

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

高山 毅[†] 弘中 哲夫[†] 藤野 清次[†]

本論文では、並列環境において実行条件の評価結果を待たずに演算を開始する投機的実行をオブジェクト指向データベース（以降オブジェクト指向を OO，データベースを DB と略す）システムの間合せ処理に導入して、検索応答を高速化することを提案する。近年、OODB における間合せ処理を並列化するさまざまな研究が行われている。しかしながら、OODB ではオブジェクト識別子によるオブジェクトへのアクセス方法が原因で、負荷分散がリレーショナル DB の場合よりも困難である。一般に、DB ユーザにとって重要なのは、DB システム側の間合せ処理時間ではなく、DB ユーザ側から見た応答時間である。投機的実行は、並列環境においてジョブの応答時間を短縮するために有効な技術として知られている。本提案では、検索画面を表示してから DB ユーザによって検索条件が選択されるまでの、従来 DB システムがその DB ユーザのために処理を行っていない時間に間合せ処理の投機的実行を行う。本提案の評価用に用意した特殊な環境下での評価実験の結果によれば、本提案は実現可能であり、また検索応答の高速化において有効である。本提案を OODB の検索操作における既存の並列処理方法に付加すると、従来得られていた検索応答の高速化の度合いがさらに拡大する可能性がある。

Reduction of Response Time for Retrieval Operation in Object-oriented Database Systems by Introducing Speculative Execution Technique in Parallel Environments

TSUYOSHI TAKAYAMA,[†] TETSUO HIRONAKA[†] and SEIJI FUJINO[†]

This paper proposes to reduce a response time for a retrieval operation in object-oriented database systems by introducing speculative execution technique, which starts to execute a set of program codes before evaluating its execution condition, in parallel environments. Hereafter, we abbreviate "object-oriented" to "OO" and "database" to "DB". Recently, many researchers have tried parallel query processing in OODB. However, load balancing is known to be more difficult than that in relational DB because of an access method to an object using its object identifier. In general, it is a response time that is important for a DB user, and not a query processing time for a DB system. Speculative execution technique is known as one of the effective techniques in order to reduce a response time of a job in parallel environments. In our approach, a DB system executes speculative query processing while conventional ones do not work for a DB user from the time which it shows him/her a retrieval display to the time which it receives a single retrieval condition from him/her. Results from our experiments in a specified environment for evaluating our approach shows that our approach is feasible and effective in order to reduce a response time. If our approach is added to conventional parallel query processing approaches, it has possibility to obtain more gain.

1. はじめに

近年、オブジェクト指向データベース⁶⁾（以降オブジェクト指向を OO，データベースを DB と略す）システムにおける検索操作に対する応答速度を高速化するために、間合せ処理を並列化するさまざまな研究が

行われている^{4),8)}。いうまでもなく、DBでのデータ操作には検索操作のほかに更新操作も存在するが、ここでは議論の対象を検索操作に絞る。間合せ処理の並列化に際しては、データのフラグメント化をいかに行うかが問題となる。しかしながら、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、得られたフラグメント内のすべてのオブジェクト

[†] 広島市立大学情報科学部情報工学科

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

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 におけるそれよりも困難である。

一般に、「DB システムが問合せ処理によって検索条件に適合するデータを選択するのに要する時間」と「DB ユーザ（以降ユーザと略す）が検索操作により検索条件を指定してから応答を得るまでの時間、すなわち応答時間」には線形の相関がある。この両者を厳密に区別した場合、ユーザにとって重要なのはあくまで後者である。

並列環境において、いくつかの演算の実行をその実行条件の評価結果を待たずに開始することを、投機的実行 (speculative execution) という¹³⁾。近年、並列環境の利用技術の 1 つとして、投機的実行に関するさまざまな研究が進められている^{12),13)}。

ここで本論文での言葉の定義を 1 つ行う。すなわち、本論文で「考慮時間」といった場合、以下の 3 つの時間の総和を指すものとする。

- (1) 有限個の選択肢に限定され表示された検索画面をユーザが読む時間。
- (2) 読んだ情報をもとに、検索条件をユーザが検討する時間。
- (3) 検索条件に見合う選択肢をユーザがタイプ入力するのにかかる時間。

著者らはこれまで、並列環境における OODB システムの検索操作の処理への投機的実行の導入に向けて検討を進めてきている^{9)~11)}。本論文では、これまでの検討結果を基に、並列環境における OODB システムの問合せ処理に投機的実行を導入して、検索応答を高速化することを提案する。具体的には、検索条件が確定する前の、上で定義した考慮時間に問合せ処理の投機的実行を行い、これによってユーザから見た応答時間、すなわちターンアラウンドを短縮することを試みる。投機的実行を問合せ処理に適用する試みは、これまではなされておらず、本論文の特長といえる。

本提案を評価するために用意した特殊な環境の下での評価実験の結果は、以下のようである。

シングルユーザによるシングルトランザクションの場合：もし投機した検索条件が本当に選択されるならば、ある考慮時間までの範囲内においては、投機を行わない場合に比べてほぼ考慮時間分に相当する応答時間の短縮を達成でき、また、その境界と

なる考慮時間以上の考慮時間の範囲では、境界となる考慮時間で得られる量に等しい一定量の応答時間の短縮が達成できる。また投機を失敗した場合の損失も、無視できる程度に抑制できる。

マルチユーザによるマルチトランザクションの場合：もし投機を行ったものが実際に採用される確率である投機効率がある割合以上で、また、ある台数以上のプロセッサが利用可能であれば、投機効率とプロセッサ台数に見合って応答時間が短縮される。投機効率とプロセッサ台数に関するこれら 2 つのしきい値は、必ずしも非現実的な数字ではない。

以上より、投機的実行を導入する本提案は実現可能であり、かつ有効である。

ここで注意すべき点として、本研究は OODB の検索操作における既存の並列処理方法を否定し、代替させるものではなく、それらに本提案を付加することを目指している。したがって、従来法で得られていた検索応答の高速化の度合をさらに拡大する可能性を秘めている。

