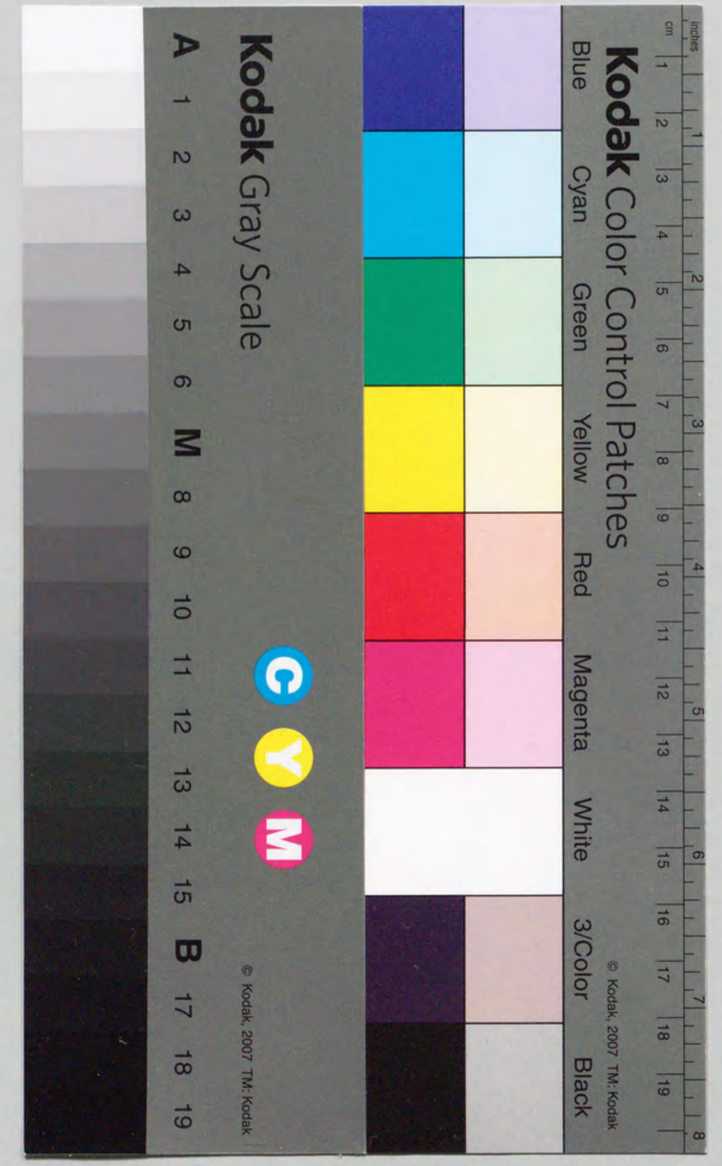


報告番号 ※乙第 4312 号

主論文の要旨

題名

大規模リレーショナルデータベースの高速処理方式に関する研究

氏名 井上 潮

主 論 文 の 要 旨

報告番号	※乙 第　　号	氏 名	井上　潮

本研究は，リレーショナル・データベース管理システムにおける大規模なデータベースの処理を飛躍的に高速化させることを目的としている．リレーショナルデータベースの歴史は１９７０年から始まるが，システムの開発当初からデータベース処理の高速化は大きな技術的課題となっていた．データベース処理は，予め決められたキーを用いてデータベース中のデータに対してアクセスする定型処理と，このようなキーを用いないでアクセスを行う非定型処理の２種類に大別できる．従来のリレーショナル・データベース管理システムでは，インデックス技術によって定型処理に関しては十分短い時間内で処理を完了できるようになっているが，非定型処理に関しては大規模なデータベースでは非常に長い処理時間がかかっていた．しかし，高度なデータベース応用システムが開発されるのに伴って，リレーショナル・データベースが本来持つ集合的な操作を含む非定型処理を高速化したいという利用者の要求が，急速に高まってきている．本研究は，このような背景をもとに，リレーショナル・データベースにおける非定型処理を高速化すること，および定型処理と並行して実行できるようにすることを目的とするものである．

本論文は６章で構成され，第１章：序論，第２章：データベース専用プロセッサ方式，第３章：専用プロセッサにおける最適化処理方式，第４章：並列処理における動的負荷配分方式，第５章：多版管理による並行処理制御方式，第６章：結論，となっている．以下，その内容と主な成果を述べる．

第１章では，研究の背景，従来の高速処理方式の概観，本研究の方針について述べた．データベースシステムの基本的な構成において，大規模なリレーショナルデータベースの高速処理を行おうとした場合，ＣＰＵネック，メモリネック，入出力ネック，及びデータネックの４種類が性能上のボトルネックとなりうる．これらのボトルネックの原因と従来技術を分析した結果，ＣＰＵネックに対しては専用プロセッサと並列処理，入出力ネックに対してはディスク入出力の並列化とインテリジェント化，データネックに対してはデータの多版管理の技術が有望であると考えた．従って本研究では，２つのアプローチで問題解決を図ることにした．第１のアプローチでは，通常のデータベース管理システムが搭載された汎用計算機システムにサーチ処理とソート処理のための専用プロセッサを付加する

主 論 文 の 要 旨

報告番号	※乙 第　　号	氏 名	井上　潮

ことにより，ＣＰＵネック，メモリネック，及び入出力ネックの解消を図った．第２のアプローチでは，複数の汎用マイクロプロセッサによる並列処理により，ＣＰＵネックの解消を図った．さらに，第１，第２のアプローチで共通して残されるデータネックは，多版管理を用いた並行処理制御により解消を図った．なお，第１，第２のアプローチは互いに排他的ではなく，第１のアプローチにおける汎用計算機を全資源共有型のマルチプロセッサに置き換えることにより，両者の利点を同時に享受できることも述べた．

第２章では，第１のアプローチとして，サーチ処理とソート処理を専用プロセッサで実行する付加型のデータベースプロセッサ方式を提案した．この方式は，汎用計算機の主記憶装置とディスク制御装置の間に入出力インタフェースを介して専用プロセッサを接続することにより，ＣＰＵネックと入出力ネックの両方の解消を図るものである．本章では，まず付加型のデータベースプロセッサ方式を実現する上での設計条件を示した．続いて，提案方式のシステム構成，機能分担，およびその動作手順について，ハードウェアとソフトウェアの両面から述べた．次に，リレーショナルデータベースでは常に性能上の問題となる結合演算の高速化手法を示した．ここでは，結合演算を実行する上での基本となる３フェーズジョイン法とそのバリエーションについて述べ，処理の高速化に大きく寄与するハードウェア化したハッシング関数と，小型で大容量のハードウェアソータについて詳しく説明した．次に，ベンチマークテストにより，専用プロセッサを用いた場合と従来のソフトウェアでデータベース処理を行った場合で，最高１００倍以上の性能差が生じることを示し，その要因を分析した．最後に，提案方式の限界について論じた．

第３章では，第２章で提案した付加型のデータベースプロセッサ方式の実用性を向上させることを目的とした，本体装置上のソフトウェアによるアクセス法選択手法を提案した．付加型のデータベースプロセッサ方式では，専用プロセッサによってすべてのデータベース処理を高速化できる訳ではないため，処理内容に応じて，専用プロセッサを用いるか，従来のソフトウェア処理で行うかを選択するアクセス法選択が必要になる．一方，利用者に対しては，専用プロセッサを意

主 論 文 の 要 旨

報告番号	※乙 第　　号	氏 名	井上　潮

識することなく利用者プログラムを記述できるようにすることが必要であり，データベース管理システムでの自動的なアクセス法選択が重要になる．ここで述べたアクセス法選択手法は，ほとんどコストをかけずに取得できる情報だけを用いる．具体的には，アクセス法選択処理を機能判定部と性能判定部から構成し，機能判定部では，専用プロセッサが機能的に実行できる処理とできない処理を分離する．次の性能判定部では，専用プロセッサによる処理と従来のソフトウェアによる処理をモデル化することにより入出力回数を推定する．その結果，専用プロセッサによる処理が明らかに不適当と推定される場合にのみ，従来のソフトウェアによる処理方式を選択する．このアクセス法選択処理によって，十分実用的に満足できる結果が得られることを示した．

第４章では，第２のアプローチとして，データベースの並列処理を行うプロセッサに対するデータ配分コストを削減する動的な負荷配分量決定法を提案した．全資源共有型のマルチプロセッサは，既存の集中制御によるデータベース処理方式をほとんど変更しなくても複数のプロセッサを効率よく使用できるため，ＣＰＵネックを解消するアプローチとして有効である．しかし，従来方式では負荷，即ち，処理対象となるデータを固定の単位でプロセッサに動的に配分するため，負荷配分に要する時間とプロセッサのアイドル時間の間にトレードオフが生じ，かつこれらの時間はプロセッサ数，データベースの大きさ及びデータの分布に依存するため，配分単位の最適値を事前に決定することが困難であった．この問題を解決するため，本章で提案した方式では処理の進行とともに配分単位を動的に変えていくことにより，プロセッサのアイドル時間を増加させずに負荷配分に要する時間を削減する．プロトタイプ上での性能評価によって，提案方式は従来の方式と比較して高い性能を安定的に得られ，かつその性能がプロセッサ数にほぼ比例する理想的な特性を持つことを述べた．

第５章では，第１のアプローチと第２のアプローチの共通の課題であるデータネックを解消することを目的として，定型処理の代表である部分更新処理と非定型処理の代表である全数検索処理の間の並行処理性能が高い多版並行処理制御方式を提案した．多版並行処理制御は，先に述べたように定型処理と非定型処理が

主 論 文 の 要 旨

報告番号	※乙第　　号	氏 名	井上　潮

混在して実行されるシステムにおけるデータネックの解消手段として極めて有効であるが，従来の方式では定型処理と非定型処理の間の干渉や定型処理相互の間の干渉により，十分な性能が得られないという問題があった．本章では，2版2相ロックの改良方式と，この改良方式と多版時刻印の混成方式の2つの方式を提案した．これらの提案方式は，部分更新処理によって更新されたデータの値はそのデータの更新前の値によってのみ決定されるという性質を利用して，直列可能性を満たすスケジューリングの範囲を拡張するものである．シミュレーションにより評価した結果，提案方式は並行処理される部分更新処理数にかかわらず，部分更新処理の完了率と全数検索処理の完了率の両方について，従来方式より大幅に良好な特性が得られたことを報告した．さらに，複数の部分更新処理が同一データの更新で競合する頻度の高い状態においても，提案方式は従来方式よりも良好であることを示した．

第6章は本研究の総括である．第5章までで述べた本研究の技術範囲と成果についてまとめるとともに，将来のデータベースの利用動向とハードウェアの技術動向を展望することにより，今後データベースの高速処理に関連して研究を展開していくべき方向性について論じた．第2章及び第3章で述べた専用プロセッサ方式は既に商用化され，従来オンライン処理が不可能だった業務が可能になり，利用者から高い評価を受けている．さらに第4章及び第5章で述べた方式の導入により，現在のシステムが抱えている性能上の問題も原理的には解決できる見通しが得られた．今後の研究課題としては，スケーラブルなデータベース処理アーキテクチャ，フォールトトレラントなシステム構成法，及び主記憶データベースの実現法が重要になることを述べた．

大規模リレーショナルデータベースの

高速処理方式に関する研究

井上　潮

(2)

大規模リレーショナルデータベースの

高速処理方式に関する研究

井上　潮

目次

表目次

図目次

1　序論

1.1　はじめに

近年，データベース処理に関する技術が著しい進展を遂げ，計算機応用の基本的な柱として定着してきた．データベースとは，一つの組織体に関するデータを統一された方法でまとめ，異なる業務を行う複数の利用者が共同利用できるようにしたものであり，データベース処理とは，このデータベースの作成，利用，管理に関する種々の操作のことを言う．

データベースの歴史は１９６０年代から始まるが，１９７０年にＩＢＭ社のコッドがリレーショナルモデル[1]を提案して以来，モデルの簡明さと強固な理論的裏付けに支えられて，このモデルに基づいたリレーショナル・データベース管理システムが開発されるようになった．現在，商用になっているリレーショナル・データベース管理システムは，ＩＮＧＲＥＳ，ＯＲＡＣＬＥ，ＩＮＦＯＲＭＩＸ，ＵＮＩＦＹ，ＤＢ２ほか数多く，大型汎用計算機からパーソナル・コンピュータまで非常に広範囲のハードウェア／オペレーティングシステム環境で動作している．また，１９８７年にデータベース言語ＳＱＬがＩＳＯ[2]とＪＩＳ[3]の両方で規格化されたことにより，リレーショナル・データベース管理システムは，益々幅広い分野で利用されるようになった．

一方，リレーショナル・データベース管理システムの開発当初から，データベース処理の高速化は大きな技術的課題になっていた．このため，データベース処理を高速化するために専用化された計算機，即ちデータベースマシンの研究・開発も多数行われてきた．専用化のレベルとしては，単にオペレーティングシステムを含むソフトウェアを効率化するレベルから，並列処理や専用ハードウェアを導入するレベルまで広範囲に及ぶ．初期のデータベースマシンは，ＲＡＰ[4]，ＣＡＳＳＭ[5]，ＤＢＣ[6]のように，専用のハードウェアロジックを付加した特殊な磁気ディスク装置を用いるものが多かった．しかし，これらの装置は記憶

容量が小さく，汎用の磁気ディスク装置に比べてビット当たりの記憶コストが1桁以上高くなるため，実用には程遠かった．次に開発されたデータベースマシンは，ＤＩＲＥＣＴ[7]，ＥＤＣ[8]のように，磁気ディスクの代わりにＣＣＤや磁気バブル等の電子ディスクを用いるものであったが，これらも当初期待していたようには記憶コストが低下しなかったため，やはり実用にはならなかった．現在，実用になっているデータベースマシンは，データベースを汎用の磁気ディスク装置に格納し，その制御装置または汎用プロセッサの主記憶装置に専用のハードウェアロジックを付加するか，または多数の汎用プロセッサによる並列処理を行うかのいずれかの形態となっている．

本論文は，このような状況において，リレーショナル・データベース管理システムにおける大規模なデータベースの処理を飛躍的に高速化させることを目的として行った一連の研究結果を報告する．本研究では，従来のリレーショナル・データベース管理システムにおいて性能上のボトルネックとなっていたＣＰＵネック，入出力ネック，およびデータネックを解消することを目指す．具体的な研究の内容は，次の3点である．

① データベース専用プロセッサ方式
　付加型の専用プロセッサによるＣＰＵネックと入出力ネックの解消
② 並列処理における動的負荷配分方式
　汎用プロセッサの並列処理によるＣＰＵネックの解消
③ 多版管理による並行処理制御方式
　データベースの共用に伴うデータネックの解消

1.2　本研究の背景

データベース処理は，定型処理と非定型処理の2種類に大別できる．定型処理とは，予め決められたキーを用いてデータベース中のデータに対してアクセスする処理であり，非定型処理とは，このようなキーを用いないでアクセスを行う処理である．具体的な例として，銀行の預金システムを考えてみる．預金システムでは，銀行の顧客ごとに，顧客名，口座番号，住所，電話番号等を記録した顧客

表と，預金口座ごとに，口座番号，預金残高，最終更新日付等を記録した口座表がある．この預金口座に対する預け入れ，払い戻し処理は定型処理の典型的な例であり，口座を一意に識別する口座番号をキーとして口座表の1行にアクセスして，預け入れまたは払い戻した金額に応じてその行の預金残高と最終更新日付を更新する．このような定型処理では，キーを用いることにより1回の操作においてデータベースの表中の単一の行のみにアクセスすることが特徴となる．一方，非定型処理の例としては，顧客サービスのために預金残高が一定の金額以上の口座を検索する処理や，業務合理化のために過去1年間に全く取り引きのなかった顧客を検索する処理等が考えられる．このような非定型処理では，キーを用いないために1回の操作で一般に多数の行にアクセスすることが特徴となる．

従来のリレーショナル・データベース管理システムでは，キーとして使用する列について予めＢツリー[9]やハッシング[10]等によって構成されたインデックスを作成しておき，行へのアクセス時にインデックスから行のアドレスを求めることにより，定型処理に関しては十分短い時間内で処理を完了できるようにしている．一方，非定型処理に関しては，インデックスの効果的な利用が困難なために，大規模なデータベースでは非常に長い処理時間がかかっている．このため，リレーショナル・データベースを用いた従来のオンラインシステムでは，定型処理のみをオンライン・リアルタイムで行い，非定型処理はオフライン・バッチで実行する形態が一般的である．しかし，戦略情報システム（ＳＩＳ）に代表される高度なデータベース応用システムが開発されるのに伴って，リレーショナル・データベースが本来持つ集合論的な操作を含む非定型処理をオンラインで実行したいという利用者の要求が，急速に高まってきている．

本研究は，このような背景をもとに，リレーショナル・データベースにおける非定型処理を高速化すること，および定型処理と並行して実行できるようにすることを目的とするものである．

1.3　本研究の問題設定

データベースの処理を行うシステムは，一般的には図1.1に示す構成をとる．

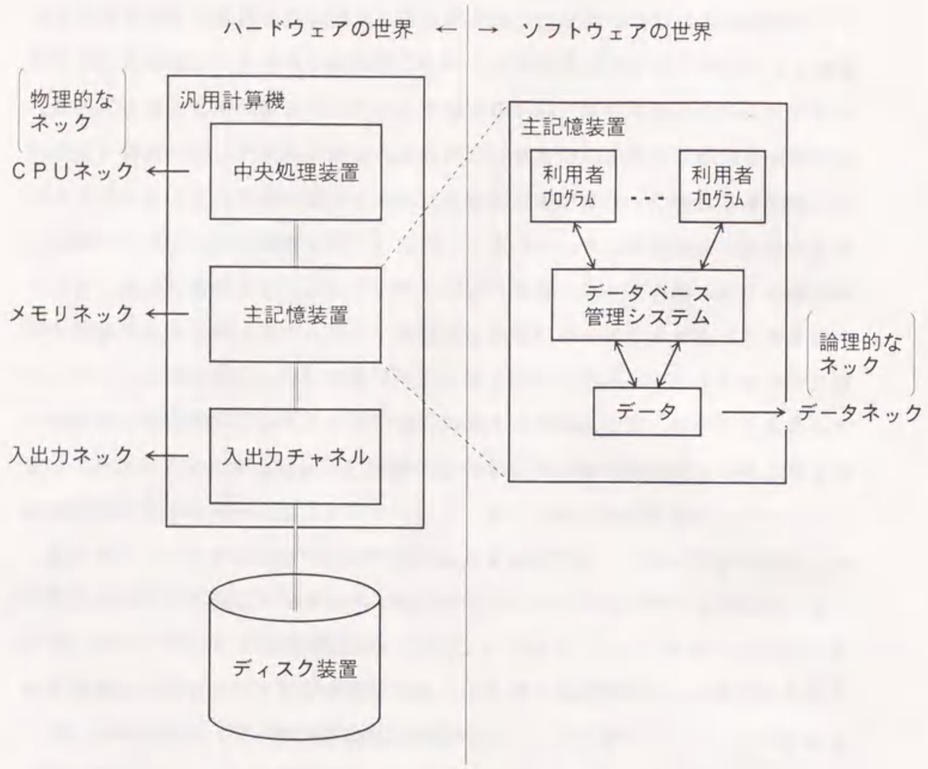


図1.1　データベースシステムの構成
Figure 1.1　Database system organization

ハードウェアの側面から見たシステムの基本的な構成要素は，データベース処理を実行する中央処理装置，中央処理装置で操作されるデータを一時的に保持する主記憶装置，データベースを永続的に保存するディスク装置と，主記憶装置とディスク装置を接続する入出力チャネルである．このようなシステムにおいてデータベースの高速処理を行おうとした場合，中央処理装置，主記憶装置，入出力チャネルの3つが性能上のボトルネックとなりうる．以下，それぞれをＣＰＵネック，メモリネック，入出力ネックと呼ぶ．さらにソフトウェアの側面から見ると，複数の利用者プログラムがデータベース管理システムを経由して同一のデータに対するアクセスを行うため，このアクセスの競合によって性能上のボトルネックが生じ得る．以下，これをデータネックと呼ぶ．

データベース処理において，これらのネックが生じる原因を以下に示す．

(a) ＣＰＵネック

① 従来のプロセッサは数値計算向きに設計されており，データベース処理のような非数値処理への適用はあまり考慮されていない．

② データベース処理でも特に非定型処理では，大量のデータをプロセッサが処理しなければならない．

(b) メモリネック

③ 中央処理装置の速度に比べて主記憶装置のアクセス速度は数倍から1桁程度遅い．

④ 非定型処理では，大量のデータを主記憶装置から中央処理装置へ供給しなければならない．

(c) 入出力ネック

⑤ 主記憶装置の速度に比べてディスク装置のアクセス速度は2桁から3桁程度遅い．

⑥ 非定型処理では，大量のデータをディスク装置から主記憶装置へ供給しなければならない．

(d) データネック

⑦ 複数の利用者が同じデータを共同利用するためには，利用者からのデータベースアクセス要求を直列化する必要がある．

⑧ 非定型処理では，単一の利用者からの要求によって大量のデータが長時間

使用されるため，定型処理におけるデータ更新を妨げる頻度及び期間が長い．

本研究では，以上述べた物理的なボトルネックであるＣＰＵネック，メモリネック，入出力ネックと，論理的なボトルネックであるデータネックについて，問題点の解決を図るものである．

1.4　従来の研究

1.4.1　ＣＰＵネックの解消

ＣＰＵネックが生じる原因は，①従来のプロセッサは非数値処理への適用をあまり考慮されていない，②大量のデータをプロセッサが処理しなければならないためであることは前に述べた．ＣＰＵネックを解決するためのアプローチとしては，次の３種類が考えられる．

- データベース処理のための専用プロセッサの開発
- 複数の汎用プロセッサによる並列処理の実現
- アルゴリズムの改良による処理データ量の削減

(1) 専用プロセッサの開発

データベース処理，特に非定型処理を行う上で，汎用プロセッサにとって大きな負担となるのはサーチ処理とソート処理である．従って，これまでに開発されたデータベース処理のための専用プロセッサは，サーチ処理とソート処理のいずれかを高速化するものがほとんどである．また，サーチ処理については，もともとデータが格納されている場所の近くで実行することが，入出力ネックの解消にもつながるため，ディスクのインテリジェント化と一体で実現されている．これについては，1.4.3節で詳しく述べることにする．

一方，ソート処理については，データ数をＮとした場合，汎用プロセッサでは$N \times \log_2 N$回の比較が必要になり，データ数が大きい場合には処理時間が大幅に増加していく．そこで，Ｎまたは$\log_2 N$に比例した個数のプロセッシングエレメントを用いることにより，データ数Ｎに比例した時間内で処理が可能なハードウェアソータが開発されている．例えば，バブルソート・アルゴリズムに基づくバブルソータ[11]はＮ/2個のエレメントを用いるものであり，マージソートアルゴリズムに基づくマージソータ[12]は$\log_2 N$個のエレメントを用いるものである．このようなソータをベースとした専用プロセッサの一例がＧＲＥＯ[13]である．ＧＲＥＯは，マージソータにマイクロプロセッサを付加した構成となっており，主にソート処理とインデックス生成の高速化に寄与している．

また別のアプローチとして，科学技術計算を高速処理するために用いられているベクトル処理をデータベースに適用する試みもある．ＩＤＰ[14]はその一例であり，既存のアレイプロセッサを拡張することにより，ベクトル形式で主記憶装置上に展開されたデータに対してソート処理をはじめとするリレーショナルデータベースの各種演算を実行する．さらに別のアプローチとして，シストリックアレイ[15]上でデータベース処理を行うものもある．シストリックアレイは，１ビット比較器からなるセルを一次元または二次元状に配列し，データをシフトさせながら選択演算や結合演算を実行する．

以上述べた方式はすべて，処理対象となるデータ量に見合ったハードウェア量を必要とするため，コスト性能比の良いシステムを実現するためには，他の手段と組み合わせることにより，データ量を削減することが重要である．

(2) 並列処理の実現

リレーショナルデータベースは，もともと互いに独立なデータの集合として定義されているため，データ単位での処理の分割が容易であり，並列処理に向いている．従って，リレーショナルデータベースの各種演算を複数のプロセッサで並列に実行するアルゴリズム[16][17]が各種提案されている．一方，最近の汎用マイクロプロセッサの高性能化と低コスト化により，多数の汎用プロセッサを用いた並列処理システムを実際に構築することが可能になってきた．並列処理システムには，各プロセッサ毎に独立した主記憶とディスクを持ち，プロセッサ間の通信はメッセージの送信によってのみ行う資源非共有型と，各プロセッサが主記憶とディスクの両方を共有し，プロセッサ間の通信を主記憶上のアドレスで行うこ

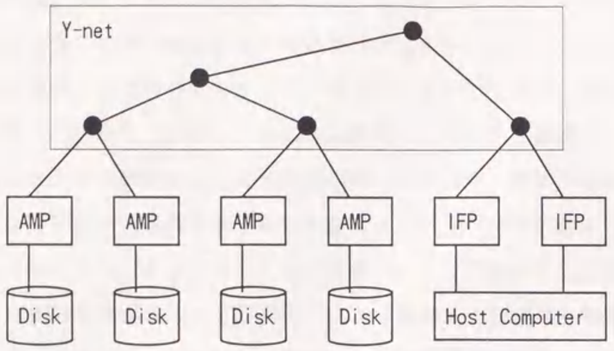


図1.2　ＴＥＲＡＤＡＴＡのアーキテクチャ
Figure 1.2　Architecture of TERADATA

とが可能な全資源共有型，及び両者の中間で，主記憶は独立でディスクのみを共有するディスク共有型の３種類がある．それぞれには一長一短があるが，資源非共有型及びディスク共有型はプロセッサ間の同期のために分散型の制御が必要になるのに対し，全資源共有型は単一プロセッサのシステムと同じ集中型の制御が可能であり実現が容易である．また，資源非共有型ではプロセッサ間の負荷バランスがデータベースの配置状態によって支配されるのに対し，全資源共有型及びディスク共有型ではプロセッサ間の動的な負荷の均衡化が可能である．現時点で，並列処理を実現しているデータベースマシンとしては，ＧＡＭＭＡ[18]とＴＥＲＡＤＡＴＡ[19]がある．ＧＡＭＭＡは１７台のミニコンピュータをリング接続した構成であり，ＴＥＲＡＤＡＴＡは図1.2に示すように３～１０２４個までのマイクロプロセッサをＹ－ｎｅｔと呼ばれるツリー状のネットワークで接続した構成である．いずれのマシンも資源非共有型であるため，プロセッサ間での負荷の不均衡が生じてプロセッサ全体の処理能力を十分に利用できなくなる可能性があるという問題がある．

(3) アルゴリズムの改良

リレーショナルデータベースの演算の中で，結合演算は処理量が極めて大きいものの１つである．結合対象となる２つの表の行数をそれぞれＮ，Ｍとすると，単純なネステッドループ・ジョイン法ではＮ×Ｍ回の比較処理が必要になる．しかし，２つの表がソートされていれば必要な比較回数はＮ＋Ｍとなる．別の方法として，ハッシュ・ジョイン法[20]が提案されている．この方法は，２つの表を同じハッシング関数で分割し，同一の分割単位に属する行のみを比較対象とすることにより，比較回数をＮ＋Ｍに近いレベルにまで削減する．ただし，片方の表をすべて主記憶上に展開できない場合にはネステッドループ・ジョイン法と同じような処理が部分的に必要となり，比較回数は多くなる．また，以上の方法とは独立に，ハッシュ化ビットアレイ法[21]がある．この方法は，ジョインキーとなる列のハッシュ値をビットアレイに登録することにより，一方または両方の表から結合演算の結果として残らない行を，結合演算の実行に先立って除去するものである．従って，Ｎ，Ｍの値を予め小さくすることにより，プロセッサが行わな

ければならない比較回数を削減する効果がある．

1.4.2　メモリネックの解消

メモリネックが生じる原因は，①中央処理装置に比べて主記憶装置へのアクセス速度が遅い，②大量のデータを中央処理装置へ供給しなければならないためであった．このメモリネックを解決するためのアプローチとしては，次の2種類が考えられる．

- メモリアーキテクチャの見直しによるアクセスの高速化
- メモリのインテリジェント化による転送データ量の削減

(1) メモリアクセスの高速化

メモリへのアクセス速度を向上させるために一般に用いられている方法は，キャッシュとインターリブである．キャッシュは，主記憶装置に用いられているメモリ素子よりも高速の素子を用い，使用頻度の高い命令またはデータを保持することにより，主記憶へのアクセス回数を削減する技法であり，インターリブは，命令またはデータをビット単位で複数のメモリ素子に分割して記憶させることにより，主記憶へのデータ転送時間を短縮する技法である．これらは，計算機システム一般の性能改善には役立つが，データベース処理に対して特に有効となるものではない．特にキャッシュは，メモリアクセスが局所的な場合に大きな効果があるが，非定型処理のように大量のデータを順に処理していく場合にはほとんど効果を期待できない．

(2) メモリのインテリジェント化

連想メモリ（ＣＡＭ，Content Addressable Memory）は，メモリをインテリジェント化した形態の１つである．連想メモリは，通常の半導体メモリ（ＲＡＭ）のようなアドレスを用いた検索ではなく，検索したいデータの内容によりメモリ

