データベースシステム 89−18
（１９９２．７．２４）

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
ＮＴＴ情報通信網研究所

ネットワーク管理等の２４時間サービスを提供するデータベース応用では，，従来バッチで処理していた統計処理等をオンライン処理を中断することなく実行できることが望まれている．本稿では，メモリ共有型マルチプロセッサを用いることで，オンライン処理性能を低下させることなくバッチを混在実行することを狙いとした並列処理アーキテクチャ，共有メモリ上のバッファ管理法，バッチ問合せの並列実行法を述べる．トランザクションの持つアクセスパタンを反映した優先度を用いたバッファ管理の有効性と，共有資源の割当て量によって実行速度を制御するトップダウン実行制御法の有効性をシミュレーションとプロトタイピングによって確認した．

Concurrent Execution of Online and Batch Transactions on Shared Memory Multiprocessors

Tetsuji Satoh, Ryoji Kataoka, Yasuhiro Hirano and Ushio Inoue
NTT Network Information Systems Laboratories

In non-stop online applications, batch transactions such as statistical analyses are required to be executed concurrently with online transactions. This paper discusses on a parallel processing architecture and a priority-based buffer management algorithm which enables concurrent execution of online and batch transactions. In the proposed algorithm, priorities of buffers are decided by both data-types and access-patterns of transactions. A top-down execution control with resource allocation balances the execution speeds of transactions on a shared memory multiprocessor system. The effectiveness of them are evaluated by simulation and prototyping.

1 まえがき

関係データベースは，金融システム，チケット予約システム，在庫管理システム、人事管理システムなど多くのビジネスアプリケーションで利用され，大規模なデータベースシステムが実現されている．これらのシステムの多くでは，オンライントランザクションが更新したデータベースに対して，バッチトランザクションが統計処理を行ない販売戦略を作成するような使い方が行なわれている．一般に，オンライントランザクションはインデックスを活用して表（テーブル）の特定の行(タプル)を検索・更新するのに対して，バッチトランザクションは表全体のフルスキャンやジョイン等を必要とし処理時間が長大化する傾向にある．このため，従来のデータベース管理システム（ＤＢＭＳ）では，オンライントランザクションとバッチトランザクションを走行させる時間帯を分ける使い方，例えば，昼間はオンライン，夜間はバッチ，をしていた．金融システムを例にすると，個々の個人口座の預金残高は，預金・引出し操作に伴ってオンラインで更新されなければならないし，支店毎の総預金残高の変動は日単位で集計されなければならない．前者はオンライントランザクションの，後者はバッチトランザクションの典型例であり，それぞ昼間のオンラインと夜間のバッチに分離して処理していた．

しかし，24時間サービスを提供する大規模なネットワーク管理や世界的規模での取引を扱うシステムでは，１日２４時間サービスを停止できないために，上記のような時間帯を分けた使い方が困難となる．また，統計処理を行なうバッチトランザクションを，オンライントランザクションと並行して実行できれば，統計結果の精度(鮮度)が向上し，より有効な戦略立案に利用できる利点が得られる．このようなオンライントランザクションとバッチトランザクションの混在実行は，専用データベースマシン[Nech88][Inou91]を用いてバッチを高速化することで疑似的に実現できるが，このような最新のハードウェア技術を用いてもバッチの処理時間とオンラインの処理時間とは２桁程度の差があり，オンライントランザクションに影響を与えずにバッチトランザクションを実行することは困難であった．なお，オンラインと混在できない複雑な統計処理をオフラインで実行するためにオンライン処理中にデータベースの複製をとる操作は，データベースのスキャンを伴うため本稿で示すバッチトランザクションの範疇に含まれる．

本稿では，オンライン系処理の性能を低下させることなくバッチを混在実行することを狙いとする並列処理アーキテクチャについて述べる．マルチプロセッサによる並列処理では，複数のトランザクションを異なるプロセッサで同時実行する，即ちトランザクション間並列によるオンライントランザクションのスループット向上と，負荷が重いバッチトランザクションを複数のタスクに分解して並列処理する，即ちトランザクション内並列によるレスポンス時間短縮の両方を実現できる．ここでは，通信コストが小さく比較的簡単な同期機構で並列処理を実現できるメモリ共有型マルチプロセッサを前提とした並列処理アーキテクチャ，共有メモリ上のバッファ管理法について述べる．