上で、投機的実行の問合せ処理への導入が本論文の特長であると述べたが、本提案自身の特長で従来法にはない特筆すべき点は、以下の 4 点である。

- (1) 考慮時間が応答時間の短縮に寄与すること。最も理想的なケースでは、通常はリアルタイムでの応答が困難な問合せがリアルタイムで応答されること。
- (2) 考慮時間に応じて応答時間が短縮され、ゆっくり考えることがトータルの作業時間の増加に直結しないということを知っているユーザは、安心して検索条件を吟味できること。
- (3) (2)の結果、
 - 意図していたことと論理的に異なる検索条件を指定してしまう、
 - 単純に検索条件をタイプミスする、といった誤った検索を行う確率を抑制することができること。
- (4) 本提案を方法論として見た場合、DB システムのための並列環境の新たな利用方法を提案していること。

また、キャッシュを用いる方法と共通する本提案の特長として、以下のこともいえる。

- 頻繁にアクセスされるデータほど応答速度が高速化されるので、データ間のアクセス頻度の優劣に関する時間変化に適合した応答が実現する。

以降、本論文は以下のように構成されている。まず

☆ 平成 8 年 3 月 6 日、図書館情報大学増永良文教授、および平成 9 年 3 月 19 日、東京大学喜連川優助教授（現教授）の助言による。

次章では議論の準備として、本提案を説明するために必要となるいくつかの先行研究について概括する。3章で、OODBの問合せ処理へ投機的実行を導入する必然性について述べた後、4章でその導入のための具体的なアルゴリズムを提案する。5章では、本提案の評価用に実装した特殊な環境下で評価実験を行い、本提案の実現可能性および有効性について評価する。最後に6章で結論と今後の展望を述べる。

2. 先行研究

2.1 OODBの問合せ処理の並列化

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。そして、それらのDB上での検索操作を高速に処理するために、問合せ処理の並列化が不可欠なものとなってきている。逆にいえば、問合せ処理の並列化を要求するような複雑なアプリケーションは、データインテンシブであることが少なくない。したがって、リレーショナルDBよりは、OODBで対応すべきである。

1990年にK.C. Kimは、文献5)でOODBの問合せ処理では以下の3種類の並列性が抽出できることを指摘している。

- パス並列性 (path parallelism): 問合せグラフ (query graph) における異なる航行パスの処理における並列性。
- ノード並列性 (node parallelism): 単一の述語 (predicate) に関係する複数のオブジェクトクラスのノード群の処理における並列性。クラス合成階層 (class-composition hierarchy) のクラス間の並列性である。
- クラス階層並列性 (class hierarchy parallelism): クラス階層 (class hierarchy) を構成するクラス間の並列性。

しかしながら、これらはいずれも並列環境における処理ノードごとの負荷に差異が生じ、効果的に負荷分散が行えるとはいえない。

続いて1991年にGannonらは文献2)で、K.C. Kimが指摘した並列性のほかに、データ並列性に類するオブジェクト並列性を提案している。しかしながらこれに関しても、前章で述べたオブジェクト識別子を用いるというオブジェクトへのアクセス方法が、並列化の際の負荷分散を困難なものにしている。

検索結果がそれ以降のさらなる検索に再利用可能であるという性質を閉包性 (closure property) という。OODBの問合せの並列処理に関する研究で、ごく最

近、中心的となってきた話題は、この閉包性の充足に関して、オブジェクトの配置方法に関してである。前者に関しては、1995年に文献8)でThakoreらが、閉包性を具備したものとしては初めての、問合せの並列処理モデルを提案している。しかしながら、ここでは各クラスに所属するオブジェクトがオブジェクト識別子の出現順にソートされているという、OODBとしては不適切な仮定がなされており、負荷分散についての議論が十分とはいえない。後者に関しては、1994年に文献4)でGhandeharizadehらが、いくつかの配置方法に関しての評価実験を行っている。その中で、「負荷分散しつつ並列度を高める」、「通信のオーバーヘッドを抑制する」というトレードオフの関係にある2つの目的をとともに達成するための方法として、同一のデータのコピーを多数の処理ノードに置く方法を紹介している。しかしながら、この方法はDBの一貫性の維持を複雑化させるもので、実用化に向けてはさらなる議論が必要である。

このように、OODBの問合せの並列処理においては負荷分散がつねに問題となり、またその解決が望まれている。問合せをDBシステムが並列処理する場合、ユーザ側から見た応答時間は、最も実行時間の長いプロセッサの処理時間に依存する。この最も負荷の重いプロセッサが自身に割り当てられた処理を終えるとき、他の大部分のプロセッサはすでに自身に割り当てられた処理を終え、この問合せ処理以外に処理を割り当てられていなければ、アイドル状態にある。本論文ではこのような状態にあるプロセッサを余剰プロセッサ (unused processor) と呼ぶことにする。負荷分散が均等に近い形でなされないと、余剰プロセッサが多数生じ、資源量に見合った応答時間の短縮が実現しない。

2.2 投機的実行

投機的実行は近年、並列環境の利用技術の1つとして関心が高まっている。投機的実行について、文献12)、13)をもとに概括する。

プログラミング言語レベルで見れば、手続き型、関数型、論理型のすべての言語において、投機的実行の適用が試みられている。

- 手続き型言語: 条件分岐を含んだ逐次プログラムにおいて、各種実行条件に対応する命令セットを実行条件の評価値確定前に実行する。
- 関数型言語: 引数データ群の確定前に引数データ群の組合せを仮定して実行する。
- 論理型言語: 論理式群の真偽確定前に真偽の組合せを仮定して実行する。

いずれの場合にも、実行条件の評価値が確定すると、投機の実行したそれぞれの結果のうち、確定した評価値のときに得られるべき結果のみが活用される。

ここにおいて著者らが主張することは、これらの各種プログラミング言語ごとの投機の実行の試みは、そのまま対応する各種 DB 言語に拡張可能であるということである。この方針の基に、4 章で本論文での中心となる投機の実行導入の提案を行う。

再び従来の投機の実行に議論を戻す。

投機の実行は通常の実行に加えて行うもので、タスクの数を増加させるため、逐次実行ではなく並列実行する場合に有効な技術として知られている。

投機の実行を行う適性を持つアプリケーションとは、以下のような性質を持つものである。

- 実行条件の評価値確定後の計算量が十分大きく、短時間では計算結果が得られないもの。
- メモリの範囲内であらかじめすべてのケースを実行できず、通常は実行条件の評価値確定後に実行を開始するもの。なお、すべてのケースを実行することは慣習的に投機の実行とは呼ばない。
- 投機効率がある程度判断でき、かつ過度に低いもの。