セル内のデータを並列に検索できる素子である．ＶＬＳＩ技術の発達により，最近では２０Ｋビット程度の容量を持つ連想メモリ[22]が実現されている．しかしながら，データベース処理への適用を考えた場合，ＲＡＭと比較して容量が小さい，記憶コストが高い，可変長データの取り扱いが困難という問題があり，実用には程遠い．

1.4.3　入出力ネックの解消

入出力ネックが生じる原因は，①主記憶装置に比べてディスク装置へのアクセス速度が格段に遅い，②大量のデータを主記憶装置へ供給しなければならないためであった．入出力ネックを解決するためのアプローチとしては，次の3種類が考えられる．

- データベース処理向きの高速ディスク装置の開発
- 複数の汎用ディスク装置による並列入出力の実現
- ディスク装置のインテリジェント化による転送データ量の削減

(1) 高速ディスク装置の開発

高速な入出力が可能なディスクとして，ＣＣＤ（Charge Coupled Device）や磁気バブル等を用いた電子ディスクが一時期有力視され，ＤＩＲＥＣＴ[7]やＥＤＣ[8]のようなデータベースマシンが開発された．ＤＩＲＥＣＴは１６ＫバイトのＣＣＤメモリをクロスバスイッチでプロセッサと結合した構成であり，ＥＤＣは１２８Ｋバイトの磁気バブルメモリとプロセッサの組をバスで結合した構成であった．しかし現在では，半導体メモリ（ＲＡＭ）のコストダウンが著しく，ＣＣＤや磁気バブルはほとんど使用されなくなった．最近の汎用計算機システムでは，小容量で高速なアクセスが要求されるデータを格納するために，半導体メモリを用いたディスクキャッシュや半導体ディスクが用いられるようになった．しかしながら，半導体メモリは電源が切れると記憶している内容が失われるという本質的な問題と，経済的にもこれらの装置の記憶コストが磁気ディスクに比べ

て約１桁高いという問題のために，大容量のデータベースをこれらの装置に記憶させることは現実的ではない．

(2) ディスク入出力の並列化

複数の汎用ディスク装置に対する入出力を並列化する方法には２種類がある．１つ目は，複数のディスクの回転をすべて同期させ，メモリと同じようにバイトまたはセクタ単位でデータをインターリブした状態で入出力を行う方法である．ＲＡＩＤ[23]はこれを実現した代表例であり，複数ディスクからの並列転送による高いデータ転送速度と冗長なディスクの設置による高い可用性を同時に実現している．この方法の難点は，複数のディスクを同期させるためにディスク制御装置が複雑になることと，スキューの問題によりあまり多数のディスクを並列化できないことである．２つ目は，単にデータを複数のディスク装置に分散格納しておき，それぞれのディスクに並行してアクセスを行う方法であり，ディスク間の同期は特に制御しない．ＧＡＭＭＡ[18]やＴＥＲＡＤＡＴＡ[19]はこの方法を用いたデータベースマシンであり，データ値のレンジ，ハッシング，ラウンドロビンのいずれかにより，データを複数のディスクに分散させる．この方法は，通常のディスク制御装置が使用できること，並列化できるディスク数に理論的な限界がないことから大規模なシステム向きであるが，データの更新時にはソフトウェアによるディスク間でのデータ書き込みの同期制御が必要になる．

(3) ディスクのインテリジェント化

データベースマシンの研究が始まった当初から，ディスク装置をインテリジェント化し，この上でデータベース処理，特にサーチ処理を実行させる試みがなされてきた．その目的は，データが格納されているディスク上でサーチ処理を行うことにより，不要なデータがディスク装置から主記憶装置へ転送されることを防ぐことにある．例えば，ＲＡＰ[4]やＣＡＳＳＭ[5]は，固定ヘッド磁気ディスクのトラックごとにプロセッサを設けた構成であり，ディスクが１回転する間にディスク内のすべてのデータに対する処理を行う．また，ＤＢＣ[6]では，可動

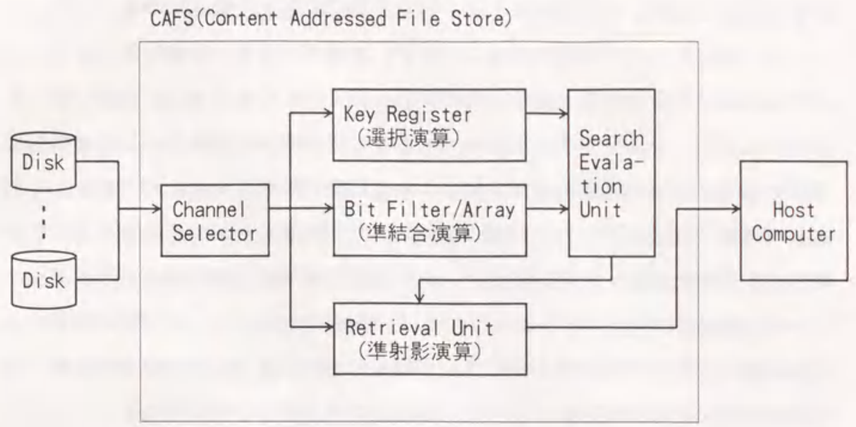


図1.3　ＣＡＦＳのアーキテクチャ
Figure 1.3　Architecture of CAFS

ヘッド磁気ディスクのヘッドごとにプロセッサを設け，ディスクが１回転する間に１つのシリンダ内のすべてのデータに対する処理を行う．しかし，これらの方式は磁気ディスク装置そのものを専用化するため，汎用の磁気ディスク装置を使用する場合と比較して記憶コストが１桁以上高くなり，実用的ではなかった．

一方，磁気ディスク装置には手を入れず，磁気ディスクの制御装置にインテリジェンスを付加する方法も実現されている．例えば，ＣＡＦＳ[24]は図1.3に示すようにリレーショナルデータベースにおける選択演算，準結合演算及び準射影演算に相当する処理機能を磁気ディスク制御装置に組み込んだものであり，１個の制御装置で複数のディスク装置に格納されたデータを逐次的に処理することができる．この方法は，汎用の磁気ディスク装置をそのまま使用するために記憶コストが小さく，大容量のデータベースへの適用が可能である．ここでの問題点は，制御装置に付加された機能がディスク装置上のデータに対してのみ働くため，結合演算のような複雑な機能を単独で実現することが難しい点である．

また，サーチ処理の中でも特に処理量が大きいテキストサーチすなわち文字列検索処理に関しても各種の専用ハードウェア[25]が研究されている．テキストサーチを実現する主な方法には，文字列比較器を並列に設ける方法，１文字比較器を直列に接続する方法，有限オートマトンを使用する方法がある．しかし，複雑な検索条件を処理させようとした場合，前２者はハードウェア量が大きくなり，後者は処理速度が低下するという問題がある．なお，現在のリレーショナルデータベースでは，このテキストサーチ処理機能はあまり使用されていないが，標準化されたデータベース言語ＳＱＬにはＬＩＫＥ述語という形でこの機能が取り入れられており，非定型処理の一部として今後は使用されるケースが多くなると予想される．

1.4.4　データネックの解消

データネックが生じる原因は，①利用者からのアクセス要求を直列化する必要がある，②非定型処理では大量のデータが長時間占有されることにあった．従って，データネックを解決するためのアプローチは，次の２種類が考えられる．

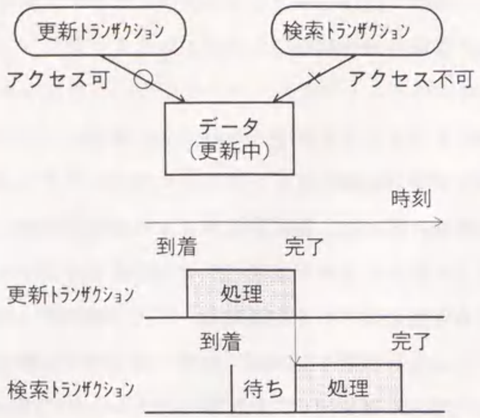
- データへのアクセス順序制御によるデータ占有時間短縮
- データの多版管理による直列化自由度の向上

(1) データへのアクセス順序制御

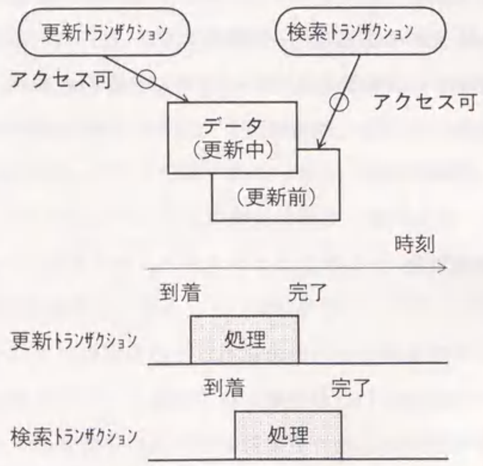
データベースにアクセスする一連の手続きを，トランザクション[26]と呼ぶ．トランザクションの実行に際してデータベース管理システムは，ＡＣＩＤ性，即ち，原子性，一貫性，孤立性，耐久性を保証する必要がある．これらの性質は，トランザクションに対して直列可能性[27]に基づくスケジューリングを行うことにより満足できる．先に述べた定型処理は，ごく少数のデータを参照または更新するトランザクション，非定型処理は，非常に多数のデータを参照するトランザクションである．現在一般に用いられている２相ロック方式[27]では，非定型処理によって多数のデータに対して確保されたロックの影響で，定型処理の進行が長時間にわたり妨げられる．この問題に対するアプローチとして，予めトランザクションの直列可能性を満たすデータへのアクセス順序を規定し，この順に従うロック要求のみを認める方法[28]が提案されている．この方法はトランザクションがアクセスするデータの集合を予め宣言する必要があるが，一般にはトンザクションの実行段階になるまで検索条件またはキーの値が決まらないため，汎用性に欠けるという問題がある．

(2) データの多版管理

定型処理と非定型処理の並行性を高めるための別のアプローチとして，データを多版管理する方式[29][30]が提案されている．これらの方式では，トランザクションがデータを更新する際に更新前のデータに更新後のデータを上書きするのではなく，図1.4に示すように更新前のデータを残した状態で更新後のデータに対する新しい版を作成する．従って，１つのデータはその更新履歴に相当する複数の版によって構成されることになり，直列可能性を満たすスケジューリングの自由度が増す．例えば，あるデータの最新の版に対して定型処理が更新を行っている際に，同じデータの１つ前の版を非定型処理が検索を行っても，両者の直列



(a) データの単版管理



(b) データの多版管理

図1.4　データの単版管理と多版管理

Figure 1.4　Single version and multiple version control

可能性を保証できるようになる．この多版管理を実現する上では，1つのデータに複数の版があることを利用して直列可能性を満足するトランザクションのスケージュル・アルゴリズムと，複数の版を維持することに伴うオーバヘッドが問題となるが，現時点ではまだ前者について各種の提案がなされている段階であり，後者を検討して具体的なシステムにインプリメントできるようになるまでにはもう少し時間が必要である．

1.5　本研究の考え方

1.5.1　従来技術に対する考え方

前節で述べた従来の研究状況に対して，本研究では以下のように考える．

(1) ＣＰＵネック

汎用プロセッサにとって負担の大きいソート処理をハードウェアソータで実行することは，ＣＰＵネックの問題を解決する上で極めて有効な方法である．ハードウェアソータ自体のアルゴリズムやアーキテクチャはほぼ研究されつくした観があるため，本研究ではむしろ，ハードウェアソータをいかに効率良くデータベース処理に適用するかに重点を置く．具体的には，小さなハードウェアソータで大容量のデータベースを処理できるようにするため，アルゴリズムの工夫によってソート対象となる全体のデータ量を削減するとともに，データを一定量の大きさに分割して繰り返し処理を行う．

一方，最近のマイクロプロセッサの高性能化を考えると，ハードウェアソータのような専用ハードウェアに頼らずに，汎用プロセッサの並列処理によってＣＰＵネックの解決を図る方法の有効性は今後大きくなることが確実である．リレーショナルデータベースの各種演算の並列処理アルゴリズムの研究は進んでいるので，本研究では演算実行時に各プロセッサをいかに効率よく働かせるかに重点を置く．具体的には，全資源共有型のマルチプロセッサシステムにおいて，プロセ

ッサ間の負荷の不均衡が生じない動的な負荷制御の実現を図る．

(2) メモリネック

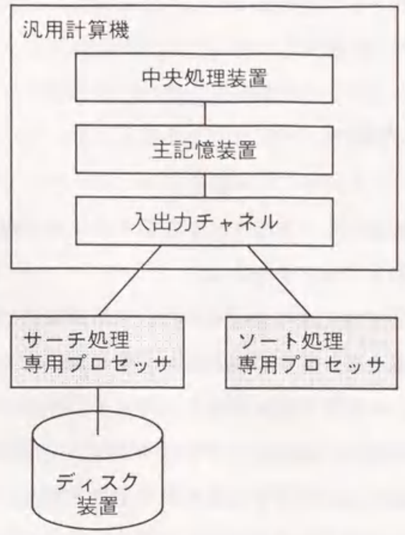
現状の技術レベルでは，メモリネックの問題を直接的に解決することは困難であると考える．本研究では，入出力ネックの解決策と合わせて，ディスク装置から主記憶装置に転送されるデータ量を削減することにより，結果として主記憶装置から中央処理装置に転送されるデータ量も削減し，メモリネックが生じないようにすることを図る．

(3) 入出力ネック

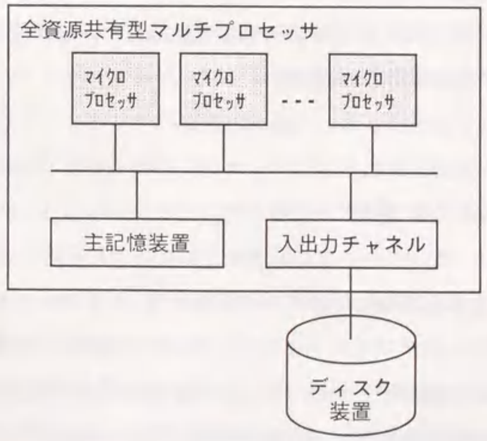
ディスク装置そのものを高速化またはインテリジェント化することは，記憶コストの増大につながり，大容量のデータベースに対する適用性が失われると考える．本研究では，汎用の磁気ディスク装置を用いることを前提条件とし，ディスク制御装置のインテリジェント化と入出力の並列化によって入出力ネックの問題の解決を図る．前者に関しては，ディスク装置から主記憶装置に転送されるデータ量をできる限り削減することを目指し，必要な機能をディスク制御装置に付加する．後者に関しては，高速化が必要な非定型処理ではデータの更新がないことに着目して，データを複数のディスクに分散配置し，各ディスクに対して非同期アクセスする方法を用いる．

(4) データネック

データの管理を多版化することは，原理的にトランザクションの直列可能性を満たすスケジューリングの自由度を増すことができるため，データネックの問題を解決する上で極めて有効な方法であると考える．データの多版化に伴う最大の問題は，データベース処理で必要となる主記憶容量が増大する点であるが，今後とも半導体メモリの大幅なコストダウンが続くことが予想されるため，将来的にはほとんど問題にはならないと考えてよい．本研究では，多版化によって実現し



(a) 第1のアプローチ



(b) 第2のアプローチ

図1.5　本研究の方針
Figure 1.5　Basic approaches of the study

得る定型処理と非定型処理間の並行性を最大限に引き出すことが可能な並行処理制御アルゴリズムについて検討する．

1.5.2　研究方針

本研究では，大規模リレーショナルデータベースの高速処理を実現するために，図1.5に示す2つのアプローチをとる．

第1のアプローチでは，通常のデータベース管理システムが搭載された汎用計算機のシステムに対して，サーチ処理とソート処理のための専用プロセッサを付加することにより，非定型処理で問題となるＣＰＵネック，メモリネック及び入出力ネックの解消を図る．第2のアプローチでは，全資源共有型のマルチプロセッサ上での並列処理によってＣＰＵネックの解消を図る．なお，第1のアプローチと第2のアプローチは互いに排他的ではなく，第1のアプローチにおける汎用計算機を全資源共有型のマルチプロセッサに置き換えることにより，両者の利点を同時に享受することも可能である．さらに，第1，第2のアプローチで共通的に使用できる方法として，多版管理による定型処理と非定型処理の並行制御の導入によるデータネックの解消を図る．

1.5.3　本論文の構成

本論文は，第2章以降を以下のように構成する．

第2章では，第1のアプローチとして，サーチ処理とソート処理を専用プロセッサで実行する付加型のデータベースプロセッサ方式を提案する．この方式は，汎用計算機の主記憶装置とディスク制御装置の間に入出力インタフェースを介して専用プロセッサを接続することにより，ＣＰＵネックと入出力ネックの両方の解消を図るものである．本章では，まず付加型のデータベースプロセッサ方式を実現する上での設計条件を示す．続いて，提案方式のシステム構成，機能分担，およびその動作手順について，ハードウェアとソフトウェアの両面から述べる．

次に，リレーショナルデータベースでは常に性能上の問題となる結合演算の高速化手法を示す．ここでは，結合演算を実行する上での基本となる3フェーズジョイン法とそのバリエーションについて述べ，処理の高速化に大きく寄与するハードウェア化したハッシング関数と，小型で大容量のハードウェアソータについて詳しく説明する．次に，ベンチマークテストにより，専用プロセッサを用いた場合と従来のソフトウェアでデータベース処理を行った場合で，最高１００倍以上の性能差が生じることを示し，その要因を分析する．最後に，提案方式の限界について論じる．

第3章では，第2章で提案した付加型のデータベースプロセッサ方式の実用性を向上させることを目的とした，本体装置上のソフトウェアによるアクセス法選択手法を提案する．付加型のデータベースプロセッサ方式では，専用プロセッサによってすべてのデータベース処理を高速化できる訳ではないため，処理内容に応じて，専用プロセッサを用いるか，従来のソフトウェア処理で行うかを選択するアクセス法選択が必要になる．一方，利用者に対しては，専用プロセッサを意識することなく利用者プログラムを記述できるようにすることが必要であり，データベース管理システムでの自動的なアクセス法選択が重要になる．ここで述べるアクセス法選択手法は，ほとんどコストをかけずに取得できる情報だけを用いる．具体的には，アクセス法選択処理を機能判定部と性能判定部から構成し，機能判定部では，専用プロセッサが機能的に実行できる処理とできない処理を分離する．次の性能判定部では，専用プロセッサによる処理と従来のソフトウェアによる処理をモデル化することにより入出力回数を推定する．その結果，専用プロセッサによる処理が明らかに不適当と推定される場合にのみ，従来のソフトウェアによる処理方式を選択する．このアクセス法選択処理によって，十分実用的に満足できる結果が得られることを示す．

第4章では，第2のアプローチとして，データベースの並列処理を行うプロセッサに対するデータ配分コストを削減する動的な負荷配分量決定法を提案する．全資源共有型のマルチプロセッサは，既存の集中制御によるデータベース処理方式をほとんど変更しなくても複数のプロセッサを効率よく使用できるため，ＣＰＵネックを解消するアプローチとして有効である．しかし，従来方式では負荷，即ち，処理対象となるデータを固定の単位でプロセッサに動的に配分するため，

負荷配分に要する時間とプロセッサのアイドル時間の間にトレードオフが生じ，かつこれらの時間はプロセッサ数，データベースの大きさ及びデータの分布に依存するため，配分単位の最適値を事前に決定することが困難であった．この問題を解決するため，本章で提案する方式では処理の進行とともに配分単位を動的に変えていくことにより，プロセッサのアイドル時間を増加させずに負荷配分に要する時間を削減する．プロトタイプ上での性能評価によって，提案方式は従来の方式と比較して高い性能を安定的に得られ，かつその性能がプロセッサ数にほぼ比例する理想的な特性を持つことを述べる．

第５章では，第１のアプローチと第２のアプローチの共通の課題であるデータネックを解消することを目的として，定型処理の代表である部分更新処理と非定型処理の代表である全数検索処理の間の並行処理性能が高い多版並行処理制御方式を提案する．多版並行処理制御は，先に述べたように定型処理と非定型処理が混在して実行されるシステムにおけるデータネックの解消手段として極めて有効であるが，従来の方式では定型処理と非定型処理の間の干渉や定型処理相互の間の干渉により，十分な性能が得られないという問題があった．本章では，２版２相ロックの改良方式と，この改良方式と多版時刻印の混成方式の２つの方式を提案する．これらの提案方式は，部分更新処理によって更新されたデータの値はそのデータの更新前の値によってのみ決定されるという性質を利用して，直列可能性を満たすスケジューリングの範囲を拡張するものである．シミュレーションにより評価した結果，提案方式は並行処理される部分更新処理数にかかわらず，部分更新処理の完了率と全数検索処理の完了率の両方について，従来方式より大幅に良好な特性が得られたことを報告する．さらに，複数の部分更新処理が同一データの更新で競合する頻度の高い状態においても，提案方式は従来方式よりも良好であることを示す．

第６章は本研究の総括であり，第５章までで述べた本研究の技術範囲と成果についてまとめるとともに，将来のデータベースの利用動向とハードウェアの技術動向を展望することにより，今後データベースの高速処理に関連して研究を展開していくべき方向性について述べる．

1.6　用語の定義

本論文で用いる用語の定義を以下に示す．なお「列」「行」「表」の定義は，ＪＩＳ規格[3]に従っている．

「列」：時間がたてば変わる値のマルチ集合（必ずしも互いに異ならない対象の順序づけられていない集まり）とする．同じ列のすべての値は，同じデータ型（表現可能な値の集合）とし，同一の表中の値とする．１つの列の１つの値は，表から選択できるデータの最小単位であり，更新できるデータの最小単位である．

「行」：値の並びとし，その並びは空ではない．同じ表のすべての行は，同じ基数を持ち，その表のすべての列の値を含む．表のすべての行のｉ番目の値は，その表のｉ番目の列の値とする．行は，表に挿入でき，表から削除できるデータの最小単位である．

「表」：行のマルチ集合とする．表の次数は，その表の列の数とする．表の次数は，あらゆる時点において，それの各行の基数と同じとし，表の基数は，それの各列の基数と同じとする．

「ページ」：データを格納する固定の大きさを持つ領域とする．原理的には，ページ対表，ページ対行の関係はともに多対多の関係が可能であるが，本論文では議論を単純にするために，行対ページは多対１，ページ対表は多対１の関係のみを扱う．

「データベース」：論理的には，表および表に付帯するすべてのデータの集まりである．表に付帯するデータには，表の構造を定義したスキーマや，表へのアクセスを高速化するためのインデックス等が含まれる．物理的には，ページの集まりである．本論文では，ページの実体はディスク装置上に格納され，必要に応じてそのコピーが主記憶装置上に転送されるものとする．

「データベース管理システム」：複数種類の利用者がデータベースを利用できるようにするためのソフトウェアであり，オペレーティングシステムと利用者が作成するユーザプログラムの中間に位置する．データベース処理のために使用されるハードウェアは含まれない．

2　データベース専用プロセッサ方式

2.1　まえがき

本章では，前章で述べたＣＰＵネック，メモリネック，及び入出力ネックを解消するための第１のアプローチとして，汎用計算機にサーチ処理とソート処理のための専用プロセッサを付加したデータベースマシン方式（データベース専用プロセッサ方式）について述べる．

データベースマシンの開発は，研究段階から実用化段階へと移ってきた．一般にデータベースマシンは，汎用計算機と専用プロセッサ間の機能分担により，後置型と付加型の２つのグループに分類することができる．後置型のデータベースマシンは，汎用計算機とは独立かつ自己完結的にデータベース処理を行い，汎用計算機に対するバックエンドプロセッサとして利用される．例えば，ＩＤＭ／ＢＬシリーズ[31]，ＴＥＲＡＤＡＴＡ[19]が後置型の分類に入る．

一方，付加型のデータベースマシンは，データベース処理機能のうちの一部を専用化し，特定の汎用計算機に付加または内蔵したものであり，汎用計算機上のデータベース管理システムにより制御される．付加型のデータベースマシンは，機能の専用化の範囲と手段によって種々の実現形態をとり得る．例えば，ＣＡＦＳ[24]は，ディスク中にファイルとして格納されたデータのサーチとふるい落とし機能を専用化したものであり，検索用のハードウェア論理を組み込んだ専用のディスク制御装置として実現されている．ＧＲＥＯ[13]は，主記憶上に読み上げられたデータに対するソートとインデックスの生成処理機能を専用化したものであり，ハードウェアのマージソータと３個のマイクロプロセッサにより実現されている．また，ＩＤＰ[14]も主記憶上のデータに対するソートと基本的な関係演算機能を専用化したものであるが，実現方法はＧＲＥＯとは異なり，汎用計算機に内蔵されたベクトルプロセッサをデータベース処理用に拡張することによって実現されている．一般に，付加型のデータベースマシンは汎用計算機の性能ネッ

クとなっている機能のみを専用ハードウェア化するため，小さな追加コストで大きな性能向上が実現できるという利点がある．

著者らが開発したＲＩＮＤＡ（RelatIoNal DAtabase processor)は，リレーショナルデータベースに対するインデックスの利用が困難な非定型処理を高速化することを目的とする付加型のデータベースマシンである．具体的には，非定型処理を実行する際に汎用計算機にとって負担が重いサーチ，ソート等の処理を専用ハードウェアが分担して高速に実行する．これにより，従来のシステムではオンライン・リアルタイムでの処理が困難であった大規模データベースに対する非定型処理を最高約１００倍にまで高速化することを可能にした．

ＲＩＮＤＡは汎用計算機ＤＩＰＳシリーズ[32][33]と入出力インタフェースにより接続され，ＤＩＰＳ上のソフトウェアのデータベース管理システム（ＤＥＩＭＳ－５：DEnden Information Management System-5）によって制御される．以下，ＲＩＮＤＡの接続された汎用計算機をホスト計算機，データベース管理システムのうち，ＲＩＮＤＡを制御するサブシステムをＲＣＰ（ＲＩＮＤＡ制御プログラム）と呼ぶ．ＲＩＮＤＡが実現する機能は非定型処理のみであり，その他の機能，例えばデータベースの生成処理や，定型処理は従来通りデータベース管理システムによって実現される．ユーザプログラムを作成するためのインタフェースはＳＱＬ[2][3]に準拠したものであり，ユーザプログラムからＲＩＮＤＡを特に意識する必要はない．また，ホスト計算機およびデータベースが格納されるディスクとディスク制御装置は汎用のものがそのまま使用できるため，既存ユーザの設備を無駄にすることなくＲＩＮＤＡを導入することが可能である．

本章では，ＲＩＮＤＡのアーキテクチャと実現方式について，ハードウェアとソフトウェアの両面から議論する．以下，2.2節では，従来のリレーショナルデータベース管理システムの問題点，ＲＩＮＤＡによる解決方針，及びＲＩＮＤＡ開発に際しての設計条件を述べる．2.3節では，ＲＩＮＤＡのハードウェアとこれを制御するデータベース管理システムの構成を述べる．2.4節では，非定型処理の典型ともいえる結合演算のＲＩＮＤＡによる実現方式を示す．2.5節では，ＲＩＮＤＡによる性能向上効果についてベンチマーク評価の結果と分析を示すとともに，ＲＩＮＤＡ方式の利点と限界について論じる．

2.2　基本概念

2.2.1　従来システムの問題点

リレーショナルデータベースの処理は，定型処理と非定型処理に大別できることは前章で述べた．定型処理の代表的な例は，表の中からユニークなキーを用いて単一の行を選択する処理であり，インデックスを用いることによって従来のソフトウェアでも十分に実用に耐えうる性能が実現されている．一方の非定型の検索処理には，予めインデックスの作成されていない列に対して条件を設定した検索や，列内の文字列データに対する任意文字列の部分一致検索等があり，これらを従来のソフトウェアのみのデータベース管理システムで実行すると多大の処理時間がかかっていた．その原因の１つは，インデックスを使用できないために表の中の全ての行を逐次的に読み出して検索条件の判定を行うサーチ処理に多大のＣＰＵ時間とＩ／Ｏ時間がかかることである．原因の２つ目は，非定型の検索処理では単一の表のグループ化，または複数の表を対象とする副問合せや結合演算を伴うことが多く，そのために表を特定の列の値の順序で並べ換えるソート処理に多大のＣＰＵ時間がかかることである．これら２つの原因によって，データベース管理システム内部でＣＰＵネックと入出力ネックの両方が発生し，オンライン・リアルタイム処理で要求される性能条件を満足させることは極めて困難であった．

2.2.2　解決方針

サーチ処理の基本的な問題は，全ての行をディスクから汎用計算機の主記憶に転送した上で検索条件の判定を行うため，多大のＣＰＵ処理とデータ転送が必要になることである．一方，ソート処理の問題は，対象となる行の数Ｎに対して，通常のソフトウェア・アルゴリズムではオーダ$N\times(\log_2 N)$の時間がかかることと，ソート処理を効率良く実行するためには，対象となるデータを格納するための主記憶を大量に必要とすることである．

これらの問題の解決を図るため，サーチ処理とソート処理を専用ハードウェアで実行するデータベースプロセッサＲＩＮＤＡを開発した．サーチ処理に関しては，ディスクのデータ転送速度で，ディスク中に格納された表のオン・ザ・フライ（on-the-fly）処理，即ちディスクからのデータ読み出しと並行した検索条件の判定処理を行い，条件に合致した行の中の必要な列のみをホスト計算機に転送する．ソート処理に関しては，Ｉ／Ｏチャネルのデータ転送速度でホスト計算機から表を受け取り，ソートした結果をホスト計算機に返送する．以上により，ＲＩＮＤＡを接続したホスト計算機のＣＰＵネックと入出力ネックの両方を同時に解決することを目指す．