以下，第２章ではオンラインとバッチを混在実行するための課題を明らかにし解決の方針を示す．第３章では，メモリ共有型マルチプロセッサによる並列処理方法，特にプロセッサやバッファ等の共有資源の割り当て量を変えることで実行速度を制御するトップダウン実行制御法について述べる．第４章では，２章で示したオンラインとバッチの混在実行に適したバッファ管理法として，トランザクションの持つアクセスパタンを反映した優先度付きバッファ管理法を提案し，シミュレーション評価の結果を示す．第５章では，提案した優先度付きバッファ管理法とトップダウン実行制御法を適用したＤＢＭＳプロトタイプでのトランザクション内並列処理方法を示す．

2 混在実行の課題と解決の方針

2.1 混在実行のための条件

関係データベースの処理は，顧客対応の窓口業務に代表されるオンライントランザクションと，日報・月報作成のための統計処理に代表されるバッチトランザクションに分類できる．オンラインとバッチの処理特性を以下に示す．

オンライントランザクション：

オンラインの検索・更新は，複数ユーザが同時に実行できる高い並行性とユーザをほとんど待たせない高い応答性が要求される．このため，予め作成されたインデックスを用いて，処理対象とするデータ（ロー）を高速にアクセスして検索・更新するのが一般的である．TPCベンチマーク[Serl91]は典型的なオンライントランザクションであり，銀行の窓口業務をモデル化した４個の更新処理と１個の追加処理からなる．本稿では，最も単純なオンライン処理として，インデックスを用いたテーブルの１件検索を対象とする．この処理は，検索条件に指定したカラムに予め付与されたユニークインデックスを用いて目的とするローを直接アクセスして検索結果を出力する．このようなオンライン処理では，アクセス特性として検索するデータに80-20則に示されるような

アクセスの偏りを仮定できる．つまり，アクセスするインデックスとテーブルともに，80%の処理がデータベース中の20%の領域を頻繁にアクセスするといえる．

バッチトランザクション：
統計処理に代表されるバッチトランザクションは，テーブルのフルスキャンを伴う条件検索の繰り返しで実現される．検索条件は複雑で，ジョインやソート等の膨大な演算量を必要とする処理を含む場合もある．ここでは，データベース中の特定の表に対するスキャンを想定する．通常，データベースは固定サイズのページを格納の単位としていることから，ページ番号が連続する複数(数十から数万)ページに対するページ読み込みと条件検索の繰り返し処理である．バッチトランザクションはアクセスの偏りが無いと仮定し，テーブルを格納している全ページを同一の確率でアクセスするものとする．ある種のバッチ，例えばソートやジョインを含む処理では，中間結果を格納しなければならない．この中間結果は，処理対象とするデータベースの大きさとそれまでの処理の選択率の積で容量が決まり，引き続く処理の過程で少なくとも１回は読み出され処理されるデータである．

オンライントランザクションは複数のユーザによって同時に処理されるが，バッチトランザクションは負荷が大きく処理時間が長いので同時に処理するバッチは高々1つに限定できるとする．これらの異なる２種類のトランザクションを単に混在実行したのでは，以下の理由により負荷の大きなバッチの影響でオンラインのスループットが低下したりレスポンスが悪化すると考えられる．