投機の実行をするレベルの動向としては、当初の命令レベルに加えて、最近ではタスクレベルで行うことに対する関心が高まってきている。

前節で述べたとおり、余剰プロセッサは、負荷分散が困難なほど、生じやすい。ここで投機の実行は、余剰プロセッサを用いて行う場合に、ジョブの応答時間を効果的に短縮できることが知られている。

一般に、投機の実行では投機が失敗した場合のペナルティに注意を払う必要がある。

2.3 並列環境と PVM

一般に、並列環境は、

(1) 共有メモリ型並列処理環境

(2) ネットワーク型並列処理環境

の 2 つに分類される。(1) は、多数のプロセッサ間に共有メモリを持つ並列計算機上での並列環境である。それ以外の並列環境である (2) は、さらに 3 つに分類される。

(2-1) 共有メモリを持たない並列計算機上の並列環境。

(2-2) 複数のマシンをネットワークで結合することによってマルチプロセッサを実現している並列環境。

(2-3) ソフトウェアまたはハードウェアによって、疑似的に共有メモリを実現している並列環境。

PVM (Parallel Virtual Machine)³⁾は、(2) の並

列環境におけるポータブルなプラットフォームとして、MPI, PARMACS, Zipcode などとともに代表的である⁷⁾。

PVM はマスター・スレーブ方式のプログラミングが可能であり、親タスクがその子タスクを生成する関数 `pvm_spawn()` や、タスク間でのメッセージの送受信を行う関数 `pvm_send()`, `pvm_recv()` などを持つ。タスク生成では、すべてのプロセッサの負荷状態を観測したうえで、最も負荷が軽いプロセッサを PVM 自身が選択することができるので、PVM は負荷分散においてすぐれている。

3. OODB の問合せ処理への投機の実行導入の必然性

2.1 節で、OODB の問合せの並列処理においては負荷分散が困難であり、余剰プロセッサが生じやすいことを述べた。2.2 節で述べたとおり、投機の実行は余剰プロセッサの利用技術として有効であることが分かっている。DB システムに投機の実行を導入する試みは、並行制御において Bestavros ら¹⁾によってなされているが、問合せ処理においては 1 章で述べたとおり、今までのところなされていない。

Bestavros らの方法は、トランザクションのアボート (abort) に起因する時間的損失を抑制するために、複数の異なるトランザクションが同一のデータへ同時にアクセスしていることが発見された時点で、アボートされる危険性があるトランザクションについて、シャドウ (shadow) と呼ばれる、保険に相当するトランザクションを投機的に発生させるものである。Bestavros らの方法では、DB の一貫性を保ちつつ並行制御を行うために、シャドウの実行を一時ブロックしたり、競合しているトランザクションのコミットを遅延させるなどのオーバヘッドをとまう。また、その性能評価においては、資源量を無限大としていることに加え、投機をするためのコスト評価を行っていないため、実環境での定量的な評価として、また投機の実行の導入によって資源量に見合った有効性が得られているのかに関する議論として、十分とはいえない。

本論文では更新操作を考察の対象外とし、投機の実行の導入を並行制御ではなく問合せ処理において行うことによって、Bestavros らの方法に見られるオーバヘッドを回避し、利用可能な資源を効果的に利用することを目指す。

1 章で定義した考慮時間には、DB システムがそのユーザのための処理を行う余地が残されている。DB 内の各データの検索頻度は、問合せ処理の投機の実行

を試みる場合の投機効率の有力な判断材料となりうる。一般に、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。以降、この性質のことを『検索の局所性』と呼ぶことにする。

さらに、いかなる検索操作も、更新操作とは異なりDBの内容を変更するわけではない。したがって、たとえ誤った問合せ処理を行ったとしても、ロールバック操作によって、検索操作が行われる以前の状態に復帰する必要はない。すなわち、2.2節で述べた投機失敗の場合のペナルティとしては、投機をしない場合に対する時間的損失のみを考慮すればよい。

この時間的損失であるが、本論文で議論しているマルチプロセッサ環境では、投機の実行のためにいくつかのプロセッサを使っていて、なおかつそのすべてが投機が失敗した場合でも、その時点で使われていないプロセッサがほかにあれば、ただちにそのプロセッサを用いて、確定した検索条件に見合う問合せ処理を開始することができる。すなわち、議論している時間的損失は、はじめに投機の実行を開始するために要する時間のみを見積もればよく、これは通常、検索操作全体の処理時間と比して無視できる程度と考えられる。

以上のように、OODBの問合せ処理へ投機の実行を導入することには必然性がある。

4. 投機の実行を導入した問合せ処理のアルゴリズム

アルゴリズムの提案に先立ち、本論文では以下の2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属性値のそれぞれに対応する選択肢から選択することによって行う。
- (2) 1章で述べたとおり、本論文では検索操作のみを議論の対象としており、ユーザが検索操作を行った場合の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。

以上の仮定のもとで、図1が本論文で提案する『投機の実行を導入した問合せ処理のアルゴリズム』である。アルゴリズムは、以下の6つの論理的なステップに分けて考えることができる。

step1: 問合せ処理管理タスクは、検索条件となる属性、すなわちキー属性(key attribute)の確定時点で、問合せ処理実行タスクを起動したうえ、DBのログ情報を基に検索頻度が相対的に高い属性値を送信する(<m1>-<m3>)。複数の属性値で検

索の投機の実行を行う場合には、それぞれの属性値に対応する問合せ処理実行タスクを起動する。

step2: 問合せ処理実行タスクは検索を投機的に開始し、結果をファイル出力する(<s1>-<s2>)。出力されたファイルは、(トランザクションID, 属性値)の組によって、一意に識別される。

step3: 問合せ処理管理タスク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属性値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属性値に対応する問合せ処理実行タスクを新たに起動し、ここから検索を開始する(<m7>, <s3>)。

step5: 問合せ処理管理タスクは、ユーザ入力属性値に見合う検索をしている問合せ処理実行タスクへメッセージを送る(<m8>)。

step6: これを受信した問合せ処理実行タスク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>)。

ここで、上記のstep1におけるログ情報の利用に関して以下の補足をしておく。ここでのDBログは、更新操作のみではなく、検索操作についても記録されているものを想定している。利用状況に即したDB運用を行う観点からしても、検索操作も記録されているログは有用である。しかしながら、一般にDBログといった場合、その容量の巨大化回避などの理由で更新操作のみを記録し、検索操作は記録しないことも多い。そのような場合における、投機のための判断材料を用意する方法としては、以下の3つが考えられる。