2.3　システム構成

2.3.1　設計条件

ＲＩＮＤＡの設計に際しては，ユーザにあまり負担をかけずに稼働中のシステムにも比較的容易に導入できるように考慮した．具体的には，以下の条件に従ってハードウェア及びソフトウェアの設計を行った．

(1) ハードウェアの条件

既存のユーザ資産，即ちホスト計算機と磁気ディスク及び制御装置をそのまま使用できるようにするため，ＲＩＮＤＡは付加型のデータベースマシンとする．さらに，付加型の利点を生かすために以下の条件を追加する．

① ＲＩＮＤＡとホスト計算機との接続は，入出力インタフェースで行う．その理由は，ホスト計算機シリーズ内での共通化，即ち大型から小型まで全ての機種と接続できることと，性能に対する拡張性，即ち稼働中のシステムに対して後から追加導入できることによる．

② ＲＩＮＤＡは，既存のデータベース管理システムと同一の記憶構造を持ったデータベースを処理の対象とする．その理由は，ＲＩＮＤＡは非定型処理を実

行する機能のみを専用ハードウェアとして実現し，既存のデータベース管理システムが残りの機能を全て分担することにより，ソフトウェアの二重開発が防止できることによる．

(2) ソフトウェアの条件

既存のユーザプログラムに対する影響を最小限にするとともに，新たなユーザプログラムの開発も容易にするため，データベース管理システムとユーザプログラムのインタフェースは，標準データベース言語ＳＱＬに準拠したものとする．また，ハードウェアが付加型であるのに対応して，ＲＩＮＤＡの制御機能も独立したサブシステム（ＲＣＰ）により実現し，既存のデータベース管理システムに付加する構成とする．なお，定型処理と非定型処理を１つのデータベースに対して並行して実行できるようにするため，以下の条件を追加する．

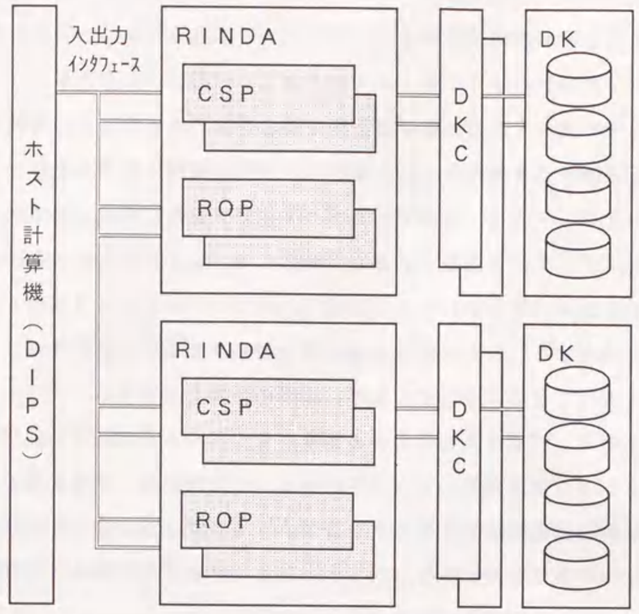
① 従来のインデックスを使用する処理とＲＩＮＤＡを使用する処理を同一のトランザクション管理機構のもとで実行する．その理由は，定型処理を行うトランザクションと非定型処理を行うトランザクションが１つの表に同時にアクセスしたり，１つのトランザクションが１つの表に対する定型処理と非定型処理の両方のアクセスを含む形態を可能にするためである．

② 定型処理と非定型処理の判定を，データベース管理システムが自動的に行うようにする．その理由は，もともと定型処理と非定型処理の間の明確な区分はなく，かつ一般のユーザにとって，１つのＳＱＬ文を実行する際に従来のインデックスを使うべきかＲＩＮＤＡを使うべきかを，ユーザプログラム作成時に判断することは難しいためである．

2.3.2　ハードウェア構成

(1) 構成

ホスト計算機にＲＩＮＤＡを接続したシステム構成例を図2.1に示す．ＲＩＮ



CSP：内容検索プロセッサ，DKC：ディスク制御装置
ROP：関係演算プロセッサ，DK　：ディスク装置

図2.1　ＲＩＮＤＡシステムの構成例
Figure 2.1　Typical RINDA system organization

ＤＡはサーチ処理とソート処理を高速化するための専用ハードウェアとして，内容検索プロセッサ（ＣＳＰ：Content Search Processor）と関係演算プロセッサ（ＲＯＰ：Relational Operation Accelerating Processor ）から構成した．ＣＳＰは，ディスクに格納された表をサーチし，検索条件に合致した行と列を選択し，その結果をホスト計算機に転送する．ＲＯＰは，ホスト計算機から転送されてくる表をソートし，その結果をホスト計算機に送り返す．各ＣＳＰ，ＲＯＰは独立の入出力インタフェースでホスト計算機に接続される．なお，ＣＳＰとＲＯＰの２つのハードウェア構成としたことにより，システム毎に異なるサーチ処理とソート処理の負荷バランスに柔軟に対応することを可能にした．

(2) 動作概要

ＲＩＮＤＡを使用した検索処理の実行例を図2.2に示す．まず，ホスト計算機上のＲＣＰがＣＳＰに対してコマンドを送り，これを受け取ったＣＳＰはディスクに対するサーチ処理を行い，選択した行の必要な列をＣＳＰからホスト計算機の主記憶上のバッファへ転送する（①）．次に，ＲＣＰはＲＯＰに対してコマンドと共にバッファ中の内容を送り，これを受け取ったＲＯＰはソート処理と重複値のカウントを行い，結果を再び主記憶に転送する（②）．最後に，ＲＣＰは処理の最終結果を１行ずつユーザプログラムに引き渡す（③）．

以上の例で明らかなように，ＣＳＰとＲＯＰを使用することにより，ＲＣＰはほとんどＣＰＵを使用することなく，もともとディスクに格納されていたデータの一部のみを転送することによって非定型処理を実現する．従って，従来のデータベース管理システムで問題になっていたＣＰＵネックと入出力ネックの両方に対する大幅な改善を実現している．

(3) ＣＳＰの機能

ＣＳＰは，ディスクに格納された単一の表に対して表2.1(a) に示す機能を実現する．ＣＳＰの特徴は，ＳＱＬで規定されている各種の述語判定機能をほぼそのまま専用ハードウェア化した点にあり，これによってＲＣＰが行わなければな

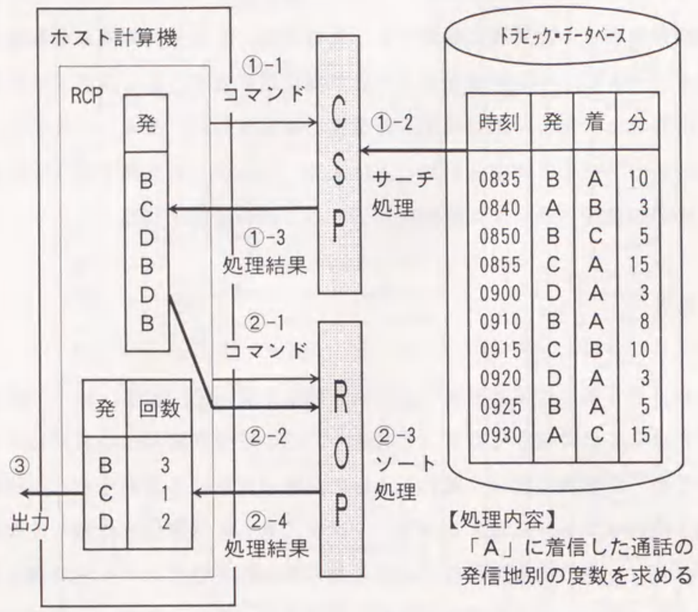


図2.2　ＲＩＮＤＡを用いた検索処理の実行例
Figure 2.2　Typical query execution by RINDA

表2.1　ＲＩＮＤＡハードウェアの主要機能
Table 2.1　Primary RINDA hardware functions

(a) ＣＳＰの機能

機能		説　明
述語判定	比較述語	＜列＞＜比較演算子＞＜定数＞　の条件判定.
	IN述語	＜列＞［NOT］IN ＜定数リスト＞　の条件判定.
	LIKE述語	＜列＞［NOT］LIKE ＜パタン＞　の条件判定.
	NULL述語	＜列＞ IS［NOT］NULL　の条件判定.
検索条件判定		述語の AND/OR による任意の論理式判定.
出力列の抽出		上記検索条件を満たす行からの任意の列の出力.
集合関数		上記検索条件を満たす行数のカウント（COUNT(*)）.

(b) ＲＯＰの機能

機能	説　明
ソート	指定された列の値（ソートキーと呼ぶ）による昇順または降順への行の並べ換え. ソートキーは任意の順序の複数列により構成可能.
ふるい落とし	ソートキーの値による2つの表からの結合可能性のない行の除去. 一方または両方の表からの除去が可能.
重複計数／除去	ソートキーの値が重複する行数のカウント，および2番目以後の行の除去.

らない処理量を削減している．例えば，ＮＵＬＬ値に対する条件の判定もＳＱＬの仕様に準拠しているため，ＲＣＰで特別な処理を行う必要はない．ＣＳＰを用いた処理を実行する際の概略手順は，以下のようになる．

① ＲＣＰがＣＳＰに対するオーダを作成する．オーダ中には，処理の対象とするディスクの機番，ディスク内アドレス等の物理制御情報と，表2.1(a)の機能を記述した論理制御情報が含まれる．

② 作成したオーダを入出力コマンドによってホスト計算機からＣＳＰへ転送する．

③ ＣＳＰは，受け取ったオーダをもとにディスクからデータを連続して読み出し，同時に検索条件の判定と出力列の抽出を実行する．この動作を中断なく行うため，ディスクから読み出すための入力バッファとホスト計算機へ転送するための出力バッファは複数個をサイクリックに使用する．

④ 出力バッファ中に所定の量のデータがたまった段階で，サーチ処理の実行とは非同期に処理結果をホスト計算機へ転送する．

なお，サーチ処理自体は新規に開発した専用ＬＳＩによって並列処理することにより，ほとんどの場合にオン・ザ・フライ処理すなわちディスクからのデータ転送時間内での処理を実現した．また，ＣＳＰの処理結果はそのままＲＯＰの入力となるデータ形式を採用することにより，ＣＳＰとＲＯＰの処理の間で制御ソフトウェアが行単位でデータを加工する必要をなくした．

(4) ＲＯＰの機能

ＲＯＰは，ホスト計算機から転送されてくる1個または2個の表に対して表2.2(b)に示す機能を実現する．このうち，ふるい落とし機能は副問合せおよび結合演算の実行時のホスト計算機の負荷を削減するため，ハッシュ化ビットアレイ[21]により一方または両方の表から対応関係のない行を除去するものである．ＲＯＰの特徴は，ソート処理のための演算回路とメモリをホスト計算機とは独立に持っている点にあり，このためホスト計算機とは独立かつ非同期に処理を実行できる．ＲＯＰを用いた処理を実行する際の概略手順は，以下のようになる．

① ＲＣＰがＲＯＰに対するオーダを作成する．オーダ中には，表2.1(b)の機能を記述した論理制御情報のみが含まれる．

② 作成したオーダを入出力コマンドによってホスト計算機からＲＯＰへ転送する．

③ ＲＯＰは，オーダを受け取ると引き続きホスト計算機から転送されてくる表を入力し，ふるい落とし，ソート，重複値の計数／除去の順にパイプライン処理する．ホスト計算機との間のデータ転送を中断させないため，入出力バッファは複数個を交代で使用する．

④ 表の入力が終了した後，結果データをホスト計算機へ転送する．

なお，ソート処理はＲＯＰ内部の大容量メモリとマルチウェイ・マージソート[34]を行う新規開発の専用ＬＳＩを用いることにより，チャネルのデータ転送時間内でのパイプライン処理を実現した．マルチウェイ・マージソートの特性としてソート対象となる行の数が少ない場合，具体的にはマージ処理が2段階ですむ場合には，マージ処理時間がホスト計算機とのデータ転送時間と完全にオーバラップされ，全体の処理時間はオーダＮ（Ｎは行の数）となる．また，マージ処理が3段階以上になる場合でも，ＲＯＰ内部では行の中のソートキーのみを扱うためにマージ処理時間が行全体を転送する時間と比べて十分小さく，処理時間はオーダＮに近い．

2.3.3 ソフトウェア構成

(1) 構成

ホスト計算機上のデータベース管理システムの構成を図2.3に示す．データベース管理システムは，既存のソフトウェアロジックでＳＱＬ文の処理を行うサブシステム群に，ＲＣＰ，即ちＲＩＮＤＡ対応の機能を持つサブシステムを追加する構成とした．ＲＣＰの主な機能は，ＳＱＬ文のＲＩＮＤＡ用最適化機能と，ＳＱＬ文実行時のＲＩＮＤＡ制御機能である．このような構成とした理由は，以下の2点である．

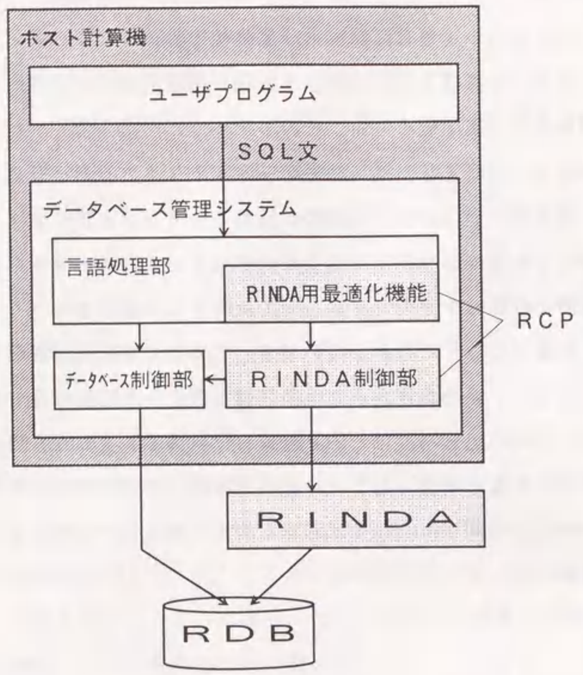


図2.3　データベース管理システムの構成
Figure 2.3　Database management system organization

① 同一のデータベースに対してＲＩＮＤＡを使用したアクセスと使用しないアクセスを同時に行うことを可能とする．

② データベース管理システムにＲＩＮＤＡ対応の機能を追加したことによる悪影響，即ち，ＲＩＮＤＡを使用しない既存ユーザに対する性能劣化の発生を防止する．

(2) 機能概要

ユーザプログラムから発行されたＳＱＬ文は，言語処理部により構文解析，意味解析，最適化が行われる．ここには，ＲＣＰのＲＩＮＤＡ用最適化機能が組み込まれており，第３章で説明する最適化処理方式により，ＲＩＮＤＡを用いた処理方式か従来のソフトウェアだけによる処理方式かのいずれかが選択される．ＲＩＮＤＡが選択されたＳＱＬ文は，ＲＩＮＤＡ制御部により実行される．ＲＩＮＤＡ制御部の主な機能は，ＲＩＮＤＡとの入出力制御，バッファ管理，選択・結合等の基本的な関係演算処理である．従来のソフトウェアが選択されたＳＱＬ文は，データベース制御部により実行される．さらに，データベース制御部は，トランザクション管理，排他制御，資源管理等のデータベース処理として一元的な管理が必要な制御機能も実現する．

(3) 高速化のための技法

ＲＩＮＤＡを使用した検索処理の実行を高速化するため，ＲＩＮＤＡ制御部では以下の工夫を行った．

① １つの表を複数（Ｍ個）のディスクに分割して格納しておき，検索処理の実行時にＭ個のＣＳＰが並列に読み出すことにより，ディスクからのデータ読み出し時間を約Ｍ分の１に短縮する．

② ２つの表に対する結合演算は，一方の表に対するＣＳＰからの出力データ量に応じて結合演算アルゴリズムを動的に選択することにより，以後の処理時間を短縮する．

③ 表全体をサーチ処理の対象とする非定型の検索処理とインデックスを用い

た単一の行を対象とする定型の更新処理の同時走行を可能にするため，ＲＩＮＤＡ側の処理をダーティデータ・リード，即ち，データへのロックをかけずに問合せを実行する機能をオプションとして提供する．

④　ＣＳＰ／ＲＯＰが出力する処理結果は，基本的に主記憶上に格納するが，領域が不足した場合は作業用ディスクに書き出すことにより，ＣＳＰ／ＲＯＰの処理を中断させることなく大量データの処理を可能とする．

2.4　結合処理の実現方式

2.4.1　基本的な考え方

結合処理の代表的手法として，ネステッドループ法[35]，ソートマージ法[35]，およびハッシュジョイン法[36]が知られている．第１表の各行に対して第２表を逐次的に比較して結合するネステッドループ法は，制御は簡単であるが２つの表の行数の積に比例する演算量を必要とすることから，表が小さい場合にのみ利用可能である．ハッシュジョイン法は，２つの表をハッシュ関数を用いて複数のバケットに分割し，対応するバケット間で結合を行うことから，並列処理向きといえる．ソートマージ法は，結合する表をあらかじめソートしておくことで，マージ結合の演算量を線形化できる．ＲＩＮＤＡでは，ソートマージ法の前処理としてフィルタフェーズを設け，フィルタフェーズとソートフェーズを専用ハードウェアで高速に処理し，マージ結合をホスト計算機で行う３フェーズジョイン法を実現した．

次に，専用ハードウェア化の方針を示す．フィルタフェーズは，不要な行を除去する手段としてハッシュ化ビットアレイ法[21]を用いる．まず，第１表の各行から結合のためのキーを作成し，そのキーをハッシュ関数を用いてハッシュし，ビットアレイの対応する位置に設定する．次に，第２表の各行から作成したキーを，第１表で用いたと同一のハッシュ関数でハッシュし，ハッシュ値に対応するビットアレイの位置が設定していなければ，その行は結合の可能性がないとする．第１表に関する操作をビットアレイの設定，第２表に関する操作をビットアレイ

の参照と称し，ハッシュ値の一致・不一致によって，第２表から結合可能性のない行をふるい落とす．従って，フィルタフェーズで用いるハッシュ関数は，種々の属性および長さからなるキーを処理できなければならない．また，ハッシュ値に衝突(Collision) が発生すると，結合可能性のない行が残る．以上示したフィルタフェーズは，結合対象となる２つの表の全ての行に対してハッシュ値を算出し，ビットアレイを設定，参照するフェーズであり，結合に用いられるキーの個数や長さ，キーの属性や分布の片寄りをあらかじめ知ることができないことから，より高度な，即ち計算量の多いハッシュ関数を使用する必要がある．このため，フィルタフェーズは専用ハードウェア化した．

ソートフェーズは，多くのソータの研究例を挙げるまでもなく，計算量が大きい処理であるから専用ハードウェア化する．データベース処理では，アプリケーションによってソートするキーの個数や長さが大きく変動することから，これらの値に柔軟に対処でき，かつ大容量のソータをコンパクトに実現できるマルチウェイマージソータ[34]を実現した．

マージ結合フェーズの入力は，既にフィルタフェーズで結合可能性のない行が大部分除去され，かつソートフェーズで結合対象となる２つの一時表がソートされている．このため，マージ結合の計算処理量は十分に小さくなっていると期待できる．また，マージに基づく行の連結は，ユーザによって指定された出力行の構成に大きく依存する．以上の理由から，マージ結合フェーズはホスト計算機上のデータベース管理システムで実行することにした．

2.4.2　３フェーズジョイン方式

ソートマージ結合法の前処理としてふるい落とし処理を行う３フェーズジョイン法は，ふるい落としを行う表が，結合対象となる２つの表の片方か両方かによって，図2.4に示す２種類の方法が実現できる．

(a) 片ハッシュ・ソートマージ結合法

ＣＳＰで検索した第１の一時表をＲＯＰに入力し，ビットアレイの設定とソートを行い，結果をソート済み一時表として出力する．次に，第２の一時表をＲＯＰに入力し，既に設定されているビットアレイを参照してふるい落としを行った

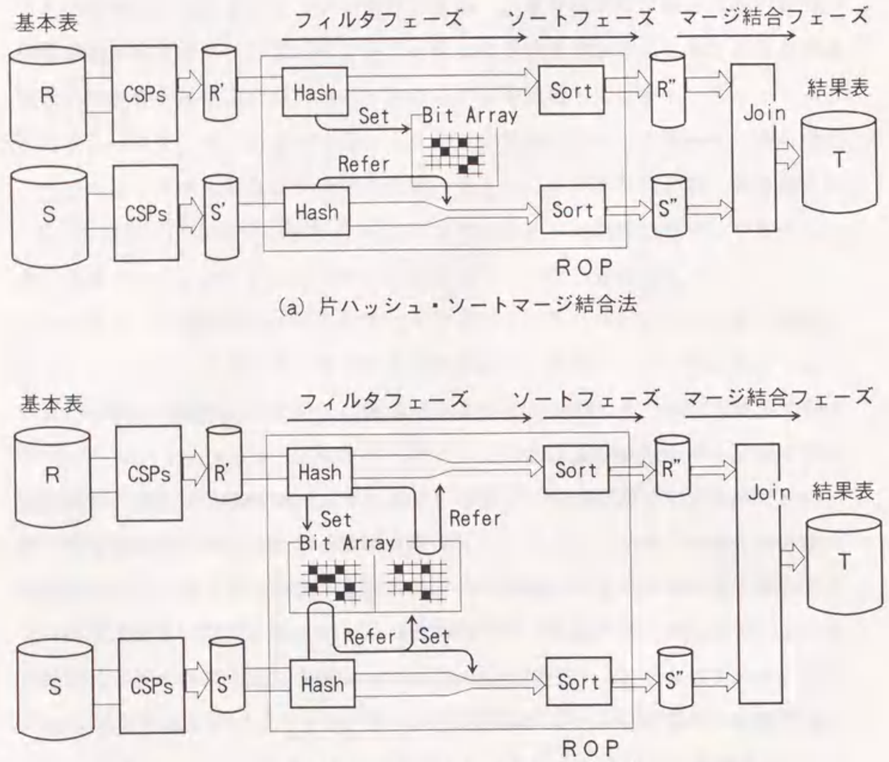


図2.4　ＲＩＮＤＡを用いたソートマージ結合処理法
Figure2.4　Sort-merge join methods using RINDA

後にソートして，結果をソート済み一時表として出力する．得られた２つのソート済み一時表をホスト計算機でマージ結合する．本方法では，第２の一時表だけがふるい落とされる．

(b) 両ハッシュ・ソートマージ結合法

第１の一時表をＲＯＰに入力してビットアレイを設定する．次に，第２の一時表をＲＯＰに入力し，第１の一時表で設定されたビットアレイを参照してふるい落としを行った後に，別のビットアレイの設定とソートを行って，結果をソート済み一時表として出力する．再度，第１の一時表をＲＯＰに入力し，第２の一時表で設定されたビットアレイを用いてふるい落としを行なった後にソートして，結果をソート済み一時表として出力する．最後に２つのソート済み一時表をホスト計算機でマージ結合する．本方法では，２つのビットアレイを用いて，第１と第２の両方の一時表がふるい落とされる．

ＲＯＰで実現したソータは，搭載しているメモリ容量から，キー長によってソートできる最大キー数が決まる[34]．ふるい落とし処理で残った行数がソートできる最大キー数以下であれば，オーバフローせずに一度にソートできる．このため，結合する表が極めて大規模な表であっても，ふるい落とし後の一時表の大きさ，すなわち，ソートするキー数が制限以下であればオーバフローしない．また，ふるい落としによってＲＯＰから出力される一時表が小さくなれば，一時表の転送に要する時間とマージ結合に要する時間を短縮できる．このように，ふるい落とし処理で結合可能性の無い行をできる限り除去することが，適用領域の拡大と処理の高速化を達成する上で重要となる．

2.4.3　ハッシュ関数の設計

ハッシュ関数は，互いに異なる２個以上のキーのハッシュ値が衝突する場合がある．リレーショナルデータベース処理では，キーを生成する列の属性が整数であったり文字列であったり，時には異なる属性を持つ複数の列からキーを生成する場合もある．また，キー値の分布に片寄りが生じる場合もあり，単純なハッシ

ュ関数では衝突率が高くなると考えられる．ハッシュ値が衝突すると，十分に結合可能性の無い行を除去できなくなるため，よりハッシュ値の片寄りが少ないハッシュ関数が必要となる．

ハッシュ関数は，キーをハッシュ表のアドレス空間に一様（ランダム）に割り当てるものが望ましい．文献[37][38]は，短い固定長キーを前提として，過去に提案されているハッシュ値の衝突を比較している．代表的なハッシュ関数は，除算法，乗算法，折り畳み法，平方採中法，桁解析法，基数変換法などであり，キーの集合が未知な場合は，除算法が比較的衝突率が低い．ふるい落とし処理では，キー長が長い場合や可変長の場合があり，そのまま除算法を適用することはできない．以下にふるい落とし処理で用いるハッシュ関数の特徴を示す．

(a) ハッシュ表は，ハッシュ値に対応するキーの有無を表示する１ビットのセルの集まりである．このため，以下に示すロードファクタを小さくできる．

$$\text{ロードファクタ} = \frac{\text{異なりキー値の総数}}{\text{ハッシュ表のセル数}}$$

(b) 衝突の軽減は重要であるが，衝突が起きた場合でもあふれ（オーバフロー）等の処理は不要である．

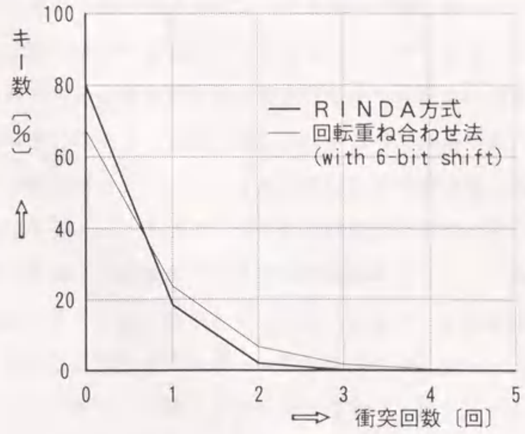
(c) 種々のデータ属性に対して，同一のハッシュ関数が適用できる必要がある．

(d) 比較的長い可変長のキーを扱える必要がある．

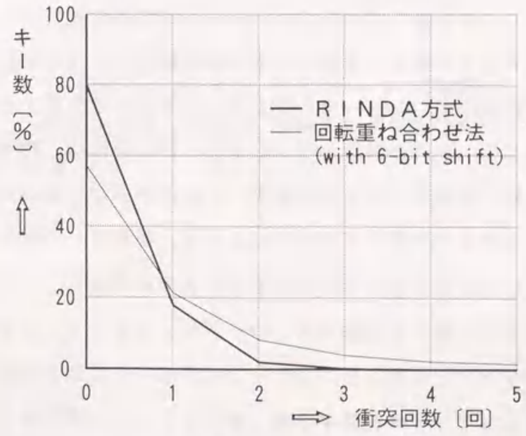
比較的長い可変長のキーを扱う方法に，キーを固定長の複数の切片に分割して，それらを排他的論理和を用いて重ね合わせる排他的論理和法がある．しかし，文字列，特に日本語文字列の文字コードは，あるビットに１が出現する確率に偏りがあるため，単純に１バイトあるいは２バイトの切片を用いて重ね合わせると，結果のハッシュ値に偏りが生じる．

ハッシュ値の偏りを軽減する手法として，適当なビット数をずらして重ね合わせたり，ビット順を入れ換えて折り畳む方法等が知られている．ＲＯＰでは，分割した切片に乗算を施した後に，得られた結果を一定ビット回転して重ね合わせる乗算重ね合わせ法（以下ではＲＩＮＤＡ方式と呼ぶ）をハードウェア化した．乗算は文字コードの偏りを分散するのに，回転重ね合わせは可変長キーを同一のハードウェアで扱うのに有効な手法といえる．

ＲＩＮＤＡ方式の有効性を検証するために実際のデータを用いて評価した結果



(a) ランダム数字列（16バイト固定長）



(b) 英単語（可変長）

図2.5　ＲＩＮＤＡ方式におけるハッシング効果

Figure 2.5　Hashing effect on RINDA method (235k keys/1M-bit Cells)

を図2.5に示す．図の横軸は，同一のハッシュ値に衝突した回数を，縦軸はキーの割合を示している．従来法である回転重ね合わせ法は，回転数を0から7まで変えて評価を行い，最も衝突の発生が少なかった回転数6の場合を示している．図2.5(a)は長さ16バイト・固定長のランダム数字列を，図2.5(b)は英語辞書の見出し語を用いた場合であり可変長の英単語である．いずれの場合であってもＲＩＮＤＡ方式が優れているといえるが，特に，キーの分布に偏りがある英単語の例では，回転重ね合わせ法の衝突無しの比率が60％未満に対して，ＲＩＮＤＡ方式は80％の単語が衝突無しである．また，キーの衝突回数で比較すると，ＲＩＮＤＡ方式は，ランダム数字列と英単語でほぼ同一の結果が得られているのに対して，回転重ね合わせ法は，文字コードの分布やキー長に依存しており，英単語の例では，ハッシュ値が衝突するキー数が5を越える割合も10％程度と高い．ＲＩＮＤＡ方式では，いずれの評価でも3個以上のキーのハッシュ値はほとんど衝突しない．