１）オンラインの実行はバッチに優先し，バッチの実行によってオンラインのスループットやレスポンスが低下しないことが望ましい．十分なプロセッサパワーが無いと，バッチの実行によってオンラインに割り当てることができる資源が不足し処理時間が長くなる．
２）主記憶容量より大きな容量を持つデータベースを扱うために，データベースの一部を主記憶上にバッファリングして処理する．バッチを混在実行することによってオンラインが頻繁にアクセスするページがバッファから追い出されると，オンラインは，再読み込みのために演算量が増大し，かつディスクアクセス待ちが生じるために応答性能が低下する．
３）データベース内の同一データをオンラインとバッチが同時に更新・検索しようとすると，データベースの一貫性を保持するための並行処理制御によっていずれかの処理が待たされる．単純な２相ロック法では，バッチがデータベース内の大量のデータを長時間ロックするためにオンラインの更新処理が実行を待たされる．

2.2 解決の方針

オンラインとバッチの混在実行で必要となる高い演算能力はマルチプロセッサによる並列処理で実現する．マルチプロセッサの構成には，メモリ共有型とメモリ非共有型とがある．メモリ非共有型は，比較的大規模なマルチプロセッサシステムを実現できるが，プロセッサ間を結合する通信路での遅延や競合・迂回が避けられないこと，プロセッサ間でデータベースや実行負荷を均等に配置することが難しい．特に，データベース処理では，データベースを均等に配置した場合でも，問合せ条件とデータの値に依存する選択率によって実行負荷が変動するために，実行前に負荷を均等に配置することはできない．以上の理由から，通信コストが小さく負荷の動的な再配置が容易なメモリ共有型マルチプロセッサを用いて並列処理を行なうこととした．

メモリ共有型マルチプロセッサでは，ディスクに格納されたデータベースの一部を共有メモリ上に設けられたバッファに読み込んで処理する．したがって，限られた容量のバッファを有効に利用するバッファ管理方法が性能を支配する重要な要因となる．特に，オンラインとバッチの混在処理では，バッチの実行によってオンラインが必要とするデータがバッファから追い出されるとオンラインの性能が低下することから，混在実行に適したバッファ管理方法が必要となる．

以上の観点から，（1）メモリ共有型マルチプロセッサにおける並列処理アーキテクチャ，（2）バッチの実行がオンラインの性能を低下させないように優先度を設けた共有バッファの管理法，（3）トップダウンに割り当てられた資源を用いたバッチ問合せの並列処理法について議論する．なお，オンライン処理による更新とバッチによる全数検索を同一のデータベースに対して混在処理するための並行処理制御法については，文献[Kata91]に述べられているため，本稿では議論しない．

3　並列処理アーキテクチャ

3.1 プロセス構成

トランザクション間並列とトランザクション内並列を実現するクライアントーサーバ型のプロセス構成を図１に示す．クライアントプロセスは，ユーザ対応のアプリケーションを走行させるプロセスでリモートサイトに配置することもできる．サーバは，複数の子プロセスによる並列処理ができる構成とし，次節に示すトップダウンな資源割り当てによってパラ

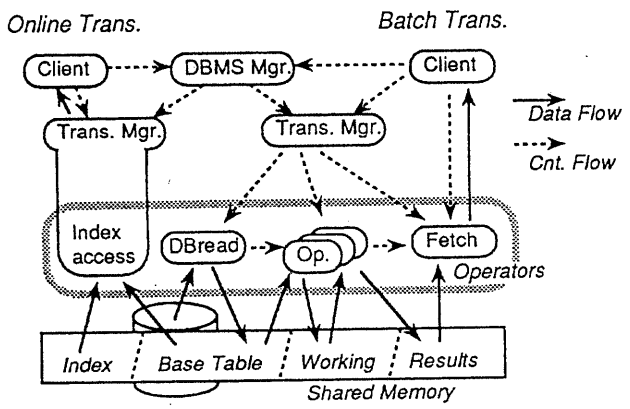

図1：プロセス構成

レルとパイプラインの2種類の並列処理を厳密に制御する．ＤＢＭＳ管理部は，プロセッサとメモリ（バッファ）の計算機資源を管理するプロセスで，クライアントから要求を受け付けるとトランザクション管理部を生成する．トランザクション管理部は，クライアントの要求に応じた処理と制御を担当し，負荷が小さいオンライントランザクションであれば自ら処理を実行し，負荷が大きいバッチトランザクションは，複数のオペレータプロセスを生成してそれらの並列処理を管理する．同時に実行する複数のトランザクション間の実行制御は，ＤＢＭＳ管理部がトランザクション管理部に割り当てる資源量で制御する．