- (1) DBログの記録方針を変更し、検索操作の記録も始める方法。
- (2) (1)において、更新操作のように操作者、操作時間などの、全データは記述せず、より単純に、キー属性とその属性値のみを各値ごとにカウントする、より簡易なログを作成する方法。
- (3) DB管理者、ユーザなどが持つヒューリスティクスを用いる方法。

図1のアルゴリズムの特徴は、マスター・スレーブ方式のプログラミング手法を用いていることと、2.3節で述べたすべてのタイプの並列環境で実現可能なことである。ただし、(2-2)のタイプの場合には、本論文では簡単のため、DBは要素プロセッサとは別個に存在するNFSサーバ上に集中的に存在させ、要素プロセッサとなる各マシンがそれをマウントするものとする。

アルゴリズムの適用例としては、映像データなどの

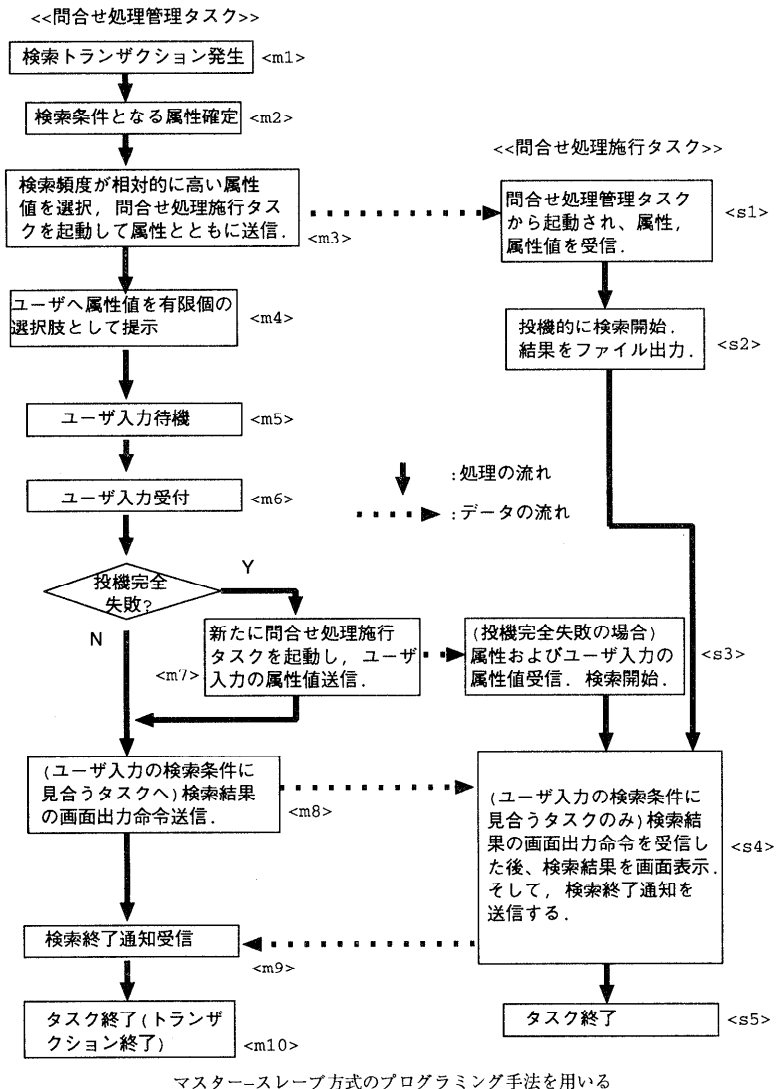


図1 投機的実行を導入した問合せ処理のアルゴリズム
Fig.1 Algorithm of query processing with speculative execution.

大容量オブジェクト (large object) を比較的高い圧縮比で格納している場合に、検索される可能性が相対的に高いオブジェクトの圧縮データを投機的に伸長しておく場合などが考えられる。本アルゴリズムの適用により、伸長に要するコストを完全または部分的に隠蔽する効果があり、有効である。

5. 評価実験

5.1 評価に用いるあるデータベース例の概要

図2は、あるOODBの例のスキーマを表現している。なお、記法は文献6)に基づいている。ここでは、OODBの種々の概念のうち、クラス階層、属性の継

承 (inheritance of attribute)、メッセージ・パッシング (message passing) の3つを考慮することにする。まず、“A”、“B”、“C”、“D”の4つのクラスを定義する。各クラスにはそれぞれ局所的な属性“a”、“b”、“c”、“d”を持たせる。これらのクラスをクラス/サブクラスリンク (class/subclass link) によって結び、クラス階層を構成する。あるクラスの属性は、それより下位のすべてのクラスが継承し、たとえば、属性“a”はクラス“B”、“C”、“D”へ継承される。オブジェクトからの属性値の取り出しは、そのオブジェクトへのメッセージ・パッシングにより実現する。

簡単のために、すべてのクラスに同数のオブジェク

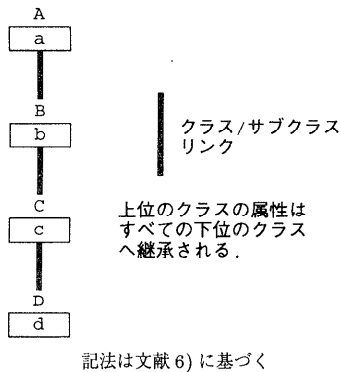


図 2 ある OODB の例のスキーマ

Fig. 2 A schema representation of an example of OODB.