次に，ハッシュ表の大きさ（セル数）が衝突の度合いに与える影響を示す．ビットアレイの大きさを変えて所定の個数のキーをハッシュすることによって，前述のロードファクタを変えて評価した．ハッシュ関数にＲＩＮＤＡ方式を用い，ロードファクタを1から1／8まで変えた時の衝突の度合いを評価した結果を図2.6に示す．評価に用いたキーは，図2.5(b)と同一である．ロードファクタが1の時，すなわち，キーの個数とセルの個数が等しい時は，約4割のキーが衝突なし，残りの約4割のキーが1回の衝突，2割のキーが2回以上の衝突を起こしている．ロードファクタが1／4以下になると，衝突なしの割合が8割を超え，2回以上衝突する確率はほぼゼロとなる．以上の結果より，ハッシュ化ビットアレイを用いたふるい落とし処理では，ロードファクタを1／4よりも小さくする，すなわち，ビットアレイの大きさをハッシュするキーの個数の4倍以上の大きさにすることによってキーの衝突率を低く抑えることが可能になることがわかった．従って，ハッシュするキーの個数に応じてビットアレイのサイズを決めることが，ふるい落としの効率向上とビットアレイのためのメモリ量削減を両立する上で重要になる．

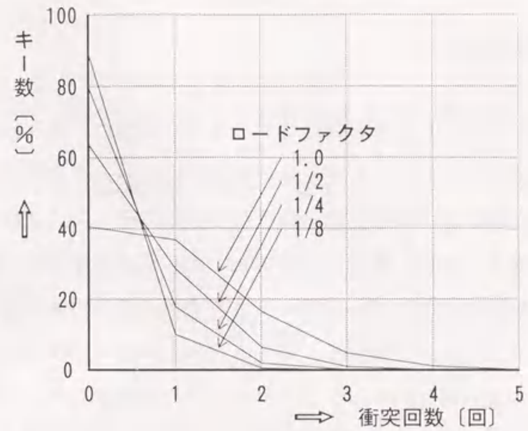


図2.6　ロードファクタ対ハッシング効果
Figure 2.6　Hashing effect vs load factor

2.4.4　結合演算の実行

フィルタフェーズとソートフェーズを実現するＲＯＰの内部構成を図2.7に示す．回路ブロックの構成については文献[39]に詳しいので，ここでは結合処理を実現する上でのＲＯＰの特徴を示す．

(1) 動的最適化方式

結合対象となる表をＣＳＰを用いてアクセスする際に，読み出した一時表に含まれる行数をカウントし，この値を基に最適な結合方式を選択する動的最適化を実現している．第１および第２の一時表が小さく，ディスク上にワーク域を作成せずに結合できるときは，ＲＯＰは使用せずにＣＳＰで得られた検索結果の一時表をホスト計算機上でネステッドループ結合する．それ以外の場合は，ＲＯＰを用いた３フェーズジョインを行う．第１の一時表のサイズが第２の一時表のサイズより十分に小さい時は片ハッシュ・ソートマージ結合法を，上記以外，即ち，第１および第２の一時表のサイズが大きくかつ均衡している場合，および，ソータ容量を超えてオーバフローが予想される場合は，両ハッシュ・ソートマージ結合法を使用する．

(2) 内部キー方式

一般に，表は複数の異なる属性を持つ列からなり，属性値のタイプには，整数，符号有り／符号無しの十進数や文字列等がある．また，属性値として値が未定のナルが許されている場合もある．これらの多種多様な属性の集まりからなる行を直接ソートする試み[40]もあるが，ハードウェアの複雑さから実現上の限界があった．

ＲＯＰでは，フィルタフェーズにおけるハッシュ関数の入力キー，およびソートフェーズで使用するキーをＲＯＰの内部キーとして抽出する専用ハードウェアを設けている．この内部キーは，キーの先頭位置からの比較で大小関係が判定できる絶対値数で，可変長文字列やナル値は定義長で固定長化している．内部キー

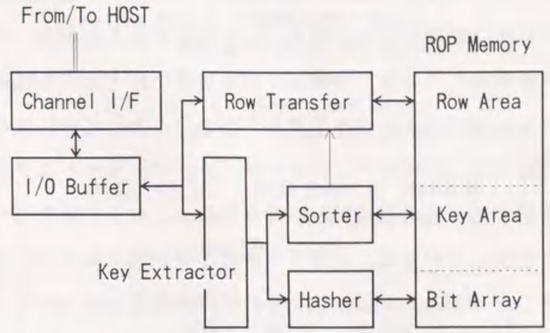


図2.7　ＲＯＰの構成
Figure 2.7　ROP block diagram

を専用ハードウェアで作成することにより，異なる属性を持つフィールドを複数個連結したキーを，高速にふるい落としまたはソートできる．

(3) 作業メモリ方式

ＲＯＰでは，行データから抽出した内部キーを用いてフィルタリングおよびソーティングを行う．処理結果は一時表として返却することから，内部キーとは独立に行データを格納している．内部キーの最後部に行データの格納位置を示すポインタを付与し，ソート後にポインタから行を読み出して一時表を作成する．従ってＲＯＰ内部には，内部キー，行データ，及びフィルタリングで使用するビットアレイを格納するための記憶部が必要となる．これら３種類のデータを独立のメモリ装置に格納したのでは，メモリの使用効率が低くなるばかりかメモリの実装密度も低下し，小型化・低価格化を阻害する要因となる．

ＲＯＰでは，大容量メモリチップを高密度に実装した１つの作業メモリを，行格納域，内部キー格納域，およびビットアレイ格納域に動的に分割して使用する．ビットアレイ域の大きさは，ハッシュ値のロードファクタを一定として衝突率を低く保つために，作業メモリの一定比率の領域を割り当てる．残りの領域を行格納域と内部キー格納域に分割して使用するが，行の長さは可変長で最悪条件では行毎に長さが異なる場合がある．このため，行を格納する毎に作業メモリのオーバフローを検出する回路を設けて，真に作業メモリが満杯になるまで処理できるように設計した．

2.5　方式評価

2.5.1　評価条件

ＲＩＮＤＡによる非定型の検索処理の高速化効果を拡張ウィスコンシン・ベンチマーク[41][42]により評価した．評価に用いた表の基本的な大きさは１万行であり，各行はシステムの制御情報を除いて 208バイトの長さである．評価に用い

表2.2　性能評価で用いたＳＱＬ文
Table 2.2　SQL statements executed

問合せ			ＳＱＬ文
Selection		1%	SELECT * FROM HUNKA WHERE UNIQUE1>=50000 AND UNIQUE1<51000
		10%	SELECT * FROM HUNKA WHERE UNIQUE1>=50000 AND UNIQUE1<60000
Join	AselB		SELECT * FROM HUNKA A, HUNKB B WHERE A.UNIQUE1=B.UNIQUE1 AND A.UNIQUE1<10000
	CselAselB		SELECT * FROM HUNKA A, HUNKB B, TENKA C WHERE C.UNIQUE1=A.UNIQUE1 AND A.UNIQUE1=B.UNIQUE1 AND A.UNIQUE1<10000 AND B.UNIQUE1<10000
MIN		Scalar	SELECT MIN(UNIQUE1) FROM HUNKA
		GROUP-BY	SELECT MIN(TWOTHOUS) FROM HUNKA GROUP BY HUNDRED
COUNT		Scalar	SELECT COUNT(*) FROM HUNKA
		GROUP-BY	SELECT COUNT(*) FROM HUNKA GROUP BY HUNDRED

たシステムは，ホスト計算機が小型汎用機のＤＩＰＳ－Ｖ３０Ｅ[33]，ＣＳＰが２台，ＲＯＰが１台と，容量が1.3GB，データ転送速度が3MB/Sのディスクおよび制御装置各２台により構成した．

処理時間を実測した問合せを表2.2に示す．このうち，Ｑ１～Ｑ６はベンチマーク標準の問合せであり，Ｑ１～Ｑ３によってＣＳＰ単独による性能向上効果，Ｑ４～Ｑ６によってＣＳＰとＲＯＰの両者による性能向上効果を評価した．さらに，ＲＩＮＤＡのハードウェア部分に着目した処理性能を評価するため，検索結果データの転送をせずに行数だけをカウントするＱ７～Ｑ９を評価した．なお，ＲＩＮＤＡの開発目的から問合せの評価はすべてインデックスを使用せずに行った．

2.5.2　評価結果

ＲＩＮＤＡを使用しない場合と使用した場合の性能を比較すると図2.8のようになる．処理時間比は，ＳＱＬ文の実行開始から検索結果をすべてユーザプログラムに返却し終わるまでの経過時間の相対比である．この図から，以下のことがわかる．

①　ホスト計算機のＣＰＵ時間が大幅に短縮される．

②　ホスト計算機のＩ／Ｏ時間も大幅に短縮される．

③　ＲＩＮＤＡによる非定型検索の性能向上効果は，最高 100倍以上に達する．

さらに，公表されている市販のデータベースマシンの性能[42][43]とＲＩＮＤＡの性能比較を表2.3に示す．この表からも，ＲＩＮＤＡは非定型の検索処理が高速であることがわかる．

2.5.3　考察

(1) 性能向上要因

ＲＩＮＤＡにより上記の性能向上が実現できた要因をまとめると以下のようになる．

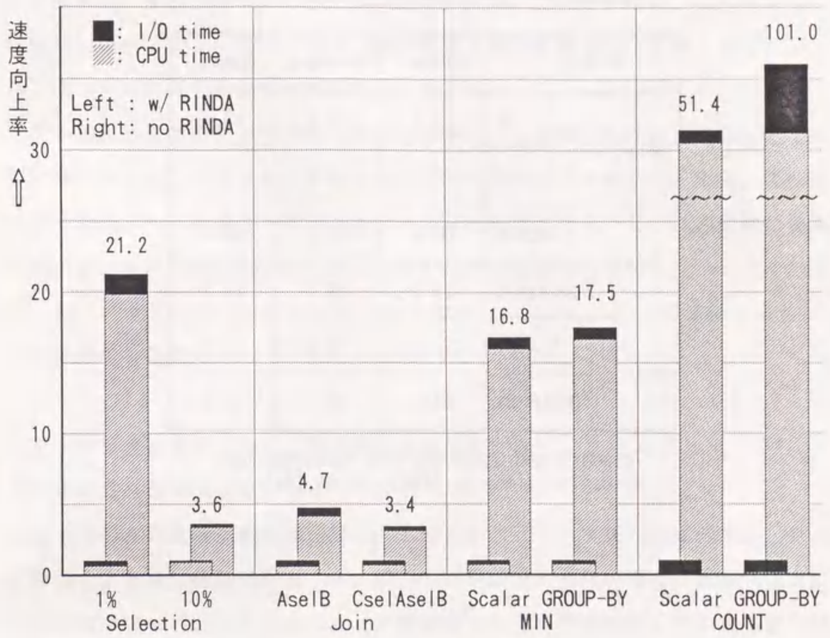


図2.8　ＲＩＮＤＡによる処理速度向上効果
Figure 2.8　Speed-up by RINDA

表2.3　問合せ処理時間の比較
Table 2.3　Query execution times in seconds

問合せ		RINDA	Teradata	Gamma
Selection	1%	5.2	18.6	13.4
	10%	9.0	14.9	12.7
Join AselB		136.8	235.6	35.8
Join CselAselB		128.5	95.7	37.9
MIN Scalar		31.7	18.3	15.5
MIN GROUP-BY		47.3	27.1	19.4

- Teradata and Gamma are the adjusted query execution times by DeWitt et al.[42]

(a) ホスト計算機のＣＰＵ時間短縮

非定型の検索処理を従来のソフトウェアで実行した場合，サーチ処理およびソート処理を表の中の全ての行に対して行うため，膨大なＣＰＵ時間がかかっていた．ＲＩＮＤＡの利用によりこれらの処理の大半をオフロードでき，ＣＰＵ時間を大幅に短縮した．なお，データベース管理システムとユーザプログラムの間で処理結果を行単位で返却する処理はオフロードされていないため，処理結果の行数が多い場合（Ｑ２，Ｑ５，Ｑ６）は性能向上効果が相対的に小さくなる．

(b) ホスト計算機の入出力時間短縮

１つの表を複数のディスクに分割して格納し，検索実行時に複数のＣＳＰを並列に動作させることにより，ディスクからの入力時間を短縮した．また，１シリンダ内の全データを途中でのディスクヘッドの機械的な動作なしに連続的に読み出すことにより，ディスクからの入力そのものの効率も向上した．

(2) 効果の有効範囲

以上の評価結果をもとに，別の観点から分析する．

(a) 表の大きさについて

ＲＩＮＤＡによる検索処理時間は，表の大きさ（データ量）に比例する．汎用計算機による検索処理時間も，インデックスを使用できない場合は表の大きさに比例する．従って，ＲＩＮＤＡによる性能向上効果（相対比）は表の大きさにはほとんど依存しない．

(b) 問合せの種類について

ＲＩＮＤＡはＳＱＬのすべての機能を分担するわけではない．例えば，ＣＯＵＮＴ関数はＲＩＮＤＡで処理するが，ＳＵＭ関数はホスト計算機で処理する．また，処理結果をユーザプログラムに返却する処理もＲＩＮＤＡは関与しない．このため，問合せの種類および処理結果の大きさによって性能向上効果はかなり変化する．

(c) コストパフォーマンス

ＲＩＮＤＡの導入によりシステムのハードウェアコストは数割のオーダで増加するが，ＲＩＮＤＡによる性能向上効果は数倍から数十倍のオーダになる．従っ

て，非定型の検索処理の比重が高いシステムについては１桁近いコストパフォーマンス向上が得られる．

(3) 方式上の限界

ＲＩＮＤＡには，２つの方式上の限界がある．１つは，更新処理との同時実行である．ＲＩＮＤＡによる検索時には，原則としてＣＳＰがアクセスする表全体を参照ロックする必要があり，ロック処理のオーバヘッドと更新処理のブロッキングを引き起こす．2.3節で述べたダーティデータ・リード機能は１つの解決策である．２つ目の限界は，問合せ自体の同時実行である．ＲＣＰは複数問合せの並行処理が可能であるが，各ＣＳＰ及びＲＯＰは１度に１つの要求しか受け付けない．従って，同一のＣＳＰまたはＲＯＰを必要とする複数の検索処理の同時実行は，処理時間の増加を引き起こす．

2.6 まとめ

本章では，性能上の物理的なボトルネックを解消するための第１のアプローチとして，データベースプロセッサＲＩＮＤＡのアーキテクチャと実現方式を述べた．ＲＩＮＤＡの目的は，汎用計算機上のデータベース管理システムにとって重い負担となる非定型の検索処理を高速化することにある．このため，サーチ処理とソート処理を専用ハードウェアにより高速に実行する機構を実現した．ＲＩＮＤＡによる性能向上効果は極めて大きく，従来のソフトウェアと比較して検索処理を最高 100倍以上に高速化できた．

ＲＩＮＤＡの適用領域は，リレーショナルデータベースでの非定型の検索処理が必要となる業務分野である．例えば情報検索の分野では，文章データに対してキーワードを抽出したりインデックスを作成したりすることなく，そのままデータベースに格納するだけでＲＩＮＤＡを用いた任意の語句をキーとする検索が可能になった．また統計処理の分野では，大量かつ頻繁に追加されるトラヒック情報をＲＩＮＤＡによって実時間で分類・集計することが可能になった．

3 専用プロセッサにおける最適化処理方式

3.1 まえがき

本章では，前章で述べたデータベース専用プロセッサ方式の適用性を向上させるために，データベース管理システム（ＤＢＭＳ）におけるデータへの最適なアクセスパスの選択方式（最適化処理方式）について論じる．

先に述べたように，ＲＩＮＤＡはデータベース処理全体のうち非定型処理のみサポートしており，定型処理は具備していない．そのため，汎用計算機上のデータベース管理システムがＲＩＮＤＡの制御を行うアーキテクチャを採用している．即ち，従来のＤＢＭＳにＲＩＮＤＡ制御プログラムを付加し，ソフトウェアによる処理（ソフト処理）とＲＩＮＤＡによる処理（ＲＩＮＤＡ処理）を統合したＤＢＭＳの開発を行った．その結果，同一のデータベースに対して，①各利用者が両処理を混在させた実行，②両処理が混在した複数利用者の同時実行，の両方を実現し，利用者はＲＩＮＤＡ処理による非定型処理とソフト処理による定型処理を共通インタフェースで利用可能となった．

両処理を統合したＤＢＭＳでは，利用者はＲＩＮＤＡの使用有無を意識せずにＳＱＬ[2][3]文を記述できることが望ましい．先に述べたように，ＲＩＮＤＡは機能上，検索処理のみをサポートしている．また，検索処理であってもインデックス（索引）が有効に利用できる場合，例えばユニークなキーを用いた１行検索では，ソフト処理の方が高速になる．従って，利用者にＲＩＮＤＡの使用有無を意識させず，かつ高速のデータベース処理を提供するためには，機能上，性能上から両処理の最適なアクセスパス選択が重要となる．特に，性能上のアクセスパス選択は全く異なる２つの処理コストを比較しなければならない難しさがある．従来，ソフト処理の最適化方式は数多く提案されているが[35][44]，ハードウェアとソフトウェアの処理を統合したＤＢＭＳの最適化方式の研究例はない．

本章では，ＲＩＮＤＡ処理とソフト処理が統合されたＤＢＭＳの最適なアクセ

スパス決定方法について述べる．以下，3.2節ではアクセスパス決定方式の概要を述べ，3.3節では評価モデルを示す．3.4節では，評価モデルに基づくアクセスパスの判定基準を導出する．

3.2　最適化処理の構成

(1) データベース管理システムの構成

ＲＩＮＤＡ処理とソフト処理は，処理方式が根本的に異なる．例えば，ソフト処理が1行単位で処理するのに対し，ＲＩＮＤＡ処理は複数行を一括処理し，中間結果を一時表の形で引き継ぎながら処理する．従ってＤＢＭＳ構成は，基本的にＲＩＮＤＡ処理とソフト処理は独立とし，データベース処理として集中制御することに必要な機能の一部を共通化する構成とした（図3.1）．共通化した機能の概要を以下に示す．

①　言語解析機能

利用者インタフェースを同一にするため，ＳＱＬ文の解析を共通に行う．

②　共通最適化機能

以下で述べるように，最適化は共通最適化と個別最適化の2段階で行う方式とした．共通最適化では，機能及び性能上のアクセスパス判定を共通に行う．

③　実行制御機能

両処理の実行時の制御を共通に行う．

④　トランザクション管理機能

トランザクション内で，ＲＩＮＤＡ処理のＳＱＬ文とソフト処理のＳＱＬ文が混在して使用された時の一貫性を保証するため，ＳＱＬ文の実行順序，カーソル等を共通管理する．

⑤　データベース資源管理機能

同一データベースに対して，ＲＩＮＤＡの検索処理とソフトウェアによる検索・更新処理が，複数利用者から同時に行われた時のトランザクション間の一貫性を保証するため，排他制御，救済制御等を共通に行う．

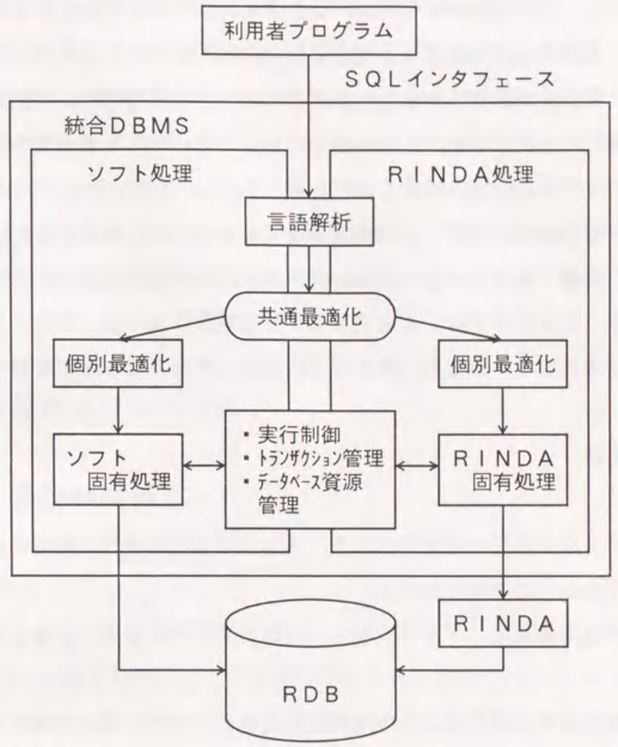


図3.1　ＤＢＭＳのソフトウェア構成
Figure 3.1　DBMS software organization

(2) 最適化の概要

ＲＩＮＤＡ処理とソフト処理の根本的な相違により，各々独自の最適化が必要となること，利用者が両者を共通インタフェースで利用可能とする必要があることより，最適化は共通最適化と個別最適化の２段階で行う方式とした．共通最適化では，機能及び性能上の観点からアクセスパス選択を行い，個別最適化では，共通最適化で選択されたアクセスパスについて更に詳細な最適化を行う．本章で対象とするのは，共通最適化方式である．

共通最適化は機能判定，性能判定の順で実行される．機能判定はＳＱＬ文の種類より，定義・更新用ＳＱＬ文はソフト処理とし，検索用ＳＱＬ文を性能判定に引き継ぐ．性能判定では，3.4節で示す判定基準に基づき，性能上最適なアクセスパスを決定する．性能判定基準導出の前提条件及び基本思想を以下に示す．

＜前提条件＞

① ＤＢＭＳのアーキテクチャ上，ＳＱＬ文はＲＩＮＤＡ処理またはソフト処理のいずれか一方で実行される．

② 共通最適化は，プリプロセス（ＳＱＬ文を中間言語に変換する処理）時に行う．

③ 性能判定は，ＳＱＬ文の実行結果を全て利用者へ返却するまでの処理コストで評価する．

④ 入出力回数とＣＰＵ処理量は比例すると仮定する．

＜基本思想＞

① 原則はＲＩＮＤＡ処理とし，明らかに処理コストが小さいと判定できる場合のみをソフト処理とする．その理由は，非定型の検索処理に対してはＲＩＮＤＡは圧倒的な速さを持ち，処理の大部分を専用ハードウェアで行えば汎用計算機側のＣＰＵ負荷も大幅に削減できると考えられるためである．

② 処理コストとして，入出力回数を評価する．その理由は，より厳密な最適

化のためにはＣＰＵ処理量の考慮も必要であるが，本最適化では明らかにソフト処理のコストが小さいケースを判定するため，前提条件④より入出力回数のみの評価で十分となるためである．

③ 索引の入出力回数は処理コストから除外する．その理由は，一般に索引は主記憶上に存在することが多いためである．

④ 共通最適化は索引定義情報，ソフト処理アクセス法，及びＳＱＬ文から得られる情報のみを用いて行う．その理由は，より正確な最適化のためには表の行数，条件に合致する行の比率等が必要であるが，これらの情報をプリプロセス時に正確に把握することは難しい．一方，索引定義情報，ソフト処理アクセス法，ＳＱＬ文の構文情報は比較的容易に取得できるためである．

3.3 最適化処理のモデル

3.3.1 単純検索処理

(1) 格納モデル

列Ａ，Ｂを含む表Ｐを仮定する．表Ｐの行数をＴp，表Ｐをファイル上に連続的に格納した時のブロック数をＢpとする．列Ａ，Ｂの値は相互に独立かつランダムの一様分布とする．また索引の種類は，クラスタード（索引のキー順に行を連続的に格納）と，非クラスタード（索引のキー順とは無関係に行を格納）を考える．

(2) ステートメントモデル

以下のＳＱＬ文の入出力評価式を考える．

select * from P where φa and φb

φx ：比較述語による列ｘの探索条件

但し，φa，φbのヒット率（条件を満たす行数の比率）をそれぞれμa，μbとし，

探索条件全体のヒット率を「$\mu p = \mu a \cdot \mu b$」とする．

(3) 入出力評価式

(a) ソフト処理

全ブロックアクセスと索引を使用した部分ブロックアクセスの2種類がある．

① 全ブロックアクセス（図3.2(a)）

表Pが格納されている全ブロック（Bp個）に入出力が必要となる．

② 索引アクセス（図3.2(b)）

列Aに付与された索引経由のアクセスを仮定する．探索条件のϕaを索引上で判定することによりアクセス範囲が絞り込める．入出力回数は，クラスタード索引経由では$\mu a \cdot Bp$，非クラスタード索引経由では$\mu a \cdot Tp$となる．

(b) ＲＩＮＤＡ処理（図3.2(c)）

ＲＩＮＤＡは全ブロックを専用ハードウェアで検索し，検索条件に合致した複数行をいったん一時表に蓄える．ＣＳＰによる複数ディスクに分割された表の並列検索と，ＣＳＰ，ＲＯＰの高速ブロックアクセスにより，実効的に入出力回数が削減されたのと同じ効果がある．並列検索の並列度をmp，高速ブロックアクセス効果をx倍とすると，表Pに対する実効的な入出力回数はソフト処理と比べて1／mp／x倍となる．また，$\mu p \cdot Tp$個の行からなる検索結果を格納する一時表サイズは$\mu p \cdot Bp$となる．一時表のライト／リードを考えると，入出力回数は$2\,\mu p \cdot Bp$となる．

(c) 以上の結果，各アクセス方式の入出力回数は次式で評価できる．

$Ep(S,F) = Bp$

$Ep(S,Ca) = \mu aBp$

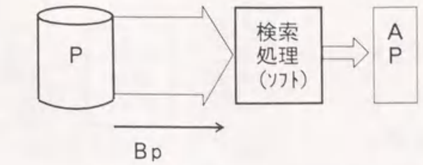
$Ep(S,Na) = \mu aTp$

$Ep(R) = Bp/mp/x + 2\,\mu pBp$

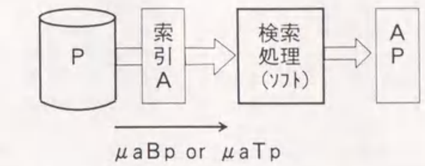
但し，Ep(S,x)：表Pをソフト処理，Ep(R)：表PをＲＩＮＤＡ処理，

x：F（全ブロックアクセス），Cy（クラスタード索引アクセス），
Ny（非クラスタード索引アクセス）

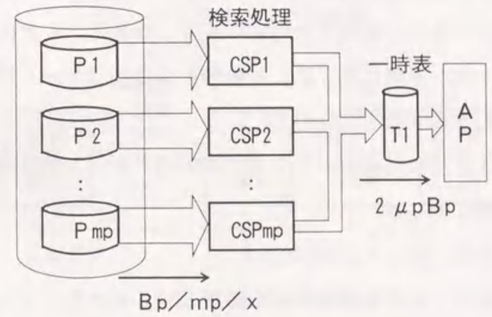
y：索引が付与されている列名



(a) 全ブロックアクセス（ソフト処理）



(b) 索引アクセス（ソフト処理）



(c) ＲＩＮＤＡ処理

図3.2　単純検索処理の概要

Figure 3.2　Access methods for simple queries

3.3.2　ソート／グループ化処理

(1) 格納モデル

単純検索処理と同じとする.

(2) ステートメントモデル

select * from P where φa and φb order by（または group by） B

(3) 入出力評価式

(a) ソフト処理

ORDER BY（ＯＢ）句，またはGROUP BY（ＧＢ）句で指定された列に付与された索引経由で行を昇順に取り出して処理する方法（非ソート方式）と，探索条件を満足する行を取り出し，結果を一時表に蓄え，ＯＢ句，またはＧＢ句に指定されたキーでソート後，昇順に取り出して処理する方法（ソート方式）がある.

①　非ソート方式（図3.3(a)）

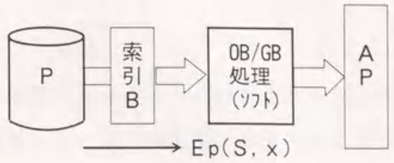
列Ｂに付与された索引経由のアクセスを仮定する．入出力回数は単純検索の索引アクセスと同じとなる.

②　ソート方式（図3.3(b)）

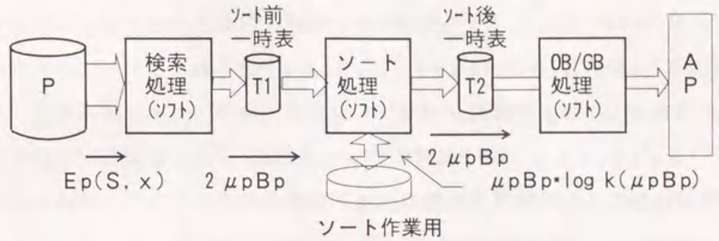
表Ｐの検索部は，単純検索のソフト処理と同じである．ソート前一時表とソート後一時表の作成と読み出し，及びソート作業用の入出力が必要となる．一時表サイズは$\mu p \cdot Bp$となり，２つの一時表の入出力回数は$4\ \mu p \cdot Bp$となる．また，一般的にＢ個のブロック中の行ソートに必要な作業用入出力回数は$B \log_k B$（ｋはマージウェイ数）となる.