並列処理のスケーラビリティを低下させる大きな要因は，プロセス間の通信遅延と負荷の不均衡によるスキューであり，筆者等は大量タスクを少ない通信回数で負荷配分する適応型動的負荷配分法を提案している[Hira91]．この方法は，並列処理しているプロセス（プロセッサ）に割り当てるタスク数を残りタスク数の関数として計算し，処理の開始時には沢山のタスクを一度に割当て，処理の進行にともなって割当て数を減らすことで，大量のタスクを少ない割当て回数（通信回数）で各プロセッサに均等配分する．この適応型動的負荷配分法は，バッチ処理におけるデータベース・スキャンで，大量のページをプロセッサに配分する際に適用する[Sato92]．

通信オーバヘッドは通信量と通信回数に依存する．このため，プロセス間で受け渡すデータ実体は共有メモリに格納しておいてポインタだけを渡すことで通信量を削減する．また，パイプライン動作しているプロセス間のデータ転送はデータ駆動型とすることでデータ転送要求の発行回数を削減する．このデータ駆動型の転送制御で問題となる先行するプロセスのオーバランは，トップダウン実行制御によるバッファ割当て量制御によって抑止する．

3.2 資源割り当てによる実行制御

サーバ内の実行制御は，ＤＢＭＳ管理部→トランザクション管理部→オペレータプロセスの階層構造である．トランザクション管理部は，クライアントからのオーダを解析して実行プランを作成する．オンライン系の処理は，並列処理する自由度が小さくトランザクション管理部が実行するので，ここではバッチ系の問合せを並列に処理する場合の資源割当てによるトップダウン実行制御法を示す．ここで割当ての対象としている資源はプロセッサとメモリであり，それぞれ，並列処理するオペレータプロセスの生成個数と同時に使用するバッファの面数に対応する．

1）トランザクション管理部は実行プランに基づいてＤＢＭＳ管理部に最適な資源数の割当てを要求する．実行に最適なバッファサイズは，既存のDB-MIN法[Chou85]やhot-set法[Sacc86]を用いることで決定できる．

2）ＤＢＭＳ管理部は，複数のトランザクション管理部からの資源割り当て要求を，実行の優先度（オンライン＞バッチ）や現在実行しているトランザクション数，残り資源数とで調停し，割当て可能な最大資源数をトランザクション管理部に提示する．

3）hot-set法によれば，トランザクションの実行時間は資源（バッファ数）に対して段階的に変化するので，2）で提示されたバッファ数以下の最大なセットポイントを求める．並列処理するプロセッサ数についても，上記セットポイントで最適となるように再計算し，これら使用資源量をＤＢＭＳ管理部に報告する．

4）トランザクション管理部は，3）で求めた資源を用いて並列処理を開始する．

プロセッサ：同時に実行できる最大数が割当てプロセッサ数となる様に複数個のプロセスを生成する．一般に，交互にパイプライン動作するプロセスや頻繁にディスク入出力待ちとなるプロセスがあるので，割当てプロセッサ数より多数のプロセスを生成できるが，どれだけ余分のプロセスを生成できるかは処理内容に依存する．

バッファ：トランザクション管理部は，割り当てられた範囲内のバッファを各プロセスに割当て・回収する．ソートや結合処理のために大量の中間結果が作られる場合は，トランザクション管理部が実行プランに基づいて，最も後まで使用されないと判断したページを解放する．解放されたページは，次章で示すバッファ管理部によって適当な時期にバッファから追い出される．

5）処理が資源の再割当てが可能な同期ポイントまで進んだ時点で，トランザクション管理部はＤＢＭＳ管理部に対して資源の再要求（要求と返却の２つの場合が有る）を行なう．単一トランザクションの実行に関して同期ポイントで資源割当てを変更する方法は[Murp91]に示されている．ＤＢＭＳ管理部は，２）３）の手順に基づいて再割当てを実行する．トランザクションの実行が完了してコミットされたらトランザクション管理部は割り当てられた資源を全てＤＢＭＳ管理部に返却する．