トを置き、単一の属性について 5 つの値が等確率で現れるようにする。換言すると、各属性値はそのクラス内で 20% の確率で現れる。

本評価実験では、すべての実験を通じて、キー属性を“a”とする場合と、キー属性を“c”とする場合について評価を行う。ここで注意すべきこととして、キー属性を 2 つ採用するのは、継承の段数による実験結果の差異を観察するためであり、キー属性“a”での検索とキー属性“c”でのそれを並列に行うという意味ではない。

一般に、DB システムでは複数の検索条件を AND, OR などによって結合することによる絞り込み検索が行われることが少なくない。しかしながら、投機的な問合せ処理が有効になるか否かを決定するのは、そのような検索条件の複雑さの程度ではなく、どれだけ投機が成功するかという投機効率にある。たとえ複雑な検索条件であっても、その検索条件での検索が頻繁に行われる場合に、その条件で投機を行ったとして、投機効率がその検索頻度に見合ったものになることには変わりはない。いうまでもなく、評価実験を行う場合にあらゆる複雑度の検索条件を用いて評価を行うことは現実には不可能である。そこで本論文では、上記のようにキー属性 1 個の場合を検索条件として、投機的な問合せ処理の基本特性を評価することに主眼を置く。

5.2 定数パラメータ

以下の 2 つのパラメータは、全実験を通して定数とする。

- 投機を行う属性値の場合の数: 1 [ケース].
すなわち本実験では最も検索頻度が高いことが既知である属性値 1 つのみについて、検索の投機的実行を行う。
- トランザクション発生周期: 10 [秒]
たとえマルチトランザクションであっても、トラ

ンザクション間で時間的な重複がない場合には、シングルトランザクションの場合と同等の性能が得られることは自明である。そこでマルチトランザクションでの評価を行う場合に、時間的な重複が生じるように、定数値を定めている。

5.3 制御パラメータ

各種制御パラメータ、およびそのデフォルト値は以下のものである。

- w [秒]: 考慮時間. デフォルト値: 5. 定義は 1 章で述べたとおりである. 図 1 中では <m4> 実行後 ~ <m6> 実行前までの経過時間にあたる。
- m [オブジェクト/クラス]: DB サイズ. デフォルト値: 100. 単一のクラスが持つオブジェクト数. 一般に、DB では DB サイズが万のオーダーに達することも少なくないが、ここでは資源量の問題からデフォルト値を比較的低いレベルに抑制している。
- l [キロバイト/ヒット]: 出力負荷. デフォルト値: 50. 4 章で述べたとおり、本研究では、検索結果の出力がリアルタイムでは困難な、検索結果を出力するまでの負荷が比較的重い応用を適用範囲として考えている。ここでは種々の検索結果の出力負荷を表現するために、テキストの出力負荷 l [キロバイト/ヒット] を採用する。この意味は、検索条件に適合するデータ 1 つの出力負荷が、 l [キロバイト] の文字を出力する負荷と等しいということである。
- n [トランザクション]: マルチトランザクションでの評価を行う場合に、マルチユーザから生じるトランザクションの総数. デフォルト値: 5.
- s [%]: 投機効率. デフォルト値: 80.
- p [プロセッサ]: プロセッサ台数. デフォルト値: 5. 並列環境の要素プロセッサとして使用するワークステーション (以降 WS と略す) の数。

5.4 評価パラメータ

以下の 2 つの評価値を評価パラメータとして採用し、原則として投機に成功、失敗、および投機をしないの 3 つの場合について計測する。

- t [秒]: トランザクション完了時間. 図 1 中では <m2> 実行後 ~ <m9> 実行後までの経過時間。
- r [秒]: 応答時間. 図 1 中では <m6> 実行後 ~ <m9> 実行後までの経過時間。

5.5 実装環境

現存する DB 管理システムを用いる代わりに、本論文では本提案を評価するために以下のような特殊な環境を用意する。

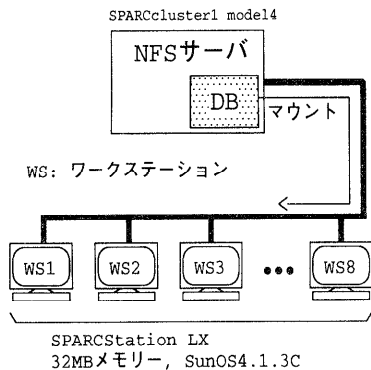


図3 評価実験で用いる並列環境

Fig. 3 A parallel environment in the benchmark.

まず, SunOS 4.1.3C 搭載の SPARCstation LX (主記憶 32 MB) がイーサネットによって 8 台結合されている WS クラスタを用いる (図 3)。これは 2.3 節で述べた (2-2) のタイプの並列環境にあたる。SPARC-cluster 1 model 4 が NFS サーバとして存在し, WS クラスタと接続されている。WS クラスタ内の全 WS は, NFS サーバ上の一部のファイルシステムをオートマウントする。マスタ・スレーブ方式を実現するためのメッセージ・パッシング・ライブラリとして PVM3.3.11 を, プログラミング言語として C++ を, コンパイラとして SPARCCompiler C++ 2.0.1 を用いる。

5.6 実験方法

本実験では, トランザクションが順次生成される様子をシミュレートするために, 前章で述べた 2 つのタスクのほかに, 一定時間ごとに問合せ処理管理タスクを起動するトランザクション生成タスクを加える。トランザクション生成タスクは, トランザクション 1 つに対して問合せ処理管理タスクを 1 つ起動する。問合せ処理施行タスクが行うファイル出力は, 各 WS が持つ局所的なファイルシステムへ行う。以上の条件のもとに, 以下の 6 項目の実験を行う。実験 1~3 まではシングルユーザ環境でのシングルトランザクションにおける評価であり, 実験 4~6 は, マルチユーザ環境でトランザクションが一定周期で発生するマルチトランザクションでの評価である。

実験 1: 考慮時間 w を変化させたときの応答時間 r , およびトランザクション完了時間 t を計測する。

実験 2: 出力負荷 l を変化させたときの応答時間 r を計測する。

実験 3: DB サイズ m を変化させたときの応答時間 r を計測する。

実験 4: 各トランザクション総数 n でのマルチトランザクションに含まれる全トランザクションの応答

時間 r の平均値を計測する。なおここでは, すべてのトランザクションで投機が成功する場合, 失敗する場合, および投機をまったく行わない場合の 3 つのケースについて計測を行う。

実験 5: 各投機効率 s での, マルチトランザクションに含まれる全トランザクションの応答時間 r の平均値を計測する。ここでは, 投機をする場合, しない場合の 2 つのケースについて計測する。

