場合で (1+c) 個, 投機失敗の場合には (2+c) 個のプロセスを生成しているのに対し, 静的起動方式ではあらかじめ立ちあげてあるプロセス上で即時に処理を開始でき, また投機失敗時でも QMT 自身が問合せ処理を開始するため, 通信 /OS のオーバヘッドが少ないためであると考えられる.

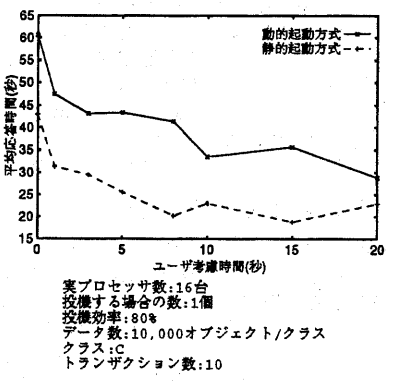


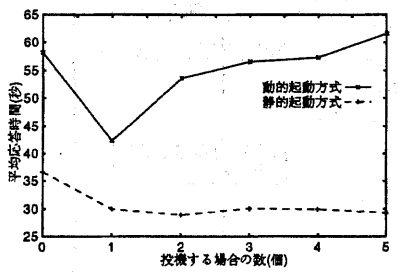
図 4: ユーザの考慮時間 w に対する平均応答時間 r の変化

5.4.2 投機を行なう属性値の場合の数 c に対する平均応答時間 r の変化

投機を行う属性値の場合の数 c を変化させたときの応答時間 r の変化を計測する.

図 5に計測結果を示す. これは, 投機を行う場合の数が増えたときの応答時間の変化を示している. 同図より, 動的起動方式では, 投機を行う場合の数 c が 2 以上になった場合, すなわち $c \geq 2$ においては, 投機による高速化があまり望めないが, 静的起動方式では投機を行なう属性値の場合の数 c が増加して

も，応答時間が極度に悪くなることはないことがわかる．これは，静的起動方式の方が，より多種の投機ができることを意味している．この現象は，先述の通り動的起動方式では投機成功時で(1+c)，投機失敗時には(2+c)個のプロセス生成を伴うため，プロセス起動によるオーバヘッドが無視できないくらいに大きいためと考えられる．



実プロセッサ数:16台
ユーザ考慮時間:5秒
投機効率:80%
データ数:10,000オブジェクト/クラス
クラス:C
トランザクション数:10

図 5: 投機を行なう属性値の場合の数 c に対する平均応答時間 r の変化

5.4.3 静的起動方式における DB の混雑時に対する性能の低減

トランザクションの発生周期 t を変化させた時の“応答時間 r”および，“応答時間 r と DB へ入るための待ち時間の合計”の変化を仮想プロセッサ台数 v が 16,32,48 台のそれぞれについて計測する．

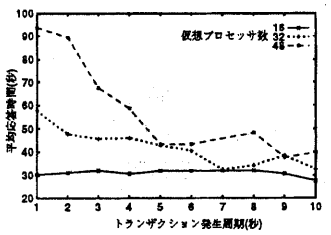
図 6に計測結果を示す．図 6(a) より，仮想プロセッサ台数 v が 16 の場合では，応答時間は DB の混み具合に関わらず一定であるが，同図 (b) でみると，“平均応答時間＋待ち時間”，すなわちユーザの待ち時間の合計は DB が混雑するに従って増加している．これは，静的起動方式では，あらかじめ QMT/QET を立ちあげておくため，同時に処理すべきトランザクション数が，立ちあげておいた QMT/QET の数を越えた場合，DB に入ることのできないユーザが出てしまうためと推定できる．

5.5 静的起動方式の問題点

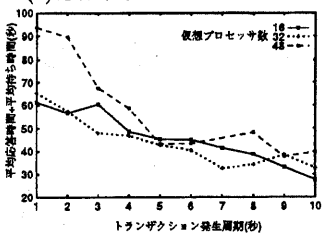
静的起動方式では，QMT/QET を最初に静的に立ちあげるため，同時に処理できるトランザクション数が有限個に固定されてしまう．そのため，同時に発生するトランザクション数を的確に予測し，必要な数だけ QET を立ちあげておく必要がある．もし，トランザクションが予測以上に発生し，QET が不足した場合は，DB に入ることができないユーザが出てしまう．

6 結論と今後の展望

著者らの提案している投機的問合せ処理を，マルチトランザクション環境下で高速に実行するための



(a) 応答時間のみに注目したもの



(b) データベースへ入るための待ち時間を考慮したもの

実プロセッサ数:16台
投機する場合の数:3個
投機効率:50%
データ数:10,000オブジェクト/クラス
クラス:C
トランザクション数:10

図 6: 静的起動方式におけるにトランザクション発生周期対する平均応答時間 r の変化

手法を提案した．本稿で提案している手法を用いることで，従来の方式と比較して，応答時間を 4 割程度短縮することができる．

今後は i) あらかじめ起動しておいたタスク数を上回るトランザクションが発生した場合に備えて，プロセスの動的起動と静的起動の効果的な融合方式の検討，ii) より細かい制御のできるメッセージ通信ライブラリとして MPI [McBryan94] 等を用いる，iii) 並列計算機上での評価などを行っていく予定である．

参考文献

[Yamana95] 山名 早人ほか：投機的実行研究の最新動向とタスク間投機的実行の有効性, 第 51 回情処全大講演論文集 (6),pp.75-76,1995.

[Takayama97-1] 高山 毅，弘中 哲夫，藤野 清次: 投機的問合せ処理：データベースのためのもう一つの並列処理, 第 8 回データ工学ワークショップ,pp.263-268,1997.

[Takayama97-2] 高山 毅，弘中 哲夫，藤野 清次: 並列環境における投機的実行の導入によるオブジェクト指向データベースシステムの検索応答の高速化，情報処理学会論文誌並列処理特集号投稿中,1997.

[Kim90] Kim,W.:Introduction to Object-Oriented Databases, MIT Press,1990.

[McBryan94] McBryan,O.A.:An overview of message passing environments, *Parallel Computing* Vol.20 No.9,pp.417-444,1994.