(b) ＲＩＮＤＡ処理（図3.3(c)）

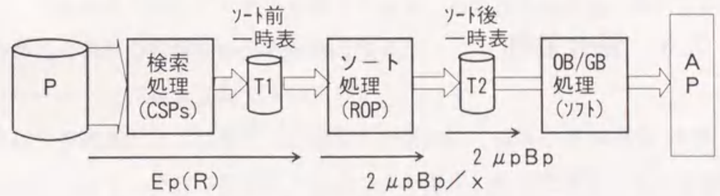
ソフト処理のソート方式と同じであるが，ＣＳＰ，ＲＯＰにより実効上の入出力削減効果がある．表検索からソート前一時表読み出しまでの入出力回数は，単純検索のＲＩＮＤＡ処理と同じである．ＲＯＰではソートをメモリ上で行うため



(a) 非ソート方式（ソフト処理）



(b) ソート方式（ソフト処理）



(c) ＲＩＮＤＡ処理

図3.3　ＯＢ／ＧＢ処理の概要

Figure 3.3　Access methods for OB or GB queries

作業用入出力は無いが，一時表のブロック転送コストが必要となる．ＲＯＰ入出力は，高速ブロックアクセスをしていることより，一時表の実効上の入出力回数は2 μp・Bp／xとなる．またに，ソート後一時表に2 μp・Bpの入出力回数も必要となる．

(c) 評価式

$Sp^*(S,Cb)=Ep(S,Cb)$

$Sp^*(S,Nb)=Ep(S,Nb)$

$Sp(S,F)=Ep(S,F)+4\mu pBp+Sort$

$Sp(S,Ca)=Ep(S,Ca)+4\mu pBp+Sort$

$Sp(S,Na)=Ep(S,Na)+4\mu pBp+Sort$

$Sp(R)=Ep(R)+2\mu pBp+2\mu pBp/x$

但し，$Sort=\mu pBp\cdot\log_x(\mu pBp)$

$Sp^*(S,x)$ ：ソフト処理の非ソート方式

$Sp(S,x)$ ：ソフト処理のソート方式

$Sp(R)$ ：ＲＩＮＤＡ処理

3.3.3 結合処理

(1) 格納モデル

列Ａ，Ｂを含む表Ｐと，列Ｃ，Ｄを含む表Ｑを仮定する．表Ｐ，Ｑの行数をＴp，Ｔq，格納ブロック数をＢp，Ｂqとする．また，索引は列Ａ，Ｂ，Ｃ，Ｄ各々にクラスタード／非クラスタードが付与された状態を考える．

(2) ステートメントモデル

select * from P,Q where P.b =Q.c and ϕa and ϕb and ϕc and ϕd

列ｘの探索条件ϕｘのヒット率をμｘとし，表Ｐ，Ｑのヒット率をそれぞれ$\mu p=\mu a\cdot\mu b$，$\mu q=\mu c\cdot\mu d$とする．また，結合条件によるヒット率μjを仮

定する（但し，$\mu j\leqq\min(\mu b,\mu c)$）．

(3) 入出力評価式

(a) ソフト処理

ソフト処理には，マージ結合とネステッドループ結合方式の２種類があるが，両者はほとんど同じ考え方で評価できるため，ここではマージ結合方式についてのみ述べる．マージ結合は，両表の結合キーを各々昇順に取り出し，マージしながら結合を行う．結合キーを昇順に取り出す方法として，

・結合キーに付与された索引を利用して取り出す．

・結合キーでソートし取り出す．

がある．前者を索引マージ結合，後者をソートマージ結合と呼ぶことにする．

① 索引マージ結合（図3.4(a)）

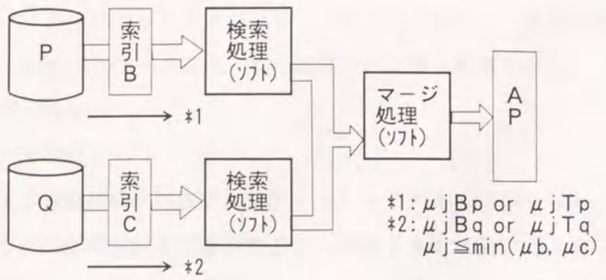
結合キー列Ｂ，Ｃに付与された索引を使用する．まず両索引上で結合可能な行の組を求め，次に索引経由で両表へアクセスし行の結合を行う．従ってこの方式では，結合を考慮したヒット率μjが入出力回数の支配項となる．各表の入出力回数は，単純検索の索引アクセスと同じである．入出力評価式は，両表の結合列に付与された索引の種類の組み合わせとなる．

② ソートマージ結合（図3.4(b)）

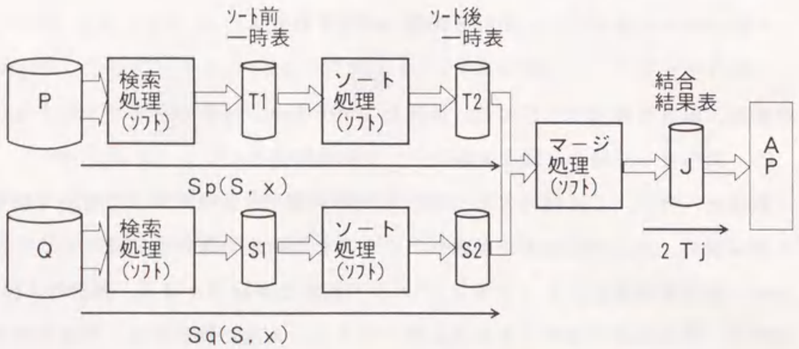
結合以外の列Ａ，Ｄに付与された索引を使用する．各表の検索からソート後一時表読み出しまではＯＢ／ＧＢモデルのソート方式と同じである．結合結果表（サイズをＴjとする）のライト／リードを考えると，２Ｔjの入出力回数が必要となる．入出力評価式は，両表のアクセス方法の組み合わせとなる．

(b) ＲＩＮＤＡ処理（図3.4(c)）

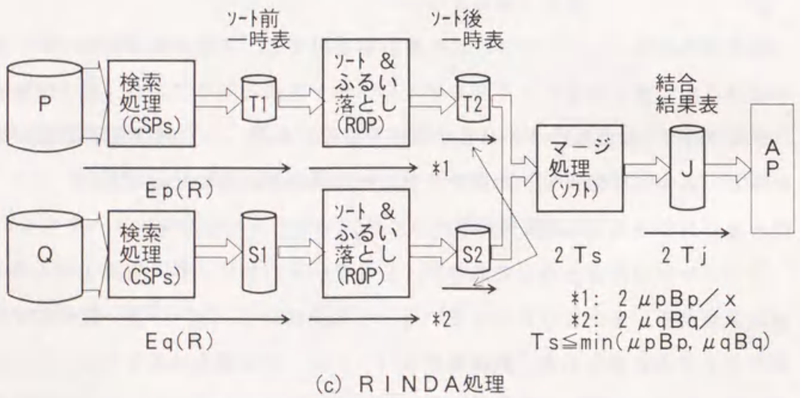
ソートマージ結合と同じであるが，ＣＳＰ，ＲＯＰにより，実効上の入出力削減効果がある．またＲＯＰのふるい落とし機能により，ソート後一時表のサイズが小さくなる効果もある．表検索からソート前一時表読み込みまではＲＩＮＤＡ処理の単純検索と同じである．入出力評価式は，ふるい落としによる入出力削減効果も考慮する．



(a) 索引マージ結合（ソフト処理）



(b) ソートマージ結合（ソフト処理）



(c) ＲＩＮＤＡ処理

図3.4　結合処理の概要

Figure 3.4　Access methods for join queries

(c) 評価式

$$Jim=\left\{\begin{matrix}\mu j\ Bp\\ \mu j\ Tp\end{matrix}\right\}+\left\{\begin{matrix}\mu j\ Bq\\ \mu j\ Tq\end{matrix}\right\}$$

$$Jsm=\left\{\begin{matrix}Sp(S,F)\\ Sp(S,Ca)\\ Sp(S,Na)\end{matrix}\right\}+\left\{\begin{matrix}Sq(S,F)\\ Sq(S,Cd)\\ Sq(S,Nd)\end{matrix}\right\}+2\,Tj$$

$$Jr=Ep(R)+Eq(R)+2\,\mu pBp/x+2\,\mu qBq/x+2\,Ts+2\,Tj$$

但し，$\mu j\leqq\min(\mu b,\mu c)$

$Ts=2\min(\mu pBp,\mu qBq)$

Jim：ソフト結合処理（索引マージ方式）

Jsm：ソフト結合処理（ソートマージ方式）

Jr ：ＲＩＮＤＡ結合処理

Ts ：ふるい落とし後のソート後一時表サイズの和

Tj ：結合結果一時表サイズ

3.3.4　評価例

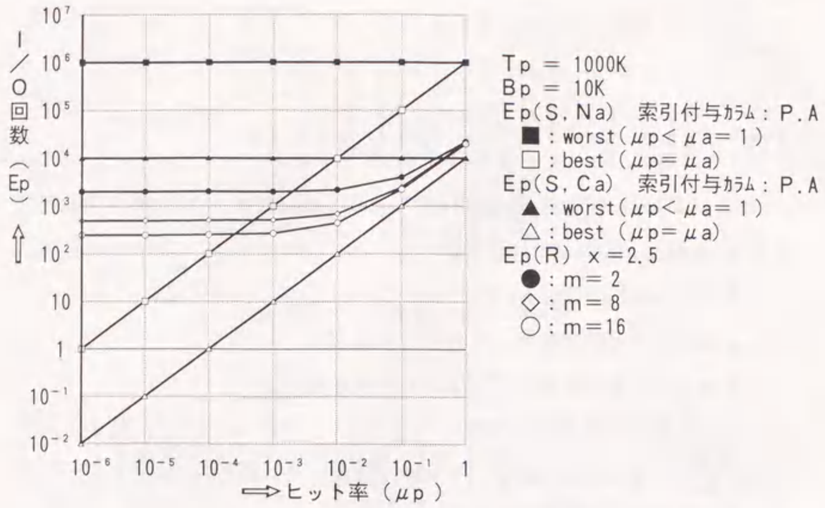
以上で求めた評価式を具体的に適用した例を示す．

(1) 評価条件

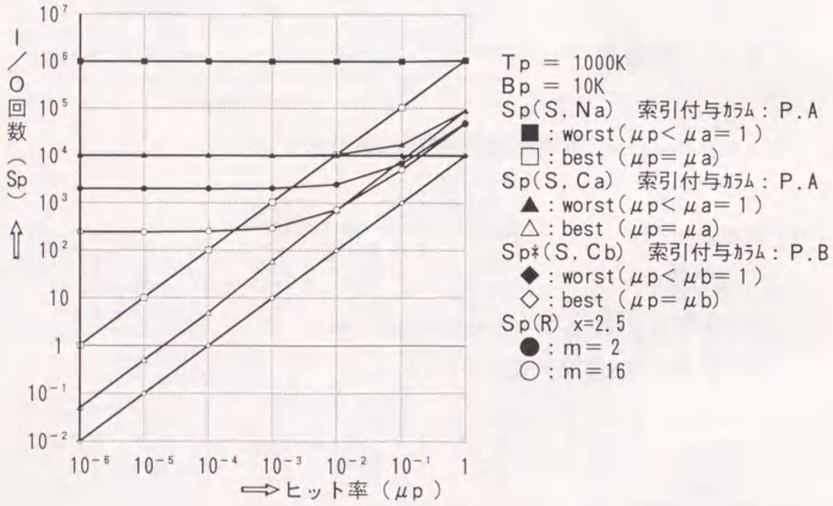
評価モデルで利用したものを使用する．但し，

- Tp ＝１０００k，Bp ＝１０k
- Tq ＝１０００k，Bq ＝１０k
- 列の値は，全てユニーク
- mp ＝mq ＝m
- x＝2.5（ページ単位アクセスとトラック単位アクセスの入出力時間比）
- 結合処理では，簡単のため$\mu p\leqq\mu q$を仮定

(2) 評価例



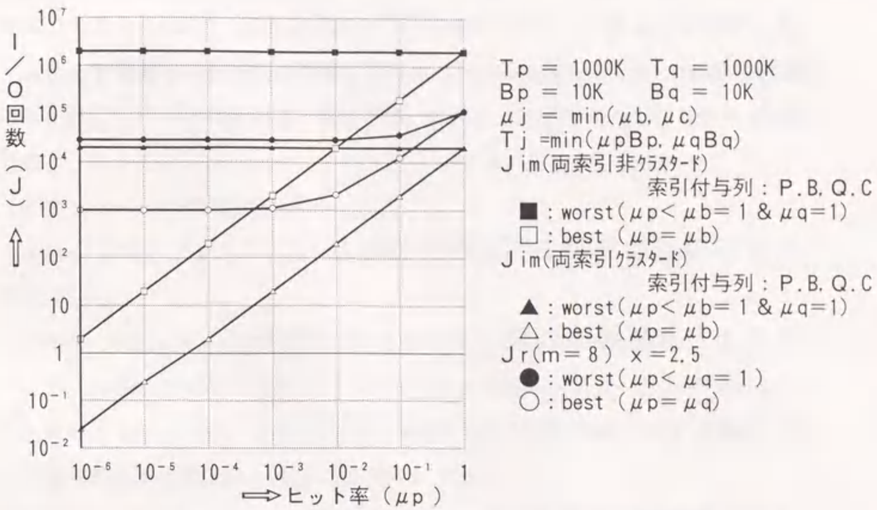
(a) 単純検索処理



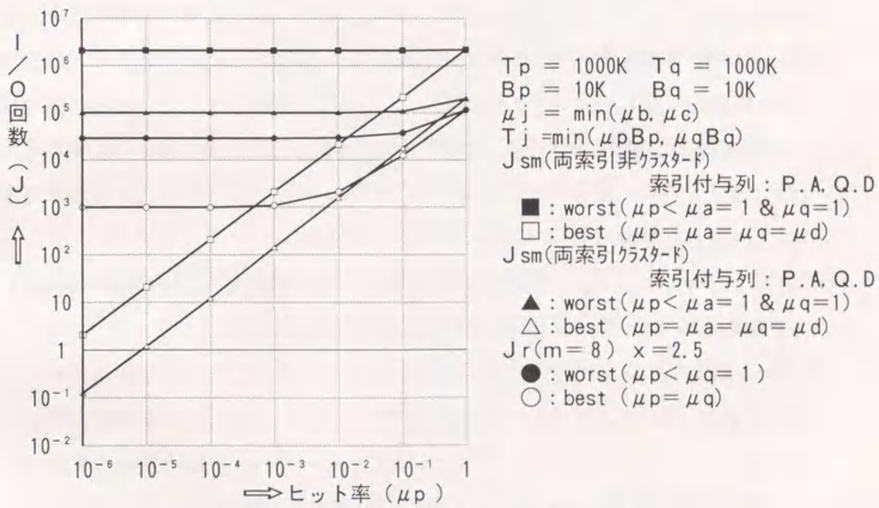
(b) ＧＢ／ＯＢ処理

図3.5　単純検索とＯＢ／ＧＢ処理のＩ／Ｏ回数

Figure 3.5　Number of accesses in simple and OB/GB queries



(a) 索引マージ結合（ソフト処理）とＲＩＮＤＡ処理



(b) ソートマージ結合（ソフト処理）とＲＩＮＤＡ処理

図3.6　結合処理のＩ／Ｏ回数

Figure 3.6　Number of accesses in join queries

ヒット率μpを横軸，ソフト処理，及びＲＩＮＤＡ処理の入出力回数を縦軸にとり，両処理を比較した．単純検索処理の計算結果を図3.5(a)，ＯＢ／ＧＢ処理の計算結果を図3.5(b)に示す．また，結合処理の計算結果は索引マージとソートマージに分け，前者を図3.6(a)，後者を図3.6(b)に示す．

3.4　アクセスパス判定基準の導出

3.4.1　単純検索処理

(1) Ep(S,F)＜Ep(R)となる十分条件

上式を整理すると，$1 < 1/mp/x + 2\mu p$．
μp＝１の時，明らかに，Ep(S,F)＜Ep(R)．

(2) Ep(S,Ca)＜Ep(R)となる十分条件

上式を整理すると，$\mu a < 1/mp/x + 2\mu p$．
(a) μp＝１の時，明らかにEp(S,Ca)＜Ep(R)．
(b) μp＝μaの時，明らかにEp(S,Ca)＜Ep(R)．
(c) $1/mp/x = 10^{-2}$程度を仮定すると，$\mu a < 10^{-2}$の時，$\mu a < 10^{-2} \fallingdotseq 1/mp/x$．従って，Ep(S,Ca)＜Ep(R)が成立する．

(3) Ep(S,Na)＜Ep(R)となる十分条件

上式を整理すると，$\mu aTp/Bp < 1/mp/x + 2\mu p$．
$1/mp/x = 10^{-2}$，$Tp/Bp = 10^{2}$程度を仮定すると，$\mu a < 10^{-4}$の時，$\mu aTp/Bp < 10^{-2} \fallingdotseq 1/mp/x$．従って，Ep(S，Na)＜Ep(R)が成立する．

(4) 判定基準の決定

上記の十分条件を元にＳＱＬ文から抽出可能な条件を求め，それを用いて判定基準を決定した．十分条件は①μp＝１，②μp＝μa，③μp≦μa＜αの３種類である．各々に対応して次の３つのＳＱＬ文抽出可能条件を設定した．

〔条件１：where句に探索条件が無い〕

μp＝１は全行ヒットを意味する．ＳＱＬ文のみから得られる条件として上記を設定した．

〔条件２；where句に探索条件のみが指定されている〕

μp＝μaは探索条件全体のヒット率が索引上の探索条件のヒット率に等しいことを意味する．これは，索引に適合する探索条件のみが指定されていると考え，ＳＱＬ文のみから得られる条件として上記を設定した．

〔条件３；where句に「列名＝定数またはホスト変数」の探索条件がある〕

μp≦μa＜αは索引でのヒット率がある値以下であることを意味する．しかし，十分条件の導出例でも判るようにαは索引の種別で異なる．また，最適化時に正確なヒット率を得るのも困難である．そこで，「キー値の重複度が小さい列に索引が付与される」という経験的事実と，「重複度が小さい列に対する等号条件のヒット率は十分小さい」という仮定より，ＳＱＬ文のみから得られる情報として上記を設定した．

【単純検索処理の判定基準】

ソフト処理のアクセス法とＳＱＬ文抽出可能条件を用い，ソフト処理とする以下の判定基準を決定した．

(a) ソフト処理が全ブロックアクセスの場合
　探索条件が［条件１］を満足．
(b) ソフト処理がクラスタード索引アクセスの場合
　索引に適用可能な探索条件が［条件１］or［条件２］or［条件３］を満足．
(c) ソフト処理が非クラスタード索引アクセスの場合
　索引に適用可能な探索条件が［条件３］を満足．

3.4.2　ソート／グループ化処理

(1) $Sp^*(S,Cb)<Sp(R)$となる十分条件

上式を整理すると，$\mu a<1/mp/x+4\mu p+2\mu p/x$.

(a) $\mu p=1$の時，明らかに$Sp^*(S,Cb)<Sp(R)$.

(b) $\mu a=\mu p$の時，明らかに$Sp^*(S,Cb)<Sp(R)$.

(c) $1/mp/x=10^{-2}$程度を仮定すると，$\mu a<10^{-2}$の時，

$\mu a<10^{-2}\fallingdotseq 1/mp/x$より，$Sp^*(S,Cb)<Sp(R)$が成立する.

(2) $Sp^*(S,Nb)<Sp(R)$となる十分条件

上式を整理すると，$\mu aTp/Bp<1/mp/x+4\mu p+2\mu p/x$.

$1/mp/x=10^{-2}$，$Tp/Bp=10^{2}$程度を仮定すると，$\mu a<10^{-4}$の時，

$\mu aTp/Bp<10^{-2}\fallingdotseq 1/mp/x$より，$Sp^*(S,Nb)<Sp(R)$が成立する.

(3) $Sp(S,Ca)<Sp(R)$となる十分条件

上式を整理すると，$\mu a+\mu p\ \log_k(\mu pBp)<1/mp/x+2\mu p/x$.

$\mu p<\mu a$より，$\mu a+\mu p\ \log_k(\mu pBp)<\mu a+\mu a\ \log_k(\mu aBp)$.

$1/mp/x=10^{-2}$，$k=10$程度，$Bp<10^{10}$を仮定すると，

$\mu a<10^{-3}$の時，$\mu a+\mu a\ \log_k(\mu aBp)<10^{-2}\fallingdotseq 1/mp/x$より，

$Sp(S,Ca)<Sp(R)$が成立する.

(4) $Sp(S,Na)<Sp(R)$となる十分条件

上式を整理すると，$\mu aTp/Bp+\mu p\ \log_k(\mu pBp)<1/mp/x+2\mu p/x$.

$\mu p<\mu a$より，$\mu aTp/Bp+\mu p\ \log_k(\mu pBp)<\mu aTp/Bp+\mu a\ \log_k(\mu aBp)$.

$1/mp/x=10^{-2}$，$k=10$程度，$Bp<10^{10}$を仮定すると，

$\mu_A<10^{-5}$の時，$\mu aTp/Bp+\mu a\ \log_k(\mu aBp)<10^{-2}\fallingdotseq 1/mp/x$.

よって，$Sp(S,Na)<Sp(R)$が成立する.

(5) 判定基準の決定

上記の十分条件を整理すると単純検索と同じ分類となる．従って単純検索で設定したＳＱＬ文抽出可能条件を利用して判定基準を決定した.

【ＯＢ／ＧＢ処理の判定基準】

ソフト処理とする判定基準

(A) ソフト処理が非ソート方式の場合

(a) クラスタード索引アクセスの場合

索引に適用可能な探索条件が［条件１］or［条件２］or［条件３］を満足.

(b) 非クラスタード索引アクセスの場合

索引に適用可能な探索条件が［条件３］を満足.

(B) ソフト処理がソート方式の場合

索引アクセスで，索引に適用可能な制限条件が［条件３］を満足.

3.4.3　結合処理

3.4.3.1　索引マージ結合の場合

簡単化のため，結合列に付与された索引の種類は同じとした．また，$\mu b\leqq\mu c$となるように表Ｐ，Ｑを選ぶ．この条件で，Ｊim＜Ｊrとなる十分条件を求める.

(1) 両索引が共にクラスタードの場合

Ｊim＝μjBp＋μjBqを考える．μj≦min(μb, μc)の関係，μb≦μcを仮定すると，Ｊim≦μbBp＋μbBqとなる．ここでＪim＜Ｊrを次の様に整理する．

μbBp＋μbBq＜２μpBp＋２μqBq＋・・・

(a) μp＝μq＝１の時，明らかにＪim＜Ｊr.

(b) μb＝μp and μc＝μqの時，

μbBp＋μbBq ＜μbBp＋μcBq (∵μb＜μc)

＝μpBp＋μqBq＜２μpBp＋２μqBq＋・・・

となり，Ｊim＜Ｊrが成立する．

(c) Ｊim＜Ｊrを以下のように再整理すると，

μbBp＋μbBq＜Bp／mp／ｘ＋Bq／mq／ｘ＋・・・．

１／mp／ｘ，１／mq／ｘが共に１０$^{-2}$程度を仮定する．μb＜１０$^{-2}$の時，

μb＜１０$^{-2}$≒１／mp／ｘ，μb＜１０$^{-2}$≒１／mq／ｘ．

よって，本条件はＪim＜Ｊrの十分条件となる．

(2) 両索引が共に非クラスタードの場合

Ｊim＝μjTp＋μjTqを考える．μj≦min(μb, μc)の関係及びμb＜μcを仮定すると，Ｊim≦μbTp＋μbTqとなる．上記条件を用いてＪim＜Ｊrを整理すると，μbTp＋μbTq＜Bp／mp／ｘ＋Bq／mq／ｘ＋・・・

１／mp／ｘ，１／mq／ｘが共に１０$^{-2}$程度，またTp／Bp，Tq／Bqが共に１０^{2}程度を仮定すると，μb＜１０$^{-4}$の時，以下が成立する．

μbTp／Bp＜１０$^{-2}$≒１／mp／ｘ，μbTq／Bq＜１０$^{-2}$≒１／mq／ｘよって，本条件は，Ｊim＜Ｊrの十分条件となる．なお，Ｊim（片方が非クラスタード索引）＜Ｊim（両方が非クラスタード索引）は自明より，本条件は少なくとも片方が非クラスタード索引でも十分条件となる．

(3) 判定基準の決定

上記の十分条件を整理すると単純検索と同じ分類となる．従って単純検索で設定したＳＱＬ文抽出可能条件を利用して決定した．

【索引マージ結合処理の判定基準】

ソフト処理とする判定基準．

(a) 両表をクラスタード索引アクセスする場合

索引に適用可能な探索条件が［条件１］or［条件２］or［条件３］を満足．

(b) 一方の表を非クラスタード索引アクセスする場合，索引に適用可能な探索条件が［条件３］を満足．

3.4.3.2　ソートマージ結合の場合

入出力回数が大きくなる全ブロックアクセスを含む組み合わせは除外し，索引アクセスの組み合わせのみを扱う．また使用する両索引は同じものと仮定する．これらの条件より，Ｊsm＜Ｊrとなる十分条件を求める．

(1) 両索引がクラスタードの場合

Ｊsm＝Sp(S, Ca)＋Sq(S, Cd)＋２Tjを考える．Ｊsm＜Ｊrを整理すると

μaBp＋２μpBp＋μpBp $\log_k$(μpBp)＋μdBq＋２μqBq＋

μqBq $\log_k$(μqBq)＜Bp／mp／ｘ＋Bq／mq／ｘ＋２Ts

上式が成立する十分条件は，

μa＋２μp＋μp $\log_k$(μpBp)＜１／mp／ｘ　　(α1)

μd＋２μq＋μq $\log_k$(μqBq)＜１／mq／ｘ　　(β1)

１／mp／ｘ＝１０$^{-2}$，ｋ＝１０程度，Bp＜１０^{10}を仮定すると，μa＜１０$^{-3}$の時，μa＋２μp＋μp $\log_k$(μpBp)＜３μa＋μa $\log_k$(μaBp)

(∵μa＞μp) ＜１０$^{-2}$≒１／mp／ｘ．よって，この条件の時，式(α1)が成り

立つ．同様に，$\mu d < 10^{-3}$の時，式($\beta 1$)が成り立つ．以上より，
$\mu a < 10^{-3}$，$\mu d < 10^{-3}$の時，Ｊsm＜Ｊrが成立する．

(2) 両索引が非クラスタードの場合

Ｊsm＝Sp(S,Na)＋Sq(S，Cd)＋２Tjを考える．Ｊsm＜Ｊrを整理すると，
$\mu aTp + 2\mu pBp + \mu pBp \log_k(\mu pBp) + \mu dTq + 2\mu qBq + \mu qBq \log_k(\mu qBq) < Bp/mp/x + Bq/mq/x + 2Ts$．

上式が成立する十分条件は，

$\mu aTp/Bp + 2\mu p + \mu p \log_k(\mu pBp) < 1/mp/x$　　($\alpha 2$)

$\mu dTq/Bq + 2\mu q + \mu q \log_k(\mu qBq) < 1/mq/x$　　($\beta 2$)

$1/mp/x = 10^{-2}$，$Tp/Bp = 10^{-2}$，$k = 10$程度，$Bp < 10^{10}$を仮定すると，$\mu a < 10^{-5}$の時，$\mu aTp/Bp + 2\mu p + \mu p \log_k(\mu pBp) < \mu aTp/Bp + 2\mu a + \mu a \log_k(\mu aBp)$ ($\because \mu a > \mu p$) $< 10^{-2} \fallingdotseq 1/m_P/x$．よって，この条件の時，式($\alpha 2$)が成り立つ．同様に，$\mu d < 10^{-5}$の時，式($\beta 2$)が成り立つ．以上より，$\mu a < 10^{-5}$，$\mu d < 10^{-5}$の時，Ｊsm＜Ｊrが成立する．

(3) 判定基準の決定

上記十分条件は全て単純検索の$\mu p \leqq \mu a < \alpha$の場合であり，単純検索で設定したＳＱＬ文抽出可能条件を利用して判定基準を設定した．

【ソートマージ結合処理の判定基準】

ソフト処理とする判定基準．

(a) 両表を共に索引アクセスする場合

両索引各々に対し適用可能な探索条件が［条件３］を満足

3.4.4　判定基準適用例

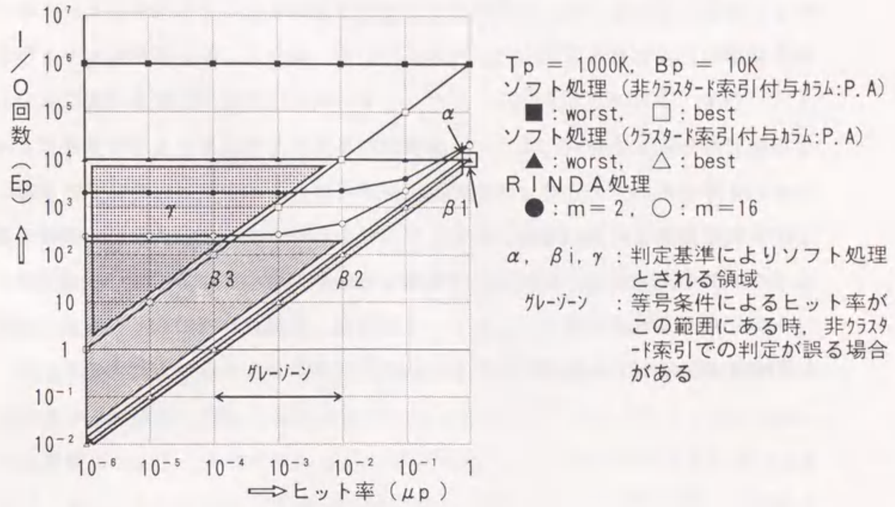


図3.7　単純検索処理への判定基準適用例

Figure 3.7　Access path selection in simple queries

3.3節で示した単純検索の入出力回数結果例（図3.5(a)）に対して判定基準を適用した例を図3.7に示す．判定基準a，b，cによりソフト処理と判定される領域をα，β（β1～β3）及びγで示す．β1～β3は判定基準bの条件1～3に対応している．特にβ3及びγは索引に適用可能な等号条件のヒット率に判定基準の正誤が左右されることを示している．例えば，等号条件のヒット率が10^{-3}の時，β3での判定は正しいが，γでの判定は誤る．しかし，先にＳＱＬ文抽出可能条件3で述べたように「索引が付与された列に対する等号条件のヒット率は十分小さい」ことを考えると判定を誤るケースは少ないと考える．また，全ての判定基準は，索引定義，ＳＱＬ文情報のみから求めているため，判定を誤る場合もある．例えば，範囲条件でも指定範囲が十分小さく等号条件と同等のヒット率となると判定を誤る．しかし，この場合，汎用計算機のＣＰＵ負荷を大幅に削減するＲＩＮＤＡ処理となるため大きな問題にはならないと考える．