実験 6: 単一のトランザクションにおける応答速度の数値を, そのトランザクションにおける応答時間の逆数によって定義する。プロセッサ台数が 1 台で, その他の制御パラメータはすべてデフォルト値, そして投機をしない場合のマルチトランザクションに含まれる全トランザクションの平均応答速度を平均応答速度の基準値とする。この基準状態から, 投機をする場合としない場合について, プロセッサ台数を変化させたときの平均応答速度の向上に関する台数効果を計測する。

具体的には, ある観測ケースの台数効果は

$$\frac{\text{その観測ケースでの平均応答速度}}{\text{基準ケースでの平均応答速度}}$$

によって表現できるが, この値は

$$\frac{\text{基準ケースでの平均応答時間}}{\text{その観測ケースでの平均応答時間}}$$

と等価であることは応答速度の数値の定義より明らかなので, 観測された応答時間を基に, こちらによって算出する。

以上において, 経過時間の計測は C++ の関数 `ftime()` を用いて行う。また, 考慮時間, およびマルチトランザクションの発生周期は, C++ の関数 `sleep()` によって実現する。

また, 今回の評価実験では, 並列環境において投機的実行を導入する効果を評価することに主眼を置き, 各属性値に対する問合せ処理施行タスクでの問合せ処理の施行は逐次的に行う。

5.7 実験結果

実験 1: 応答時間については, キー属性 “c” の場合の結果を図 4 の左半分に, キー属性 “a” の場合の結果を図 4 の右半分に示す。また, トランザクション完了時間については, キー属性 “c” の場合の結果を図 5 の左半分に, キー属性 “a” の場合の結果を図 5 の右半分に示す。

実験 2: 図 6 が, 出力負荷に対する応答時間の関係を表す実験結果である。

実験 3: 図 7 が, DB サイズに対する応答時間の関係を表す実験結果である。

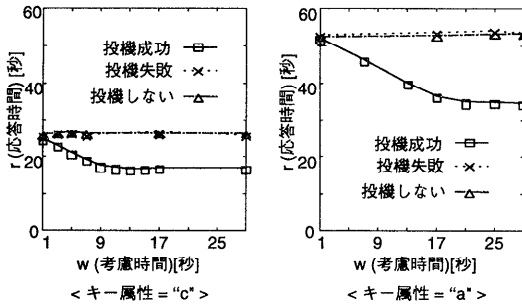


図4 実験1-1: w に対する r
Fig. 4 Experiment 1-1: r for w .

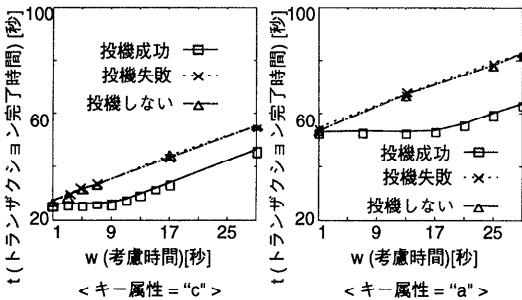


図5 実験1-2: w に対する t
Fig. 5 Experiment 1-2: t for w .

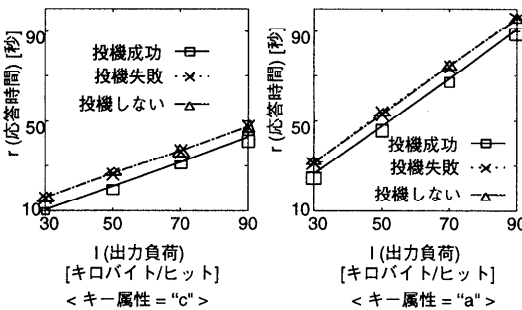


図6 実験2: l に対する r
Fig. 6 Experiment 2: r for l .

実験4: トランザクション数 n に対する各トランザクションの平均応答時間の関係を図8に示す。なお、資源量の限界から属性 c についてはトランザクション数10以上、属性 a についてはトランザクション数6以上では計測不能であった。

実験5: 各投機効率における全トランザクションの平均応答時間の関係を図9に示す。

実験6: 各プロセッサ台数における全トランザクションの平均応答速度の台数効果の関係を図10に示す。

5.8 考 察

まず、全実験を通じて、キー属性“ c ”の場合とキー属性“ a ”の場合で、各評価パラメータに関して、絶

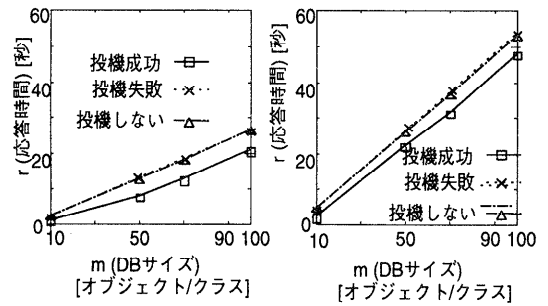


図7 実験3: m に対する r
Fig. 7 Experiment 3: r for m .

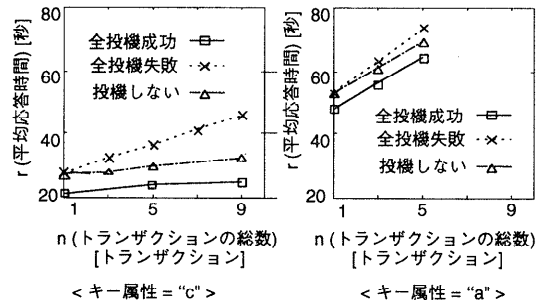


図8 実験4: n に対する r の平均
Fig. 8 Experiment 4: mean of r for n .

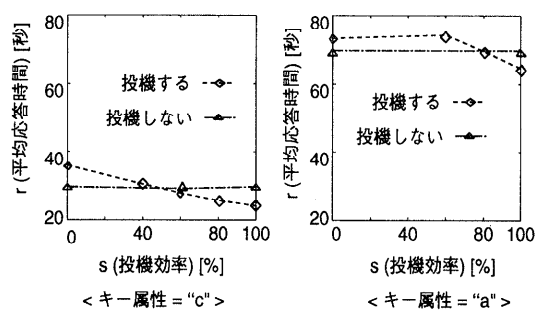


図9 実験5: s に対する r の平均
Fig. 9 Experiment 5: mean of r for s .