4　共有バッファの管理

4.1 従来法とその問題点

バッファ管理の課題は，(1)バッファ探索コストの削減と，(2)ヒット率が高いバッファの割当てと置換を実現することであり，文献[Effe84]に従来のバッファ管理アルゴリズムが分類・整理されている．近年，主記憶の大容量化が急激に進んでいるが，ハッシング技術を用いることによって(1)のバッファ探索コストを増大させずに管理することができる．一方，バッファ割当て法(2)は，バッファ全体を一括管理するグローバル管理法，トランザクション毎にバッファを割り当てるローカル管理法，ページタイプに基づいてバッファを割り当てるタイプ別管理法に分類される．トランザクションの処理内容に応じて動的にページを割当てる working-set 法はローカル管理法の代表的な例で，注意深く割当てページ数を決定することで高いヒット率を達成できる[Effe84]．しかし，この方法をオンラインとバッチ混在時のバッファ管理法に適用すると，複数のオンラインユーザのためにバッファが細分化される分割損が生じたり，バッチトランザクションに大量なバッファが割り当てられてオンラインの性能が低下する等の問題が生じる．

バッファ置換アルゴリズムには，FIFO, LRU(Least Recentry Used), LFU(Least Frequently Used)等がある．これら従来のアルゴリズムは，過去のアクセス経緯から次にアクセスされるであろうデータをできるだけ高い確率でバッファに保持する方法で，バッファ管理部内で得られる情報，たとえばアクセスの順序や頻度，データのタイプを用いて制御していた．このため，バッファ管理を独立したサブシステムとして構成できる特徴を持つが，オンラインとバッチを混在実行する場合には以下の問題が生じる．(1)バッチ処理でスキャンが完了したページは，その後使用される可能性が低いにもかかわらずLRUではバッファ内に高い確率で存在してしまう．(2)バッチで生成される中間結果は，頻繁に利用されるデータではないのでLFUアルゴリズムでは，早い段階でバッファから追い出されてしまう．

これらの問題を解決するために，バッファを一括管理するグローバル管理法を基本として，ページタイプとアクセスタイプとからバッファ置換の優先度を決定し，優先度に基づいてバッファの割当てと置換を行なう方法を提案する．従来の優先度制御による方法として，リアルタイムデータベース処理を実現するためにトランザクションの実行優先度を継承する方法[Jauh90]が知られているが，ここではオンラインとバッチの２段階の優先度しか存在しないので十分な優先度分けができない．

4.2 データとアクセスタイプによる優先度

(a)データタイプ

データベースに格納されるオブジェクトには，スキーマ，インデックス，テーブル等の種類があり，いずれのオブジェクトも一般にページと呼ばれる固定サイズのブロックを単位としてディスクに格納されている．Bツリー構成のインデックスでは，100エントリ以上/ノード，(インデックス容量)=(テーブル容量)/10程度であるから，ルートノードと中間ノードを合わせた容量はテーブルの1/1000程度と十分に小さい．一方，インデックスのアクセスは必ずルートノードから開始されるので，ルートに近いノードほど容量が小さくアクセス頻度が高くなる．このため，バッファ管理の優先度をルートノードほど高く設定することで，バッファのヒット率を高くできる．

テーブルは容量が大きく最もバッファの効果が得られにくいデータオブジェクトである．オンラインとバッチの混在実行では，両トランザクションからアクセスされるオブジェクトであるが，いずれのテーブル，あるいはいずれの領域が頻繁にアクセスされるかは，一般に問合せ内容に依存し予測することが困難である．したがって，テーブル毎にバッファ管理の優先度を設定することは，アクセス変動に弱くバッファの分割損を生じる要因となる．