3.5　まとめ

データベースプロッセッサＲＩＮＤＡとソフトウェアのＤＢ処理を統合したＤＢＭＳのアクセスパス決定方式について示した．アクセスパス決定は，原則はＲＩＮＤＡ処理とし，十分少ない入出力回数で処理可能な場合のみソフト処理とする方式である．その判定基準は，処理をモデル化して求めた入出力評価式から，ソフト処理の入出力回数が十分小さくなる十分条件を導き，ソフト処理のアクセス方法，索引定義情報，ＳＱＬ文情報を用いて決定した．また，本文中では述べなかったが，ネステッドループ方式を用いる場合にも上記と同様な考え方で判定基準を設定できる．なお，上記の判定基準は索引定義情報，ＳＱＬ文情報のみから求めているため，ソフト処理の方が速い場合を全て網羅していない。しかし，簡単に取得できる情報のみ用いて大部分のケースについて正しいアクセスパスの選択を実現している点で実用的な方法である．この結果，専用プロセッサの使用有無を利用者が意識することなく共通のインタフェースでデータベースアクセスが可能になり，ＣＰＵネック，メモリネック，及び入出力ネックの解消を目指した第1のアプローチ，データベース専用プロセッサ方式の適用性を向上できた．

4　並列処理における動的負荷配分方式

4.1　まえがき

第1章で述べたＣＰＵネックを解消するための第2のアプローチ，汎用プロセッサの並列処理を実現するためには，プロセッサを効率よく働かせるための負荷の動的配分方式が必要不可欠である．本章では，この動的負荷配分方式について議論する．

第2章及び第3章で述べた専用プロセッサを用いる方法は，ＣＰＵネックと入出力ネックの両方に対して効果がある反面，その効果は専用プロセッサが分担し得る機能についてのみ限定される．一方，汎用プロセッサの並列処理を用いる方法は，幅広い機能に対して効果が期待できる．特にリレーショナルデータベースは，もともと互いに独立なデータの集合として定義されているため，データ分割による並列処理に向いていると考えられる．なお，1つのデータベース管理システムで2つのアプローチを併用することも可能である．例えば，前章で述べたＣＳＰのような専用プロセッサでサーチ処理に伴う入出力ネックを解決した上で，残りの処理に汎用マイクロプロセッサの並列処理を適用することにより，ＣＰＵネックを解決する方法も有力である．

以下では，プロセッサ全体の処理能力を有効に利用することが比較的容易な全資源共有型のマルチプロセッサシステムを用いて，大規模なリレーショナルデータベースの並列処理を行う際の，処理対象となるデータをプロセッサに動的に配分する負荷配分方式を扱う．

従来の負荷配分法では，1回に配分するデータ数，即ち配分単位は固定であった．この配分単位の大きさによって負荷配分のオーバヘッドとプロセッサのアイドル時間が変化するため，性能が大きく変動する．しかも，配分単位の最適値を事前に決定できないという問題点があった．

本章で提案する負荷配分法は，処理の進行にともなって配分ページ数を動的に

変更することによって全体の処理時間の変動を小さくし，しかも負荷配分のオーバヘッドを削減することを目的とする．以下，4.2節で従来の負荷配分法の問題点を指摘し，4.3節でその問題を解決した新しい負荷配分法を提案する．4.4節では，提案法の有効性を定量的に示す．

4.2　従来の負荷配分法とその問題点

4.2.1　対象とするモデル

本章で扱うデータベース問合せ処理の条件を以下に示す．

① 全資源共有型マルチプロセッサで，表内の全数検索を伴う問合せを行う．

② データベース中の表は，固定サイズの複数のページに分割されている．

③ ページの総数（≫プロセッサ数）が既知である．

④ 各ページはどのプロセッサに対しても同じ時間で配分できる．

⑤ 処理はページ毎に独立に実行でき，結果は実行順序に依存しない．

⑥ ページ当たりの処理時間の最大値と最小値を事前に予測できる．

以上の前提のもとでページのプロセッサへの配分を決定する負荷配分法を検討する．負荷配分の目的は，単一の問合せの処理が開始されてから終了するまでの経過時間（以後応答時間という）をできるだけ短くすることである．このモデルでは，並列処理を行なう場合でも各プロセッサが本来の問合せ処理自体を実行している時間の総和は，逐次処理の場合に要する時間と同一であるから，応答時間を短縮するという負荷配分の目的は，並列処理によって生じるオーバヘッド時間をできるだけ少なくすることと等価である．

4.2.2　従来の負荷配分法

独立に実行可能なタスク（負荷配分の対象となる処理の単位．本章では1ページの処理）が複数のタスク源で生成され，それらのタスクを複数のプロセッサで

実行する場合の負荷配分法には，以下の2種類がある[45]．

(a) タスク源主導型

タスクが到着すると，タスク源がプロセッサにタスクを送る．

(b) プロセッサ主導型

プロセッサがタスク実行可能になるとタスク源にタスクを取りに行く．

これらのうち，負荷配分に使用する情報のレベルが同じであれば，(b) のプロセッサ主導型の方が一般には性能がよい[45]．前節のモデルにプロセッサ主導型の負荷配分を用いると以下のようになる．

① 各プロセッサは一定数のページを取り出して処理する．処理を終えると，次のページを取りに行く．

② 全てのページが処理されるまで，①の処理を繰り返す．

上記の①で，1回に取り出すページ数を配分単位と呼ぶ．従来の負荷配分法の例（配分単位が3ページの場合）を図4.1に示す．

4.2.3　従来の負荷配分法の問題点

前節で述べた従来の負荷配分法では，オーバヘッド時間は以下の2つに分類できる．

(a) 負荷配分時間

取り出すページを決定する時間，その間の排他制御および待ち時間，通信時間など．一般に，負荷配分時間は主に負荷配分回数に比例し，負荷配分回数は配分単位に反比例する．

(b) アイドル時間

全てのページの配分が終わった後に処理すべきページがなくなったプロセッサが，最後まで処理しているプロセッサが実行を終えるまで待つ時間．1プロセッサ当たりのアイドル時間は配分単位分のページの処理時間以下になる．

負荷配分時間は配分単位を大きくすると減少し，アイドル時間は配分単位を大きくすると増大する．従って，図4.2(1) のように負荷配分時間とアイドル時間はトレードオフの関係になり，オーバヘッド時間を小さく抑えるには適当な配分

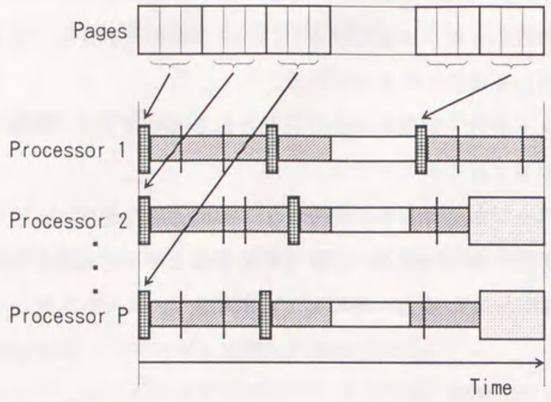
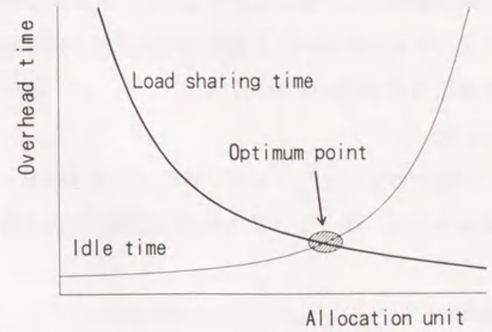
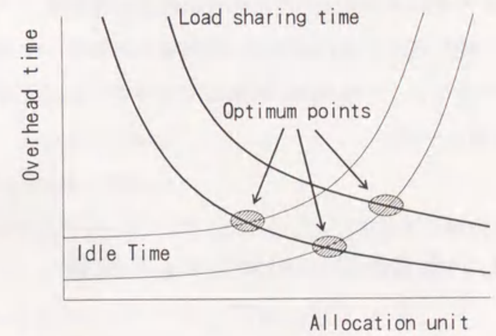


図4.1　従来の負荷配分法

Figure 4.1　Ordinary load sharing algorithm



(1) Tradeoff



(2) Variations of optimum values

図4.2　負荷配分時間とアイドル時間

Figure 4.2　Load sharing time and idle time

単位を決定することが重要になる．しかし，データベースの並列処理における配分単位の最適値は，以下の要因によって変動すると考えられる（図4.2(2)）．

① ページ数（データベースサイズ）

負荷配分の回数はページ数に比例するので，負荷配分時間はページ数に比例する．一方，アイドル時間はページ数には無関係である．従って，ページ数が多くなると配分単位の最適値は大きくなる．

② プロセッサ数

プロセッサ数が増えると，アイドル状態になるプロセッサが増えるのでアイドル時間が大きくなる．従って，プロセッサが多くなると配分単位の最適値は小さくなる．

③ データ分布

ページ毎の処理時間は，ページ内のデータ数や選択されるデータ数によって変化する．アイドル時間は終了間際に配分されたページの処理時間に依存する．従って，データ分布によって配分単位の最適値が変化する．

上記の内，一般にデータ分布は事前には判らないため，配分単位の最適値を決定することができない．そのため，従来法では必ずしも良好な性能は得ることができなかった．

4.3 新しい負荷配分アルゴリズムの提案

4.3.1 方針

従来の負荷配分法では，負荷配分時間とアイドル時間の間にトレードオフの関係があり，しかも，配分単位の最適値を事前に決定できないため，必ずしも最適な性能は得られないという問題があった．この問題を解決するために，アイドル時間を小さく抑えたままで負荷配分時間を大幅に削減することを考える．即ち，アイドル時間が小さい（配分単位が小さい）領域で負荷配分時間を，その領域でのアイドル時間程度に小さくできれば，オーバヘッド時間をその領域内でほぼ一定にでき，配分単位の大きさに関係なく最適に近い性能が得られる．

前節で述べたように，負荷配分時間は主に配分回数に依存し，アイドル時間は主に処理終了間際の配分単位に依存する．そこで，図4.3に示すように以下の方法によって，アイドル時間を小さく抑えたままで配分回数の削減を図る．

① 最初は多くのページを配分することによって，負荷配分回数を減らす．

② 処理の進行に従って配分ページ数を徐々に少なくすることによって，アイドル時間を小さくする．

4.3.2 配分ページ数決定法

配分ページ数とアイドル時間の関係を定式化し，次にアイドル時間を許容限度以下にできる配分ページ数の最大値を求める．以下の場合を考える．

- nページをP個のプロセッサで処理する．
- プロセッサi（$1 \leq i \leq P$）にbiページを配分する．
- プロセッサiがbiページを処理し終える前に，他の（P－1）個のプロセッサが，残りの（n－bi）ページの処理を終える．

この場合でアイドル時間が最大になるのは，ページ当たりの処理時間の最大値をTmax，最小値をTminとすると，

- プロセッサiに配分されたページは全て処理時間がTmaxであり，かつ
- 他の（n－bi）ページは全て処理時間がTminである

場合である．この場合のプロセッサiの処理時間は，

$$bi \times Tmax \quad (1)$$

であり，残りのページの処理に要する延べ時間は，

$$(n - bi) \times Tmin \quad (2)$$

であるから，アイドル時間の総和は，

$$bi \times Tmax \times (P-1) - (n - bi) \times Tmin \quad (3)$$

となる．

アイドル時間を小さく抑えたまま多くのページを配分するために，式(3)のアイドル時間を与えられた許容限度Tidl以下にできる最大のbiを求める．

$$bi = (n + Tidl / Tmin) / \{(Tmax / Tmin) \times (P-1) + 1\} \quad (4)$$

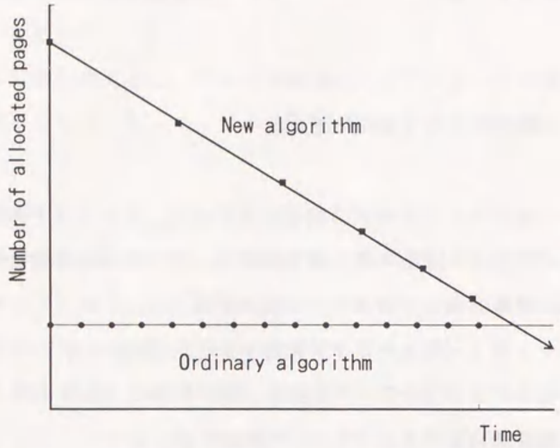


図4.3　動的な配分ページ数決定

Figure 4.3　Dynamic determination of number of pages

式(4)を，配分の度に繰り返し用いるように書き換えると次式が得られる．

$$bi(t)=\{n(t)+\sum_{j=1}^{P} mj(t)+\alpha\}\ /\beta-mi(t) \qquad (5)$$

$$\alpha=Tidl/Tmin \qquad (6)$$

$$\beta=(Tmax/Tmin)\times(P-1)+1 \qquad (7)$$

ただし，

bi(t)：時刻tにプロセッサiに配分するページ数（i＝1，2，･･･，P）

n(t)　：時刻tにおける未配分ページ数

mj(t)：時刻tにおけるプロセッサjの未処理ページ数（j＝1，2，･･･，P）

Tidl　：アイドル時間の総和の許容限度

（ただし，Tidl≧Tmax×（P－1））

Tmax　：1ページ当たりの処理時間の上限

Tmin　：1ページ当たりの処理時間の下限

P　　：プロセッサ数

なお，式(5)で，

$$n(t)+\sum_{j=1}^{P} mj(t)$$

が時刻tにおいてまだ処理されていないページ数（式(4)ではn）であり，配分されたがまだ処理されていないページ数mi(t)を差し引いて，配分ページ数を決定している．

式(5)を用いた負荷配分法は以下のようになる．

〔方法1〕

配分されたページの処理を終えた（mi(t)＝0となった）プロセッサが，他のプロセッサの未処理ページ数mj(t)を収集して式(5)を計算し，bi(t)ページを取り出し，処理する

4.3.3　データ収集問題

前節で述べた方法１では，配分１回につきＰ－１個のｍｊ(t)を収集する必要がある．しかし，ｍｊ(t)を収集してからページを配分するまでの間に，他のプロセッサでは処理が進行するため，ｍｊ(t)の正確な値を用いることはできない．本節では，負荷配分以前に収集した値を用いて配分ページ数を決定する方法を示す．

未処理ページ数ｍｊ(t)は，以下のように分解できる．

$$mj(t) = aj(t) - rj(t) - pj(t) \qquad (8)$$

ただし，

ａｊ(t)：時刻ｔまでにプロセッサｊに配分したページ数

ｒｊ(t)：時刻ｔまでにプロセッサｊが処理し，既に報告されたページ数

ｐｊ(t)：時刻ｔまでにプロセッサｊが処理したが，まだ報告されていないページ数

負荷配分を行うプロセッサが式(8)の計算で利用可能なのはａｊ(t)およびｒｊ(t)であるから，ｍｊ(t)の近似式として次の２つを利用できる．

$$mj'(t) = aj(t) - rj(t) \geqq mj(t) \qquad (9\text{-}1)$$

$$mj''(t) = 0 \leqq mj(t) \qquad (9\text{-}2)$$

そこで，アイドル時間を許容限度Ｔidl以下にするために，配分ページ数が式(5)のｂｉ(t)より小さくなる方向に近似すると，

$$+\sum_{j=1}^{P} mj(t) \rightarrow +0 \qquad (10)$$

$$-mi(t) \rightarrow -\{ai(t) - ri(t)\} \qquad (11)$$

となる．式(10)では式(9-2)，式(11)では式(9-1)の近似を用いた．以上より，

$$bi'(t) = (n(t)+\alpha)/\beta - \{ai(t) - ri(t)\} \qquad (5')$$

が得られる．

方法１で式(5)の代わりに式(5')を用いると，配分されたページの処理を終えたプロセッサ（プロセッサｉとする）では

$$ai(t) - ri(t) = 0$$

であるから，次の方法２が得られる．

〔方法２〕

配分されたページの処理を終えたプロセッサが次式で計算した数のページを取り出して，処理する．

$$b''(t) = (n(t)+\alpha)/\beta \qquad (12)$$

方法２では，近似により１回の配分ページ数が少なくなるため，配分回数は方法１を実現した場合より多くなる．しかし，配分毎に他のＰ－１個のプロセッサからデータを収集しなくて済むので，負荷配分のオーバヘッドが小さく，また，実装が簡単である．従って，方法２は単なる近似法ではなく，方法１より優れた方法であると考えられる．

4.4　性能評価

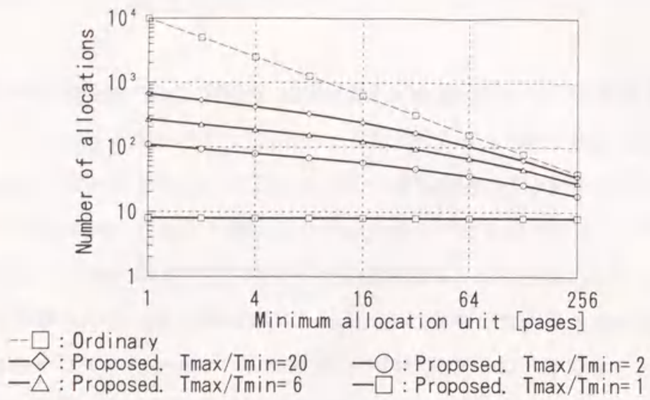
本節では，まず，前節で提案した負荷配分法の性質をシミュレーションにより評価し，次に，提案法と従来法を資源共有型マルチプロセッサに実装して，負荷配分時間とアイドル時間，応答時間を実測により検証する．

4.4.1　シミュレーション結果

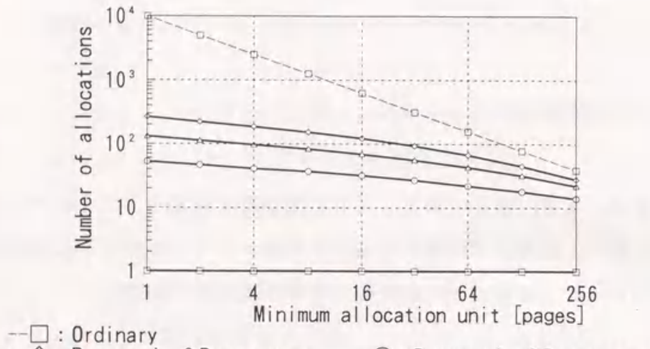
ページ当たりの処理時間の上限と下限の比Ｔmax／Ｔmin，プロセッサ数，ページ数が配分回数に与える影響をシミュレーションにより評価した．ページ当たりの処理時間は正規乱数とし，負荷配分時間は無視した．

(1) Ｔmax／Ｔminの影響

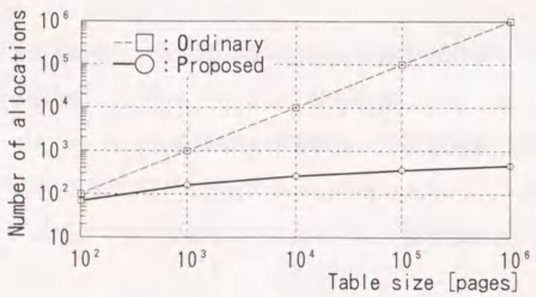
ページ数ｎ(0)を11,112ページ，プロセッサ数Ｐを８として，Ｔmax／Ｔminを変えて評価した結果を図4.4(1)に示す．



(1) Effect of Tmax/Tmin



(2) Effect of number of processors



(3) Effect of number of pages

図4.4　シミュレーション結果
Figure 4.4　Simulation results

図4.4(1)の横軸の最小配分単位は，従来法の配分単位に等しく，提案法では処理終了間際の配分ページ数で，配分ページ数の最小値となる．アイドル時間の許容限度Tidlとは以下の関係がある．

最小配分単位b→Tidl＝b×（P－1）×Tmax　　(13)

図4.4(1)より，Tmax／Tmin比が大きくなると配分回数が増加するが，この比が20程度以下であれば，最小配分単位が小さい領域では従来法より配分回数を1桁以上少なくできることが判る．

次節で述べる並列計算機への実装では，単純選択のTmax／Tmin比は約6であった．従って，選択されたデータに対してのみ実行される処理の時間が短くTmax／Tmin比が小さいような単純な問い合わせでは，提案法により負荷配分時間を十分小さくできると考えられる．なお，この比が1の場合，全てのページを各プロセッサに均等に配分する静的負荷配分に相当する．

(2) プロセッサ数の影響

ページ数n(0)を11,112ページ，Tmax／Tminを6とし，プロセッサ数Pを変えて評価した結果を図4.4(2)に示す．

プロセッサ数Pが大きくなると，配分回数が増加する．プロセッサ数Pが1の場合，全てのページを1回で配分するので，並列処理を行わない場合でもアルゴリズムを変える必要がない．

(3) ページ数の影響

プロセッサ数Pを8，Tmax／Tminを6，最小配分単位を1とし，ページ数n(0)を変えて評価した結果を図4.4(3)に示す．

ページ数の増加に対して配分回数はサブリニアになる．これは，配分ページ数が未処理ページ数に比例するため，ページ数が多いと処理の初期に一度に多くのページを配分するためである．したがって，提案法はデータベースが大きいほど有利になるといえる．

4.4.2　実装と評価モデル

従来の負荷配分法および提案した負荷配分法の方法2を資源共有型マルチプロセッサSequent S27に実装し[46]，ウィスコンシン・ベンチマーク[41]を用いて評価した．評価モデルと計算機の仕様を表4.1に示す．

負荷配分法の効果を評価にするために，データベースは主記憶上にあるとし，主記憶上の一時表への検索結果の書き込みが完了するまでの時間を測定した．問い合わせの解析やプロセス起動の時間は測定対象から除外した．

評価項目は以下の3点である．

① アイドル時間と負荷配分時間

② 応答時間

③ スピードアップ率

4.4.3　負荷配分時間とアイドル時間

最小配分単位を変えて負荷配分時間とアイドル時間を実測した結果を図4.5に示す．プロセッサ数は8である．提案法では，アイドル時間は従来法と同程度のままで，最小配分単位が小さい領域での負荷配分時間を極めて小さくできていることが判る．

図4.5において，特に最小配分単位の大きい領域では，提案法のアイドル時間は従来法より少し長くなっている．これは，提案法では最初に多くのページを配分するためプロセッサ間で処理時間のばらつきが大きく，徐々に配分するページ数を少なくすることによってばらつきを小さくしていくが，最小配分単位が大きい場合にはばらつきを充分吸収できないためと考えられる．なお，図4.5で例えば最小配分単位が64の場合，アイドル時間の許容限度は式(13)より

$$64\times(8-1)\times12.7=5.7[\mathrm{sec}]$$

であり，アイドル時間を許容限度以下にするという条件は満たしている．

表4.1　評価モデル

Figure 4.1　Evaluation model

項目	内容
データベース	ウィスコンシンモデル
行サイズ	208B
行数	100,000
ページサイズ	2KB
ページ数	11,112
格納場所	主記憶
問合せ	単純選択
選択率	10%
計算機	シークエントS27
プロセッサ	i80386(16MHz)×8
主記憶容量	40MB
ページ当たりの処理時間	実測値
最大値Tmax	12.7 msec
最小値Tmin	2.2 msec

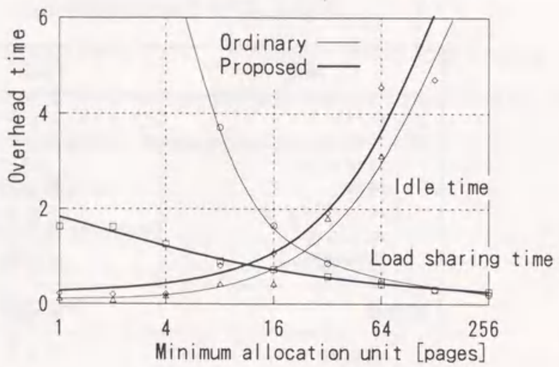


図4.5　負荷配分時間とアイドル時間
Figure 4.5　Load sharing time and idle time

4.4.3　応答時間

4.2節で述べた配分単位の最適値に影響を与える以下の3つの要因をパラメータとして，応答時間を実測した結果を図4.6に示す．

①　ページ数（1,112 または 11,112ページ）

②　プロセッサ数（4または8プロセッサ）

③　データ分布（ステップ分布（選択されるデータが終了間際に処理されるページに集中）またはランダム分布（全てのページに一様に分布））

従来法では，負荷配分時間とアイドル時間のトレードオフのために最小配分単位を変えると応答時間が大きく変化している．しかも，配分単位の最適値は上記の要因によって変化するため，最適値を事前に決定することはできなかった．

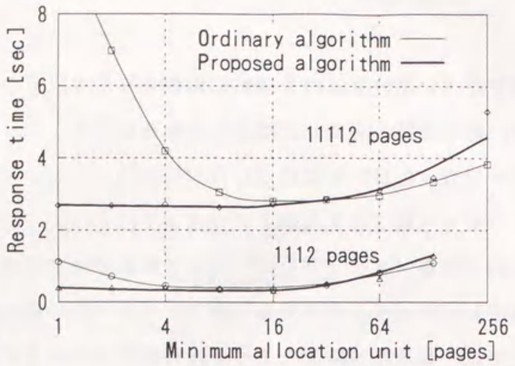
一方，提案法では，最小配分単位が小さい範囲では応答時間が一定になっている．そのため，最小配分単位を1で固定しても，①から③の何れのパラメータに対しても安定して，ほぼ最適な性能を得られている．しかも，従来法で配分単位の最適値を選んだ場合より応答時間を短縮できている．

最小配分単位が大きくなると，提案法の応答時間は従来法よりも長くなっている．これは，図4.5が示すように，提案法のアイドル時間は従来法より少し長いうえ，最小配分単位が大きくなると提案法の負荷配分時間が従来法と同程度になるためである．しかし，提案法は最小配分単位の小さい領域を使用するため，最小配分単位が大きい領域での振舞いは，提案法の優位性には影響しない．

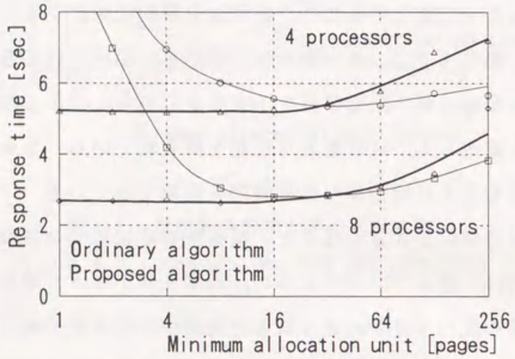
4.4.5　スピードアップ率

プロセッサ数を変えてスピードアップ率を測定した結果を図4.7に示す．従来法ではプロセッサ数を6以上にすると速度向上度がプロセッサ数に対してサブリニアになる．提案法では，従来法よりもスピードアップ率が高く，プロセッサ数に比例して直線的に性能が向上している．

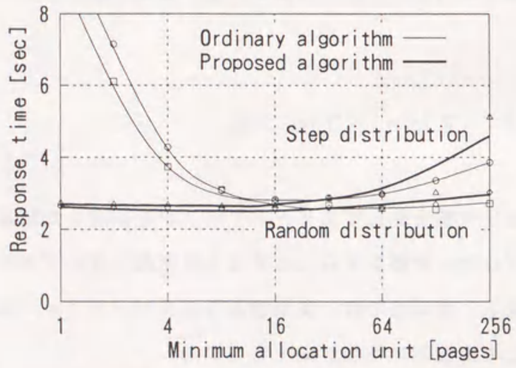
提案法ではプロセッサ数を増やすと配分回数が増加することがシミュレーションにより示されたが，プロセッサ数を増すと，むしろ，従来法に対して有利にな