対的な値は違っていても、制御変数に対する変化の傾向は類似していることが分かる。すなわち、評価パラメータの値の変動に関する傾向は、クラス階層を伝播するといえる。

以下、各実験結果をより詳細に分析する。

実験1: 応答時間について: 投機をしない場合の応答時間は考慮時間によらずにはほぼ一定であるのに対して、投機に成功した場合の応答時間は、ある考慮時間まで考慮時間の増加に従って短縮されている。この短縮の程度は、その考慮時間にはほぼ一致している。これはDBシステムとして見れば、この考慮時間の範囲内では長く考えるほど応答時間が短くなり、よりリ

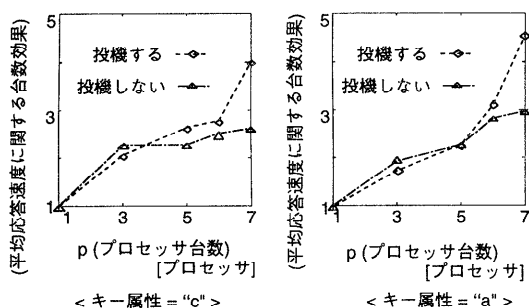


図 10 実験 6: p に対する平均応答速度の台数効果

Fig. 10 Experiment 6: Relative speed up by processors on mean of response speed for p.

アルタイムに近い応答が可能であることを意味する。

考慮時間がこの範囲を超えると、応答時間はほぼ横ばいになっている。すなわち、ある考慮時間以上では、投機をしない場合に比べての応答時間の短縮が一定量となることを示している。

時間的損失について見てみると、投機に失敗した場合の応答時間は、投機をしない場合のそれとほぼ一致しており、考慮時間によらずに一定である。3章で予想したとおり、投機を行って失敗したとしても、投機を行わない場合に比べての応答時間の増加は無視できる程度であるといえる。

トランザクション完了時間について：前節で投機に成功した場合の応答時間の短縮を確認した。しかし、たとえ応答時間が短縮されたとしても、投機を行ったためにトランザクション完了時間が伸びてしまうのであれば意味がない。

図 5 の実験結果において、投機に成功した場合のトランザクション完了時間は、投機をしない場合のそれよりもつねに短い。すなわち、応答時間とともにトランザクション完了時間も短縮されており、意味のある高速化がなされているといえる。

図 5 をより詳細に見てみると、投機をしない場合のトランザクション完了時間は考慮時間の増加にあわせて線形に増加していくのに対し、投機に成功した場合のトランザクション完了時間は、ある考慮時間まで考慮時間が増加しても一定値のまま抑制され、その後増加に転移している。DB システムとして見れば、この転移する考慮時間以下の考慮時間であればどれだけ長く考えようとも、トランザクシ

ョンが完了する時間には変化がなく、単一の検索作業に費やされる時間の総計は一定であることを意味している。考慮時間がこの範囲に納まるような応用においては、この性質は特筆する価値がある。

投機に失敗した場合のトランザクション完了時間は、投機をしない場合のそれとほぼ同様な挙動を示しており、ここにおいても投機失敗によるペナルティとしての時間的損失は無視できる程度であるといえる。

実験 2: 出力負荷によらずに、投機に成功した場合は、投機をしない場合よりも一定量の応答時間の短縮がはかれる。また、投機に失敗した場合の応答時間は、投機をしなかった場合のそれとほぼ一致し、投機失敗の損失は無視できる程度であるといえる。

実験 3: 投機に成功した場合の投機をしない場合に対する短縮の割合は、ある DB サイズまでは DB サイズの増加に依存して増加し、その後一定となる。投機に失敗した場合の応答時間は、投機をしなかった場合のそれとほぼ一致し、投機失敗の損失は無視できる程度であるといえる。

実験 4: この実験結果は、マルチユーザによるマルチトランザクションの場合の平均応答時間の短縮の最大値、および最小値を表現していると考えることができる。後者は、すべてのトランザクションにおいて投機を失敗した場合の損失、すなわち損失の最大値を表現しているともいえる。

シングルユーザによるシングルトランザクションの場合には、投機を失敗した場合の応答時間の時間的損失は投機をしなかった場合に比べて無視できる程度であったが、マルチユーザによるマルチトランザクションの場合の平均応答時間の時間的損失は、無視できる程度といい切れるわけではない。

しかしながら、3章で述べた検索の局所性を考慮すれば、マルチトランザクションのすべてのトランザクションで投機を失敗するというのは現実的とはいえない。

投機の失敗回数に比例して時間的損失が累積していく原因としては、以下のことが考えられる。すなわち、投機に失敗したタスクは検索結果を採用されないだけで、タスクとしては存続するので、プロセッサを不必要に使用し、他のタスクのプロセッサの利用機会を抑制する。PVM の関数で、不必要となったタスクを強制的に終了させる関数 `pvm_kill()` も存在するが、タスクの強制終了では

後処理の負荷が無視できず、タスクを放置する方法よりもさらに投機失敗の損失が増すことは予備実験において確認している。

この時間的損失を縮小することは、今後の課題といえる。

実験 5: この実験結果は、マルチトランザクションに対して、投機効率がどの程度あれば投機が有効であるかを示している。いずれの場合にも、投機効率が80%程度以上あれば、応答時間の短縮が確実にはかかっていることが分かる。ここでも検索の局所性を考慮すれば、投機した検索条件の採用確率80%は、必ずしも非現実的な数字とはいえない。このしきい値となる投機効率を、より低い割合に抑制することも今後の課題といえる。

実験 6: 投機を行うと投機を行わない場合よりもタスクの数が増えるので、この実験結果は投機の実行を導入するために必要な最小プロセッサ数を表現しているといえる。いずれの場合にも、プロセッサ台数が5台以上あれば、投機を行う場合の応答速度の向上に関する台数効果は、投機をしない場合のそれを上回っている。