以上の観点から，データタイプを用いたバッファ管理の優先度として，スキーマ，インデックスのルートノード，中間ノード，リーフノード，テーブルの５段階が考えられる．上述のＢツリー型インデックスの容量推計によれば，一般的に想定される容量のバッファは，スキーマ，インデックスのルートと中間ノードを常駐可能と考えられる．これより少ないバッファ容量で複数のオンライントランザクションを並行処理すると，単なる入出力バッファとしての機能しか得られない．以上のことから，現実的な応用では，インデックスのリーフノードとテーブルを格納するページがバッファの入れ替え対象となるといえる．

(b)アクセスタイプ

トランザクションの優先度を継承したバッファの優先度管理の問題点は，バッチトランザクションがスキャンしたテーブルのページがバッファから早期に追い出されないことと，所定の時間経過後に必ず使用される中間結果が不用意に追い出されてしまうことである．この問題を解決するために，処理内容に基づくデータベースのアクセス特性を考慮したアクセスタイプによる優先度の決定法を提案する．

アプリケーションは，自分がアクセスしたデータオブジェクトがバッファ内でどの程度近い将来に再利用されるかを知ることができる．たとえば，バッチトランザクションがデータベースをシーケンシャルアクセスしたページは極めて再利用される確率が低いといえる．(少なくとも自分にとっては最も再利用する確率が低い) これに対して，ジョイン等のために生成した中間結果は，近い将来に必ず再利用されることが保証される．一方のオンライントランザクションは，アクセスしたページがインデックスであれば他のオンライントランザクションによって高い確率で再利用されると考えられるし，テーブルであれば少なくともインデックスよりは確率が低いと考えることができる．このようにアプリケーションのアクセスタイプを用いることでバッファにきめ細かい優先度を指定できる．

(c)優先度の統合

データタイプに基づく優先度は，バッファするページ内の制御情報からページタイプを検出することでバッファサブシステムの内部で設定できる．すなわち，ページ読み込み時あるいは解放時にページ内の制御情報からタイプを検出して優先度を決定できる．これに対して，アクセスタイプによる優先度の決定は，アプリケーションとバッファサブシステムとのインタフェースに優先度を指定できるような拡張が必要となる．この拡張をしない場合は，サブシステム側でアクセスに関する何らかの統計情報をとることでアクセスタイプに相当する情報を収集する方法が考えれれるが，複数のオンライントランザクションとバッチトランザクションを混在処理している状況で正確な統計を取ることは極めて困難である．優先度を指定できるインタフェースを設けた場合は，アプリケーションはアクセスタイプ以外にデータタイプも当然知っているため，これら2種類のタイプを統合した優先度をバッファサブシステムに指定できる．

以上示したデータタイプとアクセスタイプとからバッファ管理の優先度を決定する．

(1)スキーマ：トランザクションの実行開始時に必要となる情報で容量が小さい．

(2)インデックス：実行優先度の高いオンライントランザクションがアクセスするデータである．ルートノードに近いほど容量が小さいので次の順序で高い優先度を与え，バッファされる確率を高くする．

(2-1)ルートノード

(2-2)中間ノード

(2-3)リーフノード

(3)テーブル：テーブルはバッファ容量より遥かに大容量のデータであるため低い優先度を設定する．実行優先度の高いオンライントランザクションによるランダムアクセスは，アクセスするページに偏りがあるためバッファリングの効果が期待できる．これに対して，バッチトランザクションによるシーケンシャルアクセスでは再利用の可能性は低い．そこで，次の順序で高い優先度を与える．