(1) Effect of number of pages



(2) Effect of number of processors



(3) Effect of data distribution

図4.6　応答時間

Figure 4.6　Response time

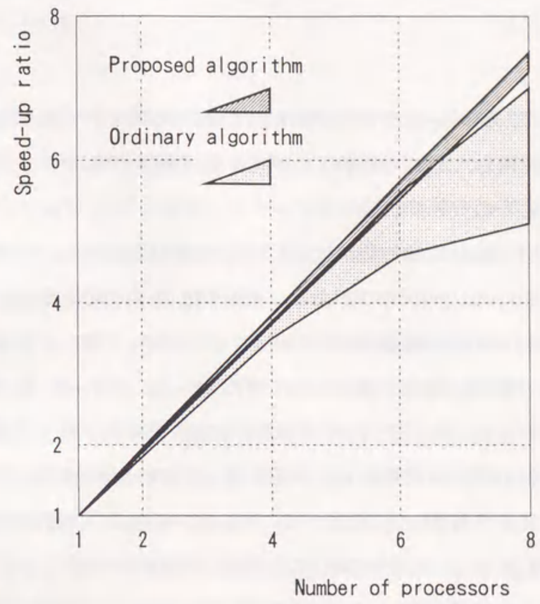


図4.7　スピードアップ率

Figure 4.7　Speed-up ratio

ることが判る．これは，プロセッサを増やすと4.2節で述べたようにアイドル時間が増加するが，提案法ではアイドル時間の小さい，最小配分単位の小さい領域を利用できるためと考えられる．

4.5　まとめ

本章では，データベースへの問い合わせの並列処理を行う際の従来の負荷配分法の問題点を指摘し，それを解決する新しい負荷配分アルゴリズムを提案し，性能評価によってその有効性を示した．

ページを固定の配分単位で配分する従来の負荷配分法では，負荷配分時間とアイドル時間の間にトレードオフがあり，配分単位を変えると性能が大きく変化した．しかも，配分単位の最適値はページ数，プロセッサ数，データ分布によって変化するため，事前に決定することができなかった．そこで，配分ページ数を動的に変えることによって，アイドル時間を増加させることなく負荷配分時間を削減して，最小配分単位に依存しない少ないオーバヘッドを実現できる負荷配分法を提案した．資源共有型マルチプロセッサを用いた性能評価の結果，提案法の性能は従来法より安定し，しかも高いことが明らかになった．

以上の結果，大規模なリレーショナルデータベースに対する非定型処理の実現方式として，第2のアプローチである汎用プロセッサの並列処理も極めて有効であり，ＣＰＵネックの問題を解決できる見通しが得られた．

5　多版管理による並行処理制御方式

5.1　まえがき

本章では，第1のアプローチと第2のアプローチの共通の課題であるデータネックを解消するための多版管理による並行処理制御方式について議論する．

これまで述べてきた方法により，リレーショナルデータベースの単独の非定型処理を高速化することが可能になった．また定型処理については，従来のデータベース管理システムで利用されているインデックスの手法により，もともと十分な性能が得られている．しかしながら，非定型処理と定型処理を同時に実行した場合には，データネックによっていずれかの処理がブロックされ，システムの処理能力が低下する．以下では，定型処理の代表例である部分更新処理と非定型処理の代表例である全数検索処理間の並行性の高い新しい多版並行処理制御方式を提案し，その効果をシミュレーションによって検証する．

部分更新処理と全数検索処理の並行性を高める手段として，これまでに2相ロックまたは時刻印に基づく多版並行処理制御方式（2版2相ロック方式，多版時刻印方式）が各種[47][48][49]提案されている．多版並行処理制御では，１つのデータをその更新履歴に相当する複数の版から構成するため，全数検索処理による参照と部分更新処理による更新を１つのデータに対して同時に行うことが許容される．しかし，単版の2相ロック方式や時刻印方式を単純に多版化しただけの従来方式では，部分更新処理と全数検索処理が相互に干渉し合うため，両者の並行性が十分に向上しないという問題があった．またこの問題に対処するため，データの参照のみを行う参照トランザクションを多版時刻印方式で，データの更新を含む更新トランザクションを単版2相ロック方式で制御する多版混成方式[27][50]が提案されているが，部分更新処理を単版2相ロック方式で制御するため，部分更新処理間の並行性が低いという問題があった．

本章では，取り扱う部分更新処理に一定の制限を設けることによって，従来方

式では解決できなかった部分更新処理と全数検索処理の並行性と，部分更新処理間の並行性がともに高い多版時刻印方式を提案する．これは，2版2相ロックを改良した方式と多版時刻印方式を混成した方式と位置づけられる．

以下，5.2節では，本章で扱うデータベースとトランザクションのモデルについて述べる．5.3節では，従来方式の概要とその問題点を示す．5.4節では，2版2相ロック方式を改良するとともに，多版時刻印方式を混成させた新しい方式を提案する．5.5節では，シミュレーションによる評価結果を示し，提案方式の優位性を示す．

5.2 前提条件

5.2.1 データベースモデル

本章で取り扱うデータベースの各データ（行またはページ）は，複数の版により構成される．トランザクションは，データの最新の版に対して更新を施すことにより新しい版を生成する．このとき，1つの版をもとに生成される版は常に1つに制限される．版は，次の3つの状態のいずれかをとる．

① 新版： コミットしていないトランザクションにより生成された版．
② 現版： コミットしたトランザクションにより生成された版のうちで最新のもの．
③ 旧版： コミットしたトランザクションにより生成された版のうちで現版以外のもの．

5.2.2 トランザクションモデル

本章では，以下に示す部分更新処理と全数検索処理の2種類のトランザクションの並行処理について考える．

(1) 部分更新処理

部分更新処理は，データベースから要求条件を満たすデータを選択し，必要に応じてそれらの一部あるいは全体を更新するトランザクションである．トランザクションTによるデータdiの参照をR[di]，更新をW[di]で表すとき，s個のデータからなるデータベースD＝{d1, d2, …, ds} からm個のデータX＝{x1, x2, …, xm} を参照し，そのうちのn個のデータY＝{y1, y2, …, yn} を更新する部分更新処理は次式で示される．

T＝R[X]∪W[Y]（但し，n≦m≦sかつY⊆X⊆D）

なお，トランザクション内では1つのデータに対する参照は更新に先行して行われる必要があるが，参照と更新が連続する必要はない．また，更新すべきデータとその更新値は自身の参照値によってのみ決定されるものとする．

部分更新処理は，座席予約処理のように予約者の要求条件を満たすいくつかの候補座席をデータベースから抽出し，その一部を予約済に更新するトランザクションに代表される．この場合，抽出したすべての座席候補は予約者の要求条件を同等に満たすので，予約座席は候補座席間に従属関係を設けることなく決定することができる．

ここで定義した部分更新処理は，更新すべきデータとその更新値を自身の参照値のみにより決定する性質を持つとしたので，ある部分更新処理で参照のみを行うデータを他の部分更新処理が並行して更新しても直列可能性[27]は損なわれない．つまり，部分更新処理間の直列可能性は，各部分更新処理間の更新操作の相互干渉でデータベースの整合性が損なわれない限り保証される．従って，直列可能性を考える場合，部分更新処理Tはデータの参照操作のみからなるトランザクションTroと，更新したデータに関する参照／更新操作のみからなるトランザクションTrwへ等価に分解できる．

Tro＝R[X]，Trw＝R[Y]∪W[Y]

この分解は，直列可能性の等価性に着目して行われており，操作を単純に分割したのではない．すなわち，TroとTrwはY⊆Xなる関係からR[Y]を共通して行うため，TroとTrwで行う操作の数は元の部分更新処理Tで行う操作の数より多くなる．このような分割が可能であることは，直列可能性を保証する並行処理パ

タンが従来の並行処理制御方式による許容範囲より広いことを意味する．言い換えれば，従来の並行処理制御方式は部分更新処理間の並行性を十分抽出できなかったと言える．　次のような部分更新処理の並行処理における直列可能性について考える．ここで，ｄi(ｎ)とは，部分更新処理Ｔnがｄiを更新して生成した版を表す．

Ｔ1＝Ｒ[ｄ1(0)] Ｒ[ｄ2(0)] Ｗ[ｄ1(1)]

Ｔ2＝Ｒ[ｄ1(0)] Ｒ[ｄ2(0)] Ｗ[ｄ2(2)]

すなわち，Ｔ0が書き込んだｄ1とｄ2の値をそれぞれＴ1とＴ2が参照した後，Ｔ1はｄ1の値，Ｔ2はｄ2の値を更新する．直列可能性の判定法の１つに多版直列化グラフ[30]を用いる方法がある．多版直列化グラフは，並行処理するトランザクションをノードとし，ＴjがＴiにより生成された版を参照するとき，またはＴjがＴiにより参照された版を更新する時に，Ｔi→Ｔjなる有向枝を設けることで形成される．直列可能性は，多版直列化グラフが非巡回グラフであるならば保証される．この判定法によれば，Ｔ0～Ｔ2により形成される多版直列化グラフ図5.1(a)においてＴ1とＴ2間でサイクルが生じるため，一般的にはこの並行処理は直列可能性を保証しない．しかし，Ｔ1とＴ2を次のように分解した時の多版直列化グラフ図5.1(b)にはサイクルが生じないため，この並行処理は直列可能性を保証する．

Ｔ1ro＝Ｒ[ｄ1(0)] Ｒ[ｄ2(0)]

Ｔ1rw＝Ｒ[ｄ1(0)] Ｗ[ｄ1(1)]

Ｔ2ro＝Ｒ[ｄ1(0)] Ｒ[ｄ2(0)]

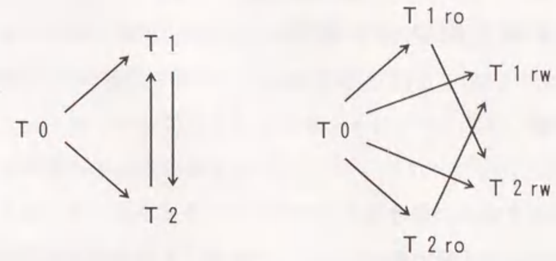
Ｔ2rw＝Ｒ[ｄ2(0)] Ｗ[ｄ2(2)]

これは，座席予約処理Ｔ1とＴ2が同時に同一要求条件で座席候補ｄ1とｄ2を抽出した後，各々異なる座席を並行して予約する例により容易に理解できる．

(2) 全数検索処理

全数検索処理は，データベースのすべてのデータを一定順序で参照するトランザクションからなる．全数検索処理は次式で示される．

Ｔ＝Ｒ[Ｄ]



(a) 巡回　　(b) 非巡回

図5.1　多版直列可能性グラフ

Figure 5.1　Multiversion serialization graphs

全数検索処理は，具体的にはデータベースのすべてのデータに関する統計情報を得るようなトランザクションに代表される．

5.3　従来方式と問題点

5.3.1　2版2相ロック方式

(1) 方式の概要

単版2相ロック方式におけるリードロック，ライトロックに加え，サーティファイロックと呼ばれる新たなロックモードを設けた多版並行処理制御方式である[27]．この方式では，以下の規則に基づきトランザクションを処理するため，1つのデータに対して高々2つの版のみが生成される．

〔参照規則〕トランザクションは，データに対するリードロックを獲得した後に現版を参照する．リードロックは，そのデータが他のトランザクションによりサーティファイロックされていない時に獲得できる．

〔更新規則〕トランザクションは，データに対するライトロックを獲得した後に現版を更新して新版を生成する．ライトロックは，そのデータが他のトランザクションによりライトロックまたはサーティファイロックされていない時に獲得できる．

〔コミット規則〕トランザクションは，獲得したすべてのライトロックをサーティファイロックにモード変換した後にコミットする．ライトロックからサーティファイロックへのモード変換は，データが他のトランザクションによりリードロックされていない時に可能である．トランザクションが獲得したすべてのロックはコミットと同時に解放される．

(2) 問題点

この方式では，コミット規則が部分更新処理と全数検索処理を干渉させる原因

となっている．部分更新処理は，全数検索が参照中のデータを更新できても，全数検索処理が終了するまでコミットできない．全数検索処理の処理時間が一般的に長いことを考慮すると，多くの部分更新処理のコミットが全数検索処理の影響で長時間に渡って遅延されると考えられる．

また，このコミット規則は部分更新処理間の並行性も制限している．5.2.2節に示した部分更新処理T1とT2の並行処理を考える．先に述べたようにこの並行処理で部分更新処理間の直列可能性は保証されるが，T1とT2がそれぞれd1とd2に対して獲得しているリードロックの影響で，T1とT2はともにライトロックをサーティファイロックに変更できずにデッドロックとなり，コミットできる部分更新処理はいずれか一方のみとなる．

5.3.2　多版時刻印方式

(1) 方式の概要

単版の時刻印方式を多版データモデル向きに拡張した多版並行処理制御方式である[27]．トランザクションは処理の開始時刻に相当する時刻印を付与され，データの各版はそれを生成したトランザクションの時刻印であるライト時刻印と，それを参照したトランザクションの時刻印の最大値であるリード時刻印を付与される．時間の経過に対して単調増加する値を時刻印とする時，トランザクションは次の規則に従ってデータをアクセスする．

〔参照規則〕トランザクションは，自身の時刻印以下で最大のライト時刻印を付与された版（新版，現版，または旧版）を選択して参照する．また，参照した版のリード時刻印を必要に応じて更新する．

〔更新規則〕トランザクションは，最新の版（新版，または現版）のリード時刻印とライト時刻印がともに自身の時刻印以下である時，最新の版を更新して新版を生成する．

〔コミット規則〕トランザクションは，参照したすべての版と更新に用いたすべての版が現版となった後にコミットする．

(2) 問題点

この方式の参照規則と更新規則は，異なるトランザクションが同一のデータに施す競合処理（参照と更新，および更新と更新）をトランザクションの時刻印の順序にスケジューリングする．従って，部分更新処理の時刻印が全数検索処理の時刻印よりも小さい場合，全数検索処理が参照したデータを部分更新処理が更新することはできず，従って全数検索処理の影響で部分更新処理がアボートすることがある．

また，部分更新処理間でも同様の干渉が起こる．5.2.2節に示した部分更新処理T1とT2の並行処理で，T2の時刻印がT1よりも大きい場合，T1はT2により参照されたd1を更新できずにアボートする．

5.3.3 多版混成方式

(1) 方式の概要

単版2相ロック方式と前述した多版時刻印方式を組み合わせた多版並行処理制御方式である[50]．この方式の特徴は，トランザクションを，データの参照のみを行う参照トランザクションと，データを更新する更新トランザクションに分類し，各々に異なるデータアクセス規則を設けた点である．この方式では，以下の規則に基づき，トランザクションを処理する．

〔参照規則〕更新トランザクションは，データに対するリードロックを獲得した後に現版を参照する．リードロックは，そのデータが他の更新トランザクションによりライトロックされていない時，獲得できる．一方，参照トランザクションは，処理開始時刻に相当する時刻印を付与され，自身の時刻印以下で最大のライト時刻印を付与された版を選択して参照する．

〔更新規則〕更新トランザクションは，データに対するライトロックを獲得した後に，現版を更新して新版を生成する．ライトロックは，そのデータが他の更新

トランザクションによってリードロックまたはライトロックされていない時，獲得できる．

〔コミット規則〕更新トランザクションの獲得したすべてのロックは，コミットと同時に解放される．この時，生成したすべての版にそれがコミットした時刻，つまり現版となった時刻に相当する時刻印（ライト時刻印）を付与する．一方，参照トランザクションのコミットに伴う操作はない．

(2) 問題点

この方式によれば，更新トランザクションと参照トランザクションは相互に影響を及ぼし合わない．従って，部分更新処理を更新トランザクションとして，全数検索処理を参照トランザクションとして扱うことで，部分更新処理と全数検索処理を全く干渉なく処理できる．

しかし部分更新処理間の並行性は，それらが単版2相ロック方式に基づき制御されるため十分ではない．5.2.2節に示した部分更新処理T1とT2の並行処理で，T1とT2がそれぞれd1とd2に対して獲得しているリードロックの影響で，T1とT2はともにライトロックを獲得できずにデッドロックとなり，コミットできる部分更新処理はいずれか一方のみとなる．

5.4 提案方式

5.4.1 基本方針

従来方式の考察結果を踏まえて以下の方針に基づき，部分更新処理と全数検索処理の並行性と部分更新処理間の並行性がともに高い多版並行処理制御方式を実現する．

(1) 部分更新処理と全数検索処理の並行性向上

従来方式の考察から明らかなように，部分更新処理と全数検索処理に異なるデータアクセス規則を適用する多版混成方式の手法は，これらの間の並行性を向上させる上で非常に有効である．そこで，この手法を基本として部分更新処理と全数検索処理の並行性を向上させることとする．すなわち，部分更新処理はロックを用いたデータアクセス規則で，全数検索処理は時刻印を用いたデータアクセス規則で制御する．

(2) 部分更新処理間の並行性向上

部分更新処理はロックを用いたデータアクセス規則で制御するという観点で，従来の2版2相ロック方式を改良することにより部分更新処理間の並行性を向上させることとする．すなわち，2版2相ロック方式の問題点と部分更新処理間の直列可能性を考慮することにより，データアクセス規則を改良して並行性を抽出する．

5.4.2　2版2相ロック方式の改良

(1) 方式の概要

部分更新処理の並行処理が可能な改良型2版2相ロック方式のデータアクセス規則を以下に示す．

〔参照規則〕トランザクションは，データに対するリードロックを獲得した後に現版または最新の旧版を参照する．リードロックは，次の2つの条件のいずれか一方が成立する時，獲得できる．条件(a)が成立する時には現版を，条件(b)が成立する時には最新の旧版を参照する．

(a) データが他のトランザクションによってサーティファイロックされていない．

(b) データが他のトランザクションによってサーティファイロックされているが，そのトランザクションがサーティファイロックしているデータの少なくとも

1つに対して自身がすでにリードロックを獲得している．

〔更新規則〕トランザクションは，データに対するライトロックを獲得した後に現版を更新して新版を生成する．ライトロックは，そのデータが他のトランザクションによりライトロックまたはサーティファイロックされていない時，獲得できる．

〔コミット規則〕トランザクションは，獲得したすべてのロックを解放してコミットする．ただし，ライトロックを獲得しているデータの少なくとも1つが他のトランザクションによりリードロックされているならば，すべてのライトロックをサーティファイロックに変更する．これらのサーティファイロックは，サーティファイロックしているデータに対するすべてのリードロックが解放される時点で一斉に解放される．

(2) 効果と問題点

改良型2版2相ロック方式は，従来方式の参照規則とコミット規則に変更を加えたものとなっている．すなわち，従来のコミット規則によれば，部分更新処理間で更新したデータの相互にリードロックを獲得していると，ライトロックをサーティファイロックに変更できずにデッドロックが生じて，片方の部分更新処理をアボートする必要が生じる．一方，改良したコミット規則によれば，更新したデータの相互に獲得したリードロックはライトロックをサーティファイロックに変更すべき条件となり，ともにライトロックをサーティファイロックに変更してコミットできる．なお，参照規則の変更は，サーティファイロックの獲得以前にデータをリードロックしたトランザクションが，サーティファイロックの獲得以後も同一の版を参照できるようにするために必要となる．

以上のように，この方式は従来方式で抽出できなかった部分更新処理の並行処理を可能とするため，従来方式に比べて部分更新処理間の並行性が高い．しかし，データを長時間リードロックし続ける全数検索処理がデータのサーティファイロック期間を長くするため，全数検索処理が部分更新処理に影響を及ぼす問題は解消されない．

5.4.3　多版協調方式の実現

(1) 方式の概要

改良型2版2相ロック方式と多版時刻印方式を多版混成方式と同様の手法により組み合わせた多版協調方式を実現する．本方式では，多版混成方式と同様にトランザクションを，データを参照のみのためにアクセスする参照トランザクションとデータを更新する更新トランザクションに分類し，以下の規則に基づき各トランザクションを処理する．

〔参照規則〕更新トランザクションは，改良型2版2相ロック方式の参照規則に従ってデータを参照する．一方，参照トランザクションは，処理開始時刻に相当する時刻印が付与され，自身の時刻印以下で最大のライト時刻印を付与された版を選択して参照する．

〔更新規則〕更新トランザクションは，改良型2版2相ロック方式の更新規則に従ってデータを更新する．

〔コミット規則〕更新トランザクションは，改良型2版2相ロック方式のコミット規則に従ってコミットする．ただし，生成したすべての版にそれを生成した更新トランザクションがコミットした時刻，つまり新版から現版に遷移した時刻に相当する時刻印をライト時刻印として付与する．一方，参照トランザクションのコミットに伴う操作はない．

(2) 効果

本方式において，部分更新処理を更新トランザクションとして，全数検索処理を参照トランザクションとして扱えば，部分更新処理と全数検索処理を全く干渉なく処理できる．また，改良型2版2相ロック方式と同様に，従来方式で抽出できなかった部分更新処理間の並行処理が可能である．従って本方式は，部分更新処理と全数検索処理間の並行性と部分更新処理間の並行性がともに高い多版並行処理制御であるといえる．

5.5　方式評価

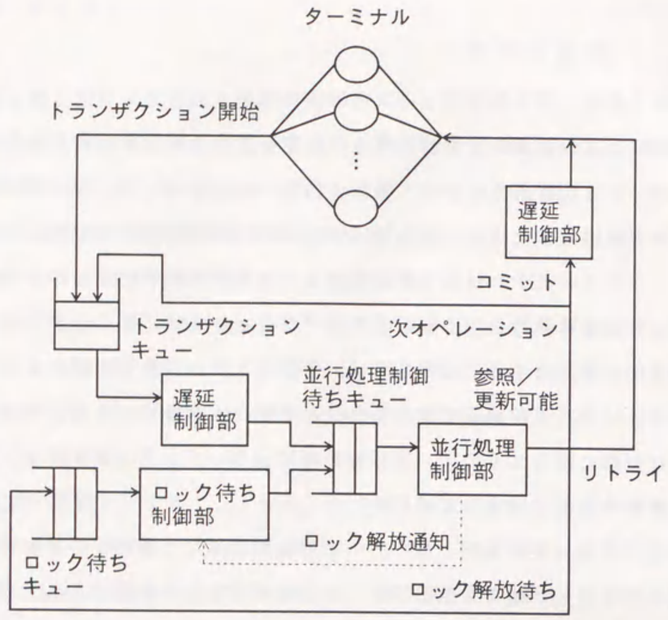
5.5.1　評価モデル

多版協調方式の効果を定量的に明らかにするため，図5.2に示すモデルによりシミュレーション評価を行った．図5.2(a)において，ターミナルは図5.2(b)に示す部分更新処理または図5.2(c)に示す全数検索処理を繰り返し発行する．ただし，1つのターミナルは部分更新処理と全数検索処理のいずれか一方のトランザクションを常に発行し，1つのトランザクションが終了すると直ちに次のトランザクションを発行する．本評価では，各ターミナルのトランザクション発行間隔はシミュレーション過程で生じるトランザクションのロック獲得待ちやアボートにより自然にばらつくので，指数分布等に基づいてトランザクション発行間隔をばらつかせる手法はとっていない．

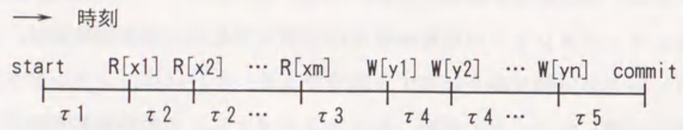
本評価では部分更新処理として，一定の時間間隔でm個のブロックをランダムに選択して参照した後にn個のブロックを更新するものを用いた．一方全数検索処理は，一定の時間間隔でデータベースの全ブロック（s個）を参照するものとした．並行処理制御部は，各方式に応じた並行処理制御を行う部分である．ロック待ち制御部は，並行処理制御部でロックを用いた制御を行う時に発生するロック獲得待ちトランザクションの起動制御を行う部分である．遅延制御部は，図5.2(b)(c)のτ1～8の時間間隔を制御する部分である．τ1～8は，トランザクションがシステムに対してデータの参照，更新等を要求するための内部処理に要する時間を意味し，実際の要求間隔は直前の要求に対するシステムの応答時間に依存して変化する．

5.5.2　評価方法

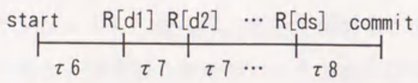
部分更新処理を発行するターミナル数（更新ターミナル数）を変化させて，部分更新処理と全数検索処理の完了率を求めた．さらに，部分更新処理で更新する



(a) 待ち行列モデル



(b) 部分更新処理



(c) 全数検索処理

図5.2　評価モデル
Figure 5.2　Evaluation model

表5.1　評価パラメータ
Table 5.1　Simulation parameters

パラメータ	値
データベースサイズ（s）	100 ブロック
更新ターミナル数（u）	2，4，6，8，10
参照ターミナル数	1
データベース参照率（α）	0.1
参照データ更新率（β）	0.1，0.5，1.0
遅延時間（$\tau 1$ ～$\tau 8$）	50 ユニット
シミュレーション時間	50,000 ユニット

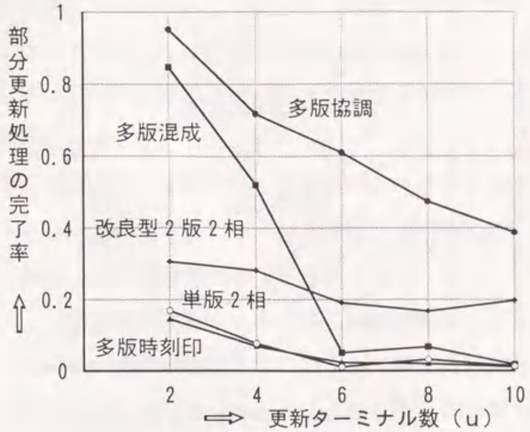
ブロック数を変化させて，部分更新処理と全数検索処理の完了率を求めた．評価に用いたパラメータを表5.1に，評価結果を図5.3と図5.4に示す．ここで，トランザクションの完了率とは，シミュレーション期間中にコミットするトランザクション数の観測値とアクセスするブロックの競合が全く生じないと仮定した場合の理想値の比である．また，表5.1における部分更新処理のデータベース参照率αとはnとmの比である．図中の各点は，評価システムが定常状態に達してから適当にサンプリングした複数の測定期間についての平均値を示している．

5.5.3 考察

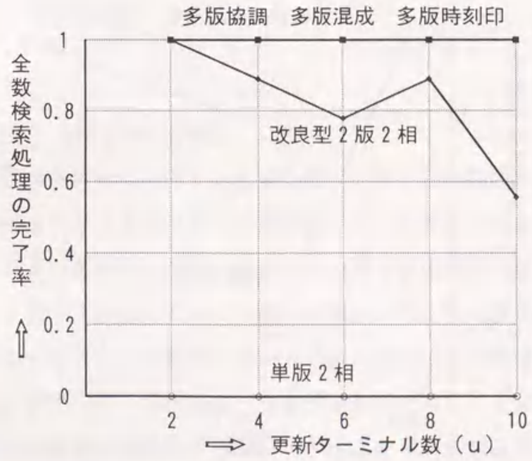
(1) 更新ターミナル数の影響

部分更新処理の完了率は，更新ターミナル数が少ない状態では主に部分更新処理と全数検索処理の干渉の影響を受けて劣化し，更新ターミナル数が多い状態ではさらに部分更新処理間の干渉の影響を受けて劣化する．また，全数検索処理の完了率は部分更新処理と全数検索処理の干渉の影響を受けて劣化し，更新ターミナル数が多いほどその干渉は強くなる．

図5.3は，多版協調方式が2種類の干渉の問題を解決した方式であるため，更新ターミナル数にかかわらず従来方式より常に良好な部分更新処理と全数検索処理の完了率を提供することを示している．多版混成方式は，部分更新処理と全数検索処理が干渉しないため，更新ターミナル数が少ない状態では比較的良好であるが，部分更新処理間の干渉が強いため，更新ターミナル数が多くなると部分更新処理の完了率が悪化する．改良型2版2相ロック方式は，部分更新処理と全数検索処理の干渉のために部分更新処理と全数検索処理の完了率がともにあまり良くないが，部分更新処理間の干渉が少ないため，更新ターミナル数が増加しても部分更新処理の完了率はあまり変化しない．多版時刻印方式は，2種類の干渉の影響により部分更新処理の完了率が多版並行処理制御の中では最悪であるが，全数検索処理の完了率は良好である．

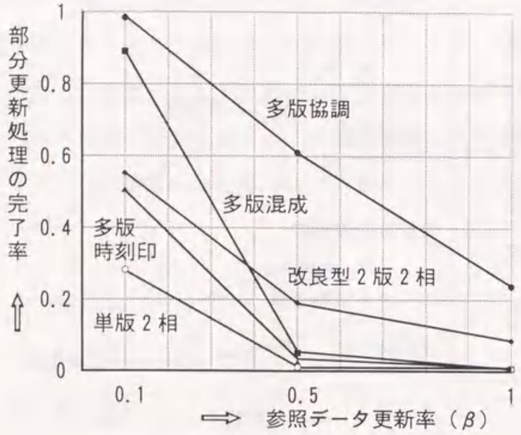


(a) 部分更新処理の完了率

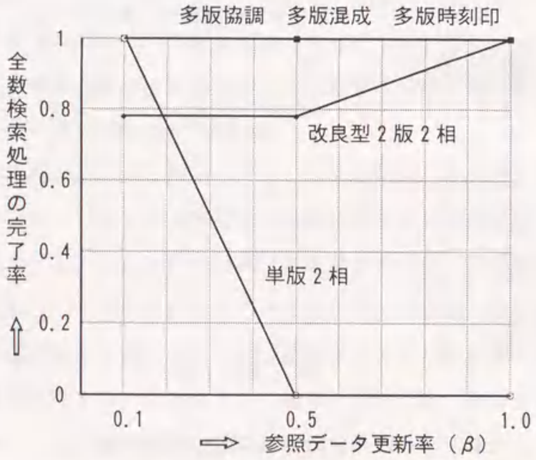


(b) 全数検索処理の完了率

図5.3　更新ターミナル数対完了率（α=0.1，β=0.5）
Figure 5.3　Number of terminals vs completion ratios