以上実験 1~6 より、並列環境において OODB システムの問合せ処理に投機の実行を導入することは実現可能であり、かつ有効である。

6. 結論と今後の展望

並列環境における投機の実行を OODB システムの検索操作の処理へ導入することにより、応答時間は短縮され、すなわち検索応答は高速化される。

検索の予測が可能な場合、結果をあらかじめ用意しておく方法も考えられる。そのような方法と比較して、投機的検索を毎回行う意義としては、以下のようないえる。

- (1) 単一の DB システムでも、DB ユーザによる複数回の操作によって、対象となるデータを絞り込んでから、最終的に検索条件を入力させる場合は少なくない。この操作の進展を木 (tree) によって表現すれば、DB ユーザの操作の進展に応じてそのつど投機的な検索を行うことは、木上の枝狩りをしつつ、検索対象となる葉 (leaf) 上で投機を行うことに相当するので、適切な検索結果をより多く用意しやすい。
- (2) DB ログの動的な読み取りを実現することにより、検索の傾向の変化へより迅速、柔軟、かつ容易に対応が可能である。

今後の展望としては、以下に関する検討を行っている。

く予定である。(i) ある検索条件で投機した結果の二次的な利用方法、たとえば、AND, OR を用いた複数条件による絞り込み検索への利用方法や、より意味的に制約された検索条件への絞り込みへの利用方法、(ii) 2.3 節の (2-2) の並列環境で DB を複数のディスクへ分散させる場合を含めた、投機を有効にするためのオブジェクトの配置方法、(iii) マルチトランザクションの場合に本提案が有効となる最小投機効率を引き下げるための投機失敗の場合の時間的損失の縮小、(iv) クラス合成階層などに代表される OODB としての機能の追加や、制御パラメータ、評価パラメータの追加、および複数の属性値による問合せ処理の投機の実行など、評価実験の内容のさらなる拡充、(v) 属性値のみならず、キー属性の投機、(vi) 問合せ処理の既存の並列アルゴリズムとの効果的な融合方法。

謝辞 本研究を進めるに際し、構想段階では京都大学の上林弥彦教授、図書館情報大学の増永良文教授から、実験段階では広島市立大学の北上始教授から、総括段階では東京大学の喜連川優教授、電子技術総合研究所小島功氏から貴重なコメントをいただいた。また、ドラフト全般について、お茶の水女子大学の藤代一成助教授から貴重なコメントをいただいた。ここに衷心より謝意を表する。

なお、本研究の一部は広島市立大学特定研究プロジェクト「A408」による。

参考文献

- 1) Bestavros, A. and Braoudakis, S.: Value-cognizant Speculative Concurrency Control, *Proc. 21th International Conference on Very Large Data Bases*, pp.122-133 (1995).
- 2) Gannon, D. and Lee, J.K.: Object-Oriented Parallelism: pC++ Ideas and Experiments, *Proc. 1991 Joint Symposium on Parallel Processing*, pp.13-23 (1991).
- 3) Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V.: *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press (1994).
- 4) Ghandeharizadeh, S., Wilhite, D., Lin, K. and Zhao, X.: Object Placement in Parallel Object-Oriented Database Systems, *Proc. 10th International Conference on Data Engineering*, pp.253-262 (1994).
- 5) Kim, K.C.: Parallelism in Object-Oriented Query Processing, *Proc. 6th International*

☆ 平成 9 年 3 月 19 日、電子技術総合研究所小島功氏の助言による。

Conference on Data Engineering, pp.209-217 (1990).

- 6) Kim, W.: *Introduction to Object-Oriented Databases*, The MIT Press (1990).
- 7) McBryan, O.A.: An Overview of Message Passing Environments, *Parallel Computing*, Vol.20, No.4, pp.417-444 (1994).
- 8) Thakore, A.K., Su, Y.W. and Lam, H.X.: Algorithms for Asynchronous Parallel Processing of Object-Oriented Databases, *IEEE Trans. Knowledge and Data Engineering*, Vol.7, No.3, pp.487-504 (1995).
- 9) 高山 毅, 弘中哲夫, 藤野清次: オブジェクト指向データベースシステムにおける投機的並列データ操作の基本構想, 第 52 回情報処理学会全国大会論文集 (4), pp.251-252 (1996).
- 10) 高山 毅, 弘中哲夫, 藤野清次, 児島 彰: 投機的実行のオブジェクト指向データベース検索への適用方式とその評価, 第 53 回情報処理学会全国大会論文集 (1), pp.185-186 (1996).
- 11) 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, 電子情報通信学会第 8 回データ工学ワークショップ, pp.263-268 (1997).
- 12) 館村純一: 投機的実行を応用したインタラクティブ並列プログラム, 情報処理学会プログラミング研究会研究報告, 2-13, pp.97-104 (1995).
- 13) 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 回情報処理学会全国大会論文集 (6), pp.75-76 (1995).

(平成 8 年 9 月 17 日受付)

(平成 9 年 9 月 10 日採録)

高山 毅 (正会員)



昭和 41 年生. 平成 2 年筑波大学第三学群情報学類卒業. 平成 7 年筑波大学大学院博士課程工学研究科電子・情報工学専攻修了. 博士 (工学). 現在, 広島市立大学情報科学部情報工学科助手. データベースの並列処理, データベースを用いた形状設計支援, データベースのマルチメディアシステムへの応用に関する研究に従事. 情報処理学会のほか, 電子情報通信学会, 日本図学会会員.

弘中 哲夫 (正会員)



昭和 40 年生. 昭和 63 年山口大学工学部電気工学科卒業. 平成 2 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了. 平成 5 年同大学大学院博士課程修了. 同年, 九州大学工学部情報工学科助手, 平成 6 年広島市立大学情報科学部情報工学科助教授, 現在に至る. 計算機アーキテクチャ, 並列処理システムの研究に従事. IEEE, ACM 各会員.

藤野 清次 (正会員)



昭和 25 年生. 昭和 49 年 京都大学理学部卒業. 三洋電機および計算流体力学研究所に勤務. 平成 2~3 年度東京大学工学部受託研究員. 平成 5 年 5 月工学博士 (東京大学). 平成 6 年 4 月広島市立大学 情報科学部教授. 平成 6 年 8~9 月 Karlsruhe 大学客員研究員. 大規模行列の高速並列計算の研究に従事. 著書「反復法の数理」(朝倉書店刊, 共著). 現在, Gerschgorin の円盤定理および Fractal 解析に興味を持つ. 日本応用数理学会会員.