(a) 部分更新処理の完了率



(b) 全数検索処理の完了率

図5.4　参照データ更新率対完了率（u=6，α=0.1）
Figure 5.4　Update ratio vs completion ratios

(2) 参照データ更新率の影響

部分更新処理の参照データ更新率の増加は，複数の部分更新処理が同一データの更新で競合する頻度を高め，部分更新処理の並行性を劣化させる．また，部分更新処理と全数検索処理が干渉し合う並行処理制御方式では，全数検索処理との並行性にも影響を与える．

図5.4は，多版協調方式が参照データ更新率にかかわらず常に従来方式より優れた性能を与えること，特に参照データ更新率が小さい時に部分更新処理と全数検索処理の両方について1に近い完了率を提供することを示している．多版混成方式は，参照データ更新率が十分に小さい時，多版協調方式とほぼ同程度の性能を与えるが，参照データ更新率が増加するにつれて部分更新処理の完了率が極めて悪くなる．部分更新処理と全数検索処理が干渉し合う改良型2版2相ロック方式では，参照データ更新率の増加により部分更新処理の完了率が劣化すると，逆に全数検索処理の完了率が向上する傾向が現れている．

5.6　まとめ

本章では，定型処理の代表例である部分更新処理と非定型処理の代表例である全数検索処理が混在する環境に適した多版並行処理制御方式である多版協調方式を提案した．従来の多版並行処理制御方式は，部分更新処理の並行処理において不要なアボートを引き起こしていた．多版協調方式では，2版2相ロック方式の改良により部分更新処理の不要なアボートを防止することで部分更新処理間の並行性を高めた．また，部分更新処理と全数検索処理を相互に全く干渉させずに行える多版混成方式の手法を有効と考え，この手法を用いて部分更新処理と全数検索処理の並行性を高めた．多版協調方式の効果を定量的に明らかにするためシミュレーション評価を行った結果，並行処理される部分更新処理数にかかわらず，部分更新処理の完了率と全数検索処理の完了率の両方について，従来方式を大幅に上回る性能を有することが明らかになった．さらに，複数の部分更新処理が同一データの更新で競合する頻度が高い状態においても，従来方式よりも性能が良

好であることも明らかとなった．

以上の結果，定型処理と非定型処理をオンライン・リアルタイムで同時に実行した場合に，データベース処理の高速化を目的とした第１及び第２の両方のアプローチで残されていた共通の問題であるデータネックを解決できる見通しが得られた．

6　結論

6.1　本研究のまとめ

本論文は，大規模なリレーショナルデータベースの高速処理方式に関するこれまでの研究成果を述べた．

この研究の背景には，リレーショナルデータベースの言語ＳＱＬがＩＳＯとＪＩＳで規格化され，大規模なデータベース処理にも利用されるようになってきたことがある．リレーショナルデータベースの歴史は１９７０年にまでさかのぼるが，当初から処理の高速化が大きな技術的課題となっていた．この課題を解決するためにこれまでに各種の方式が研究・開発されてはきたが，大規模なデータベースに対して十分に適用できるものはほとんどなかった．

本研究では，データベース処理を定型処理と非定型処理の２種類に分類し，従来の処理方式では処理時間の点からオンライン処理が事実上不可能であった大規模データベースの非定型処理の高速化技術の開発を目標とした．この目標を実現するために，従来のリレーショナル・データベース管理システムにおいて性能上のボトルネックとなっていたＣＰＵネック，メモリネック，入出力ネック，およびデータネックを解消することを目指した．本研究では，まずＣＰＵネックと入出力ネックを解消する第１のアプローチとして，通常のデータベース管理システムが搭載された汎用計算機システムにサーチ処理とソート処理のための専用プロセッサを付加する方式を提案し，実用システムとして実現した．この種の専用プロセッサに関する研究例は少なくないが，大量のデータを処理することを前提にシステムの全体，特に汎用計算機上のソフトウェアまでをも含めた研究例はこれまでにほとんどなく，本研究の大きな特徴となっている．続いてＣＰＵネックを解消する第２のアプローチとして，複数の汎用マイクロプロセッサによる並列処理を検討した．その理由は，専用プロセッサのみによるＣＰＵネックの解消には限界があることと，今後のマイクロプロセッサの技術動向を考慮すると汎用プロ

セッサの並列処理の将来性が高いためである．ここでは全資源共有型のマルチプロセッサシステムにおいて，各プロセッサを効率よく働かせるための負荷配分方式を提案し，プロトタイプを作成することによりその効果を実証した．最後に第1，第2のアプローチで共通して残されたデータネックを解消することを目的として多版管理による並行処理制御を検討した．ここでは特に，定型処理と非定型処理を並行して実行させた場合のデータネックを具体的な問題として設定し，これを解決するための新しいアルゴリズムを提案し，シミュレーション評価によってその有効性を検証した．本論文では，以上の研究成果を以下の5章に分けて述べた．

第1章では，研究の背景，従来の高速処理方式の概観，本研究の方針について述べた．従来技術の中では，ＣＰＵネックを解消することを目的とした専用プロセッサと並列処理，入出力ネックを解消することを目的としたディスク入出力の並列化とインテリジェント化，データネックを解消することを目的としたデータの多版管理の技術に着目した．本研究では，第1のアプローチである専用プロセッサについては大規模データベースへの適用，第2のアプローチである並列処理についてはプロセッサの利用の効率向上，さらに両者の共通の課題であるデータネックに関してはトランザクションの並行性向上を目標とした．

第2章では，第1のアプローチの具体的な実現手法として，サーチ処理とソート処理を専用プロセッサで実行する付加型のデータベースプロセッサ方式を提案した．この方式は，汎用計算機で問題になるＣＰＵネックと入出力ネックの解消を図るものである．本文では，専用プロセッサの設計条件を示した後，システムのハードウェア及びソフトウェア構成，結合処理を実行するための3フェイズジョイン法について述べ，試作したシステムの性能評価結果を示した．この付加型のデータベースプロセッサ方式による性能向上効果は極めて大きく，従来のソフトウェアと比較して最高 100倍以上の高速化を実現できたことを確認した．

第3章では，第2章で提案した付加型のデータベースプロセッサ方式における汎用計算機上のソフトウェアによる最適化手法を提案した．具体的には，利用者が専用プロセッサを意識することなく利用者プログラムを記述できるようにするため，データベース管理システムが機能判定と性能判定の2段階で自動的に最適化するものである．特に重要な性能判定部では，ほとんどコストをかけずに取得

できる情報のみを用いて入出力回数を推定することにより，十分実用的に満足できる最適化が可能なことを示した．

第4章では，第2のアプローチを実現する上で重要になる並列プロセッサに対するデータ配分コストを削減する動的な負荷配分量決定法を提案した．提案した手法は，処理の進行とともに負荷の配分単位を動的に変えていくことにより，プロセッサのアイドル時間をほとんど増加させることなく負荷配分に要する時間を削減するものである．プロトタイプ上での性能評価によって，従来方式よりも高い性能を安定的に得られ，かつ8プロセッサ程度まではプロセッサ数に性能がほぼ比例することを確認した．

第5章では，残されたデータネックを解消することを目的として，定型処理の代表である部分更新処理と非定型処理の代表である全数検索処理の並行性が高い多版並行処理制御方式を提案した．具体的には，従来からある2版2相ロック方式を改良し，さらに多版時刻印方式と混成させることにより，定型処理と非定型処理間の干渉を除去するとともに，定型処理間の干渉を削減するものである．シミュレーション評価によって，並行処理を行う上での制約条件である直列可能性を満足し，かつ定型処理と非定型処理双方の処理の完了率が大幅に向上することを示した．

6.2　今後の研究課題

本論文で述べた方式のうち，第2章及び第3章で述べた専用プロセッサ方式は既に商用化され，数十の実用システムで稼働しており，利用者から高い評価を受けている．典型的な利用形態は，時々刻々発生するデータが蓄積されていくトラヒック情報データベースに対して，オンラインで統計情報を求める業務であり，従来求めることが不可能であった最新情報を専用プロセッサによって容易に求められるようになった．また，現在は求めた統計情報に誤差が含まれることが許容されているため，データベースへのデータの追加処理と検索処理は並行処理制御なしで実行させているが，第5章で述べた並行処理制御方式を導入することにより，正確な統計情報を求めることが原理的には可能になった．

一方，この専用プロセッサ方式はスケーラブルでない，即ち段階的にシステムを拡張させていくのに適していない．また，フォールトトレラント化，即ちプロセッサ故障やシステム保守によるサービスの中断をなくすことが困難である．これらの問題を解決するためには，第４章で述べた並列処理方式の導入が有効であり，最近のマルチプロセッサの急速な性能向上と低価格化を考慮すると，その有効性はさらに高まっていくと考えられる．従って将来的には，本論文で述べた第１のアプローチと第２のアプローチの統合形態，即ちサーチ処理とソート処理は専用プロセッサ，その他の処理は複数汎用マイクロプロセッサの並列処理で実現する形態が有望である．また，最近のメモリの急激な大容量化と低価格化を背景に，データベースの格納媒体が磁気ディスクから主記憶装置に置き換えられるシステムも増加していくと考えられ，主記憶装置上でのデータベースの高速処理技術も必要になってきている．一方，データベースの利用分野は従来のような事務処理だけでなく，通信ネットワークやトラフィックの制御等信頼性，拡張性，さらには実時間性の要求が厳しい分野にも広がってきている．従って，本研究に関連する今後の研究課題としては，①スケーラブルなデータベース処理アーキテクチャ，②フォールトトレラントなシステム構成法，及び③主記憶データベースの実現法が重要である．

謝辞

本研究を遂行するのに際し，数多くの方々の御世話になった．全ての方々を網羅することはできないが，特に御世話になった方々を以下に列記させて頂く．これらの方々に心より深く感謝いたします．

名古屋大学杉江　昇教授には本論文の作成にあたって丁寧なご指導と激励を頂いた．また，論文の審査に関わる手続きなどでも大変御世話を頂いた．

名古屋大学伊藤正美教授には本論文執筆のきっかけを与えて頂いた．また，著者が名古屋大学学生時代に暖かなご指導を頂き，情報処理分野の研究を進める上での基礎作りをして頂いた．

名古屋大学稲垣康善教授には本論文を丁寧に読んで頂き，建設的なご助言とご指導を頂いた．

名古屋大学渡辺豊英助教授には本論文の草稿段階から非常に注意深く読んで頂き，数多くのご助言を頂いた．

豊橋技術科学大学伊藤宏司教授には著者が名古屋大学学生時代に卒業論文の指導教官として親身なご指導を頂き，著者がコンピュータのソフトウェアを実務として扱う最初の機会を与えて頂いた．

大阪大学橋本昭洋教授及び神奈川大学村上国夫教授にはデータベース専用プロセッサ方式の研究開始時に，著者の上司としてご指導頂いた．両教授のご指導なくしては，本研究は成り立ち得なかったものである．

東京大学喜連川優助教授にはデータベース専用プロセッサ方式に関して熱心な討論とご助言を頂いた．筑波大学清木　康助教授には並列処理における動的負荷配分方式に関して熱心に討論して頂いた．大阪大学西尾章次郎助教授には多版管理による並行処理制御方式に関して建設的なご意見を頂いた．

京都大学上林彌彦教授及び図書館情報大学増永良文教授にはデータベース処理全般に渡って幅広い観点からのご助言を頂いた．

ＮＴＴ松永俊雄博士，拝原正人氏，松田晃一氏には歴代の研究部長として本研

究を遂行する機会を与えて下さった．また，ＮＴＴ鈴木健司氏，福岡秀樹氏及び寺中勝美氏には著者の元の上司及び論文の共著者として貴重なご指導を頂いた．

ＮＴＴ石野福彌博士及び伊土誠一氏には著者の現在の上司として本論文作成のための暖かいご配慮と激励を頂いた．

ＮＴＴ速水治夫氏，中村敏夫氏，武田英昭氏，佐藤哲司氏，芳西　崇氏，板倉一郎氏，片岡良治氏，平野泰宏氏には著者の論文の共著者として，実験システムの作成，評価データの収集等で大変な御世話を頂いた．個々のお名前は省かせて頂くが，ＮＴＴ情報通信網研究所の研究員の方々にも数多くの討論を頂いた．また，データベース専用プロセッサ方式の実用化に際しては協力メーカの方々の多大なご協力を頂いた．

最後に，学会・研究会等を通じて数多くの先輩諸氏の方々に激励と暖かいご助言を頂いたことを感謝いたします．

参考文献

[1] Codd, E.F.:"A Relational Model of Data for Large Shared Data Banks", Comm.ACM, Vol.13, No.6, pp.377-387 (June 1970).

[2] ISO:"Information Processing Systems - Database Language SQL", International Standard, ISO 9075 (June 1987).

[3] JIS:"データベース言語ＳＱＬ", 日本工業規格 JIS X 3005 (Nov. 1987).

[4] Schuster, S.A., Nguyen, H.B., et al.:"RAP.2 - An Associative Processor for Databases and Its Applications", IEEE Trans.Comput., Vol.C-28, No.6, pp.446-458 (June 1979).

[5] Su, S.Y.W., Nguyen, L.H., et al.:"The Architectural Features and Implementation Techniques of the Multicell CASSM", IEEE Trans. Comput., Vol.C-28, No.6, pp.430-445 (June 1979).

[6] Banerjee, J., Hsiao, D.K. and Kannan, K.:"DBC - A Database Computer for Very Large Databases", IEEE Trans.Comput., Vol.C-28, No.6, pp.414-429 (June 1979).

[7] DeWitt, D.J.:"DIRECT - A Multiprocessor Organization for Supporting Relational Database Management", IEEE Trans.Comput., Vol.C-28, No.6, pp.395-406 (June 1979).

[8] 植村俊亮，弓場敏嗣，他:"磁気バブルデータベースマシン", パターン情報処理システム調査・研究報告，PIPS-R-No.28,29，電総研 (March 1981).

[9] Comer, D.:"The Ubiquitous B-Tree", ACM Computing Surveys, Vol.11, No.2, pp.121-137 (June 1979).

[10] Knuth, D.:"The Art of Computer Programming", Addison-Wesley (1973).

[11] Miranker, G., Tang, L. and Wong, C.K.:"A 'Zero-Time' VLSI Sorter", IBM J.Res.Develop., Vol.27, No.2, pp.140-148 (March 1983).

[12] 喜連川優, 伏見信也, 他:"パイプラインマージソータの構成", 電子通信学会論文誌, Vol.J-66D, No.3, pp.332-339 (March 1983).

[13] 安藤隆朗, 藤森敬悟, 他:"リレーショナルデータベースプロセッサＧＲＥＯの概要", 情報処理学会第39回全国大会講演論文集, 4N-7 (Sept. 1989).

[14] 小島啓二, 鳥居俊一, 吉住誠一:"ベクトル型データベースプロセッサＩＤＰ", 情報処理学会論文誌, Vol.31, No.1, pp.163-173 (Jan. 1990).

[15] Kung, H.T. and Lehman, P.T.:"Systolic (VLSI) Arrays for Relational Database Operations", Proc.of ACM-SIGMOD Int.Conf., pp.105-116 (May 1980).

[16] Bitton, D., Boral, H., et al.:"Parallel Algorithms for the Execution of Relational Database Operations", ACM Trans.Database Syst., Vol.8, No.3, pp.324-353 (Sept. 1983).

[17] Richardson, J.P., Lu, H. and Mikkilineni, K.:"Design and Evaluation of Parallel Pipelined Join Algorithms", Proc.of 1987 ACM-SIGMOD Int. Conf., pp.399-409 (May 1987).

[18] DeWitt, D.J., Gerber, R.H., et al.:"GAMMA - A High Performance Dataflow Database Machine", Univ.Wisconsin Tech.Report, CSTR#635 (March 1986).

[19] Teradata Corp.:"DBC/1012 Data Base Computer System - Introduction" (1986).

[20] DeWitt, D.J., Katz, R.H., et al:"Implementation Techniques for Main Memory Database Systems", Proc.of 1984 ACM-SIGMOD Int.Conf., pp.1-8 (June 1984).

[21] McGregor, D.R., Thomson, R.G. and Dawson, W.N.:"High Performance Hardware for Database Systems", Systems for Large Data Bases, pp.103-116, North-Holland (1976).

[22] 小倉武, 山田慎一郎, 他:"20kb CAM(Content Addressable Memory) LSI", 電子通信学会計算機システム研究会資料, CPSY87-33 (Jan. 1988).

[23] Patterson, D.A., Gibson, G. and Katz, R.H.:"A Case for Redundant Arrays of Inexpensive Disks (RAID)", Proc.of 1988 ACM-SIGMOD Int. Conf., pp.109-116 (May 1988).

[24] Babb, E.:"Implementing a Relational Database by Means of Specialized Hardware", ACM Trans.Database Syst., Vol.4, No.1, pp.1-29 (March 1979).

[25] Hollaar, L.A.:"Text Retrieval Computers", IEEE Computer, Vol.12, No.3, pp.40-50 (March 1979).

[26] Gray, J.:"The Transaction Concept: Virtues and Limitations", Proc. of 7th Int.Conf.on Very Large Databases, pp.144-154 (1981).

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

[28] 大森匡, 喜連川優, 田中英彦:"並列ディスクデータベースマシンにおける大量データアクセストランザクションの並行制御方式", 電子情報通信学会論文誌, Vol.J73-D-I, No.1, pp.37-46 (Jan. 1990).

[29] Papadimitriou, C.H. and Kanellakis, P.C.:"On Concurrecy Control by Multiple Versions", ACM Trans.Database Syst., Vol.9, No.1, pp.89-99 (March 1984).

[30] Hadzilacos, T. and Papadimitriou, C.H.:"Algorithmic Aspects of Multiversion Concurrency Control", Proc.of 4th SIGACT-SIGMOD, pp.96-104 (1985).

[31] Britton Lee, Inc.:"The Intelligent Database Machine - Product Description" (1984).

[32] 小柳津育郎, 塩川鎮雄, 他:"ＤＩＰＳ－１１／５Ｅシリーズの実用化", ＮＴＴ研究実用化報告, Vol.36, No.1, pp.49-56 (Jan. 1987).

[33] 矢沢良一, 平野正則, 他:"ＤＩＰＳ－Ｖ３０Ｅのハードウェア構成", ＮＴＴ研究実用化報告, Vol.37, No.9, pp.523-532 (Sept. 1988).

[34] 佐藤哲司, 武田英昭, 津田伸生:"大容量データベース処理に適したソータ構成法", 情報処理学会論文誌, Vol.31, No.11, pp.1653-1660 (Nov. 1990).

[35] Blasgen, M.W. and Eswaran, K.P.:"Storage and Access in Relational

Data Bases", IBM Syst.J., No.4, pp.363-377 (1977).

[36] Kitsuregawa,M., Tanaka,M. and Moto-oka,T.:"Application of Hash to Data Base Machine and Its Architecture", New Generation Computing, Vol.1, No.1, pp.62-74 (1983).

[37] Lum,V.Y., Yuen,P.S.T. and Dodd,M.:"Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files", Comm.ACM, Vol.14, No.4, pp.228-239 (April 1971).

[38] Knott, G.D.:"Hashing Functions", Comput.J., Vol.18, No.3, pp.265-287 (1975).

[39] 武田英昭, 佐藤哲司, 中村敏夫, 速水治夫:"関係演算高速化プロセッサ", 情報処理学会論文誌, Vol.31, No.8, pp.1230-1241 (Aug. 1990).

[40] 伊藤文英, 島川和典, 他:"可変長レコード用関係データベース処理エンジンの試作とソート処理性能の評価", 情報処理学会論文誌, Vol.30, No.8, pp.523-532 (Sept. 1988).

[41] Bitton,D., DeWitt,D.J. and Turbyfill,C.:"Benchmarking Database Systems: A Systematic Approach", Proc.of 9th Int.Conf.on Very Large Databases, pp.8-19 (Oct. 1983).

[42] DeWitt,D.J., et al.:"A Single User Evaluation of the GAMMA Database Machine", Proc.of 5th Int.Workshop on Database Machines, pp.43-59 (Oct. 1987).

[43] Simon,E.:"Update to December 1983 'DeWitt' Benchmark", Britton Lee, Inc. (March 1985).

[44] Selinger,P.G., Astrahan,M.M. and Eswaran,K.P.:"Storage and Access in Relational Data Bases", IBM Syst.J., Vol.14, No.4, pp.363-377 (1977).

[45] Wang,Y-T. and Morris,R.J.T.:"Load Sharing in Distributed Systems", IEEE Trans.Comput., Vol.C-34, No.3, pp.204-217 (March 1985).

[46] Sequent Computer Systems:"Symmetry Technical Summary", P/N:1003-44447 Rev.A (1988).

[47] Bayer,R., Heller,H. and Reiser,A.:"Parallelism and Recovery in

Database Systems", ACM Trans.Database Syst., Vol.5, No.2, pp.139-156 (June 1980).

[48] Stearns,R.E. and Rosenkrantz,D.J.:"Distributed Database Concurrency Controls Using Before-Values", Proc.of ACM-SIGMOD Int.Conf., pp.74-83 (1981).

[49] Bernstein,P.A. and Goodman,N.:"Multiversion Concurrency Control - Theory and Algorithms", ACM Trans.Database Syst., Vol.8, No.4, pp.465-483 (Dec.1983).

[50] Chan,A. and Gray,R.:"Implementing Distributed Read-Only Transactions",IEEE Trans.Software Eng., Vol.SE-11, No.2, pp.205-212 (April 1985).

《本論文に関連する著者の発表文献》

〔データベース専用プロセッサ関連〕

[51] "RINDA - A Relational Database Processor for Non-indexed Queries", Inoue,U., Hayami,H., Fukuoka,H. and Suzuki,K., Proc.of Int.Symp.on Database Systems for Advanced Applications (DASFAA'89), pp.382-386 (April 1989).

[52] "データベースプロセッサＲＩＮＤＡの設計と実現", 井上潮, 速水治夫, 福岡秀樹, 鈴木健司, 松永俊雄, 情報処理学会論文誌, Vol.31, No.3, pp.373-380 (March 1990).

[53] "Acceleration of Join Queries by a Relational Database Processor, RINDA", Satoh,T., Takeda,H., Inoue,U. and Fukuoka,H., Proc.of 2nd Int.Symp.on Database Systems for Advanced Applications (DASFAA'91), pp.243-248 (April 1991).

[54] "データベースプロセッサＲＩＮＤＡの結合演算処理機構の構成と評価", 佐藤哲司, 武田英昭, 井上潮, 福岡秀樹, 情報処理学会論文誌, Vol.32, No.8, pp.1006-1013 (Aug. 1991).

[55] "RINDA: A Relational Database Processor with Hardware Specialized for Searching and Sorting", Inoue,U., Satoh,T., Hayami,H., Takeda, H., Nakamura,T. and Fukuoka,H., IEEE Micro, Vol.11, No.12, pp.61-70 (Dec. 1991).

[56] "データベースプロセッサＲＩＮＤＡ", 井上潮, 佐藤哲司, 速水治夫, 情報処理（データベースマシン特集号：92年12月掲載予定）.

[57] "RINDA: A Relational Database Processor for Large Databases", Satoh,T. and Inoue,U., Emerging Trends in Database and Knowledge-base MachinesDatabase (Abdelguerfi,M. and Lavington,S.(eds)), IEEE-CS Press(93年発行予定).

[58] "データベースプロセッサＲＩＮＤＡのアーキテクチャ", 井上潮, 速水治夫, 福岡秀樹, 鈴木健司, 情報処理学会第37回全国大会講演論文集, 5Q-4 (Sept. 1988).

[59] "リレーショナルデータベースプロセッサＲＩＮＤＡのアーキテクチャ", 速水治夫, 井上潮, 福岡秀樹, 鈴木健司, 情報処理学会計算機アーキテクチャ研究会資料, 88-ARC-73-12 (Oct. 1988).

[60] "データベースプロセッサＲＩＮＤＡの検索処理速度向上効果", 黒岩淳一, 板倉一郎, 井上潮, 福岡秀樹, 情報処理学会第38回全国大会講演論文集, 3Q-3 (March 1989).

[61] "データベースプロセッサＲＩＮＤＡの制御プログラム", 井上潮, 中村仁之輔, 芳西崇, 片岡良治, ＮＴＴ　Ｒ＆Ｄ, Vol.38, No.8, pp.869-876 (Aug. 1989).

[62] "データベースプロセッサＲＩＮＤＡの制御方式", 井上潮, 芳西崇, 中村仁之輔, 中村敏夫, 電子情報通信学会データ工学研究会資料, DE89-41 (Dec. 1989).

〔最適化処理方式関連〕

[63] "データベースプロセッサＲＩＮＤＡにおける問合せ処理のアクセスパス決定方式", 芳西崇, 板倉一郎, 中村敏夫, 井上潮, 情報処理学会論文誌, Vol.32, No.11, pp.1412-1422 (Nov. 1991).

[64] "データベースプロセッサＲＩＮＤＡの問合せ処理方式", 芳西崇, 中村仁之輔, 中村敏夫, 井上潮, 情報処理学会第37回全国大会講演論文集, 5Q-7 (Sept. 1988).

[65] "データベースプロセッサＲＩＮＤＡのデータベースアクセス方式", 板倉一郎, 中村仁之輔, 井上潮, 情報処理学会第37回全国大会講演論文集, 5Q-9 (Sept. 1988).

〔動的負荷配分方式関連〕

[66] "Load Balancing Algorithms for Parallel Database Processing on Shared Memory Multiprocessors", Hirano,Y., Satoh,T., Inoue,U. and Teranaka,K., Proc.of 1st Int.Conf.on Parallel and Distributed Information Systems (PDIS'91), pp.210-217 (Dec. 1991).

[67] "資源共有型マルチプロセッサにおけるデータベース処理の動的負荷配分

法", 平野泰宏, 佐藤哲司, 井上潮, 寺中勝美, 電子情報通信学会論文誌, Vol.J75-D-I, No.3, pp.152-159 (March 1992).

[68] "Design and Implementation of a Parallel Database Processing on a Shared Memory Multiprocessor System", Satoh,T., Hirano,Y., Honishi, T. and Inoue,U., Proc.of 2nd Far-East Workshop on Future Database Systems (FEWS'92), pp.337-346 (April 1992).

[69] "資源共有型マルチプロセッサにおけるデータベース処理の動的負荷配分法", 平野泰宏, 佐藤哲司, 井上潮, 寺中勝美, 情報処理学会データベースシステム研究会資料, 91-DB-82-1 (March 1991).

[70] "データベース並列処理技術", 井上潮, 佐藤哲司, 片岡良治, 平野泰宏, ＮＴＴ　Ｒ＆Ｄ（データベース特集号：92年12月掲載予定）

〔並行処理制御方式関連〕

[71] "A Multiversion Concurrency Control Algorithm for Concurrent Execution of Partial Update and Bulk Retrieval Transactions", Kataoka,R., Satoh,T. and Inoue,U., Proc.of 10th Int.Phoenix Conf. on Computers and Communications (IPCCC'91), pp.130-136 (March 1991).

[72] "部分更新と全数検索の混在処理に適した多版並行処理制御方式", 片岡良治, 佐藤哲司, 井上潮, 電子情報通信学会論文誌, Vol.J74-D-I, No.3, pp.224-231 (March 1991).

[73] "多版同時実行制御に関する一考察",片岡良治, 武田英昭, 佐藤哲司, 井上潮, 情報処理学会第39回全国大会講演論文集, 5M-6 (Oct. 1989).

〔その他で関連するもの〕

[74] "A Relational Database Machine for Very Large Information Retrieval Systems", Inoue,U. and Kawazu,S., Database Machines (Sood,A.K. and Qureshi,A.H.(eds)), NATO-ASI Series, Vol.F24, pp.183-201, Springer-Verlag (1986).

[75] "An Index Structure for Parallel Database Processing", Honishi,T., Satoh,T. and Inoue,U., Proc.of 2nd Int.Workshop on Research Issues on Data Engineering: Transaction and Query Processing (RIDE-TQP), pp.224-225 (Feb. 1992).

[76] "大規模ＲＤＢ向きデータベースマシンアーキテクチャに関する一考察", 佐藤清実, 北村正, 中村敏夫, 井上潮, 武田英昭, 情報処理学会データベースシステム研究会資料, 83-DB-35-3 (May 1983).

[77] "情報提供サービスに適用可能な超大規模リレーショナル・データベースマシン", 井上潮, 北村正, 速水治夫, 中村敏夫, 情報処理学会データベースシステム研究会資料, 85-DB-47-5 (May 1985).

[78] "リレーションのデータベースマシン向きのクラスタリング方式に関する考察", 板倉一郎, 井上潮, 情報処理学会データベースシステム研究会資料, 85-DB-47-6 (May 1985).

[79] "サーチプロセッサの設計と評価", 速水治夫, 井上潮, 情報処理学会データベースシステム研究会資料, 86-DB-51-2 (Jan. 1986).

[80] "付加プロセッサ方式によるリレーショナルデータベースマシンアーキテクチャ", 北村正, 速水治夫, 中村敏夫, 井上潮, ＮＴＴ研究実用化報告, Vol.36, No.5, pp.663-671 (May 1987).

[81] "メモリ共有型マルチプロセッサにおけるオンラインとバッチ処理の混在した実行制御法", 佐藤哲司, 片岡良治, 平野泰宏, 井上潮, 情報処理学会データベースシステム研究会資料, 92-DBS-89-18 (July 1992).