(3-1)ランダムアクセスページ

(3-2)シーケンシャルアクセスページ

(4)中間結果：実行優先度の低いバッチが生成するデータであるが，後続する処理で確実に使用されるデータであるため比較的高い優先度を与える．

ここで，バッファ容量と各データタイプの容量とを比較すると，(1)スキーマ，インデックスの(2-1)ルートノードと(2-2)中間ノードを合わせた容量よりバッファ容量は大きいと考えられる．またこれらのデータの容量は，データベース全体の容量からあらかじめ推定できることから，これらのデータをバッファサブシステム内に常駐させるような容量を持つバッファを設けることができる．一方，中間結果は確実にアクセスされるデータであるから高い優先度を与えたいが，その容量は，処理内容や検索の条件(データ選択率)に容量が大きく依存し，あらかじめその容量を見積もることは困難である．このため，中間結果には上記3種類のデータより低いインデックスの中間ノードとリーフノードの間の優先度を与えることにする．1つの処理で大量の中間結果を生成するとバッファが中間結果で一杯になり，それより優先度は低いが確率的な使用頻度の高いデータが追い出される問題が生じる．このため，3章に示した資源割当てによって中間結果として生成できるページ数の上限をバッファの総容量等から設定しておき，中間結果が作成限界に到達したら，アプリケーションは自らの処理の内容に応じて最も後で使用するページの優先度を下げる．これにより，アプリケーションは共有バッファ（メモリ）を自分の拡張メモリとして利用できる．個々のアプリケーションは，内部に中間結果を格納するための大量のバッファ（変数領域）を持つ必要がなくなり，1つのバッファサブシステムによって主記憶を統一した置換アルゴリズムで有効に利用できるようになる．

(d) 同一優先度内の制御

既存の置換アルゴリズム(LRU，LFU)で問題となる不用意な追い出しは優先度を設けたことで解決さ

れる．LRUは単純なリスト操作で実現できるのに対して，LFUはバッファするページ毎に頻度カウンタを設けて頻度順に並べ換える操作が必要となる．このため，バッファ管理のオーバヘッドが小さいと考えられるLRUアルゴリズムを用いることにした．LRUアルゴリズムを用いることで，オンライントランザクションのアクセススポットに位置するページがバッファに存在する確率を高くできる．また，バッチトランザクションのデータベーススキャンで読み込まれたページは，次ぎのアクセスまでの期間が長いために早い時期にバッファから追い出されることになる．

4.3 シミュレーション評価

複数のオンライントランザクションとバッチトランザクションの混在実行をモデル化して，提案するバッファ管理法をシミュレーション評価する．シミュレーションモデルを図2に，評価パラメータを表1に示す．バッファは，インデックスのルートと中間ノードを常駐するのに十分な容量を持つとし，10000ページのテーブルとテーブルの1/10のページ数からなるインデックスのリーフを入れ替え対象とする．最大5のオンラインユーザと最大1のバッチユーザは，図1のアクセスシーケンス生成部でオンラインかバッチのいずれかのトランザクションに対応するページ要求を発生する．オンライントランザクションは，インデックスとテーブルの各1ページを各々平均５００標準偏差１００の正規分布と平均５０００標準偏差１０００の正規分布に従がうページ番号を要求する．一方，バッチトランザクションは，ランダムに発生した初期ページ番号から連続する１００ページを要求する．要求ページ番号の生成と同時にアクセスシーケンス生成部では，以下に示す3種類のバッファ優先度生成基準に基づいて，バッファする際の優先度を決定する．バッファ管理部では，指定された優先度毎にLRUリストを作成してバッファしているページを管理する．要求されたページがバッファに無い場合は，ディスクからの読み出しを行なった後にCPUを確保して対応する処理(t2,t4,t7が処理時間に相当）を行なう．なお，優先度を設けたバッファ管理部の効果を評価するために，DISKおよびCPUの資源は十分に有るとした．

比較評価した3種類のバッファ優先度の設定方法を示す．

(a) 優先度制御なし

優先度制御を行わないバッファ管理で，単純なLRUアルゴリズムによってバッファの入れ替えを行う．

(b) データタイプ優先度

優先度の決定法の節で示したように，バッファはインデックスの中間ノードまでを格納できる程度の容量を持っているとして，インデックスのリーフノードとテーブルに対応する2種類の優先度を設定する．

(c) データタイプとアクセスタイプを統合した優先度

上記データタイプによる2種類の優先度にランダムアクセスとシーケンシャルアクセスの2種類のアクセスパタンを組み合わせた優先度を設定する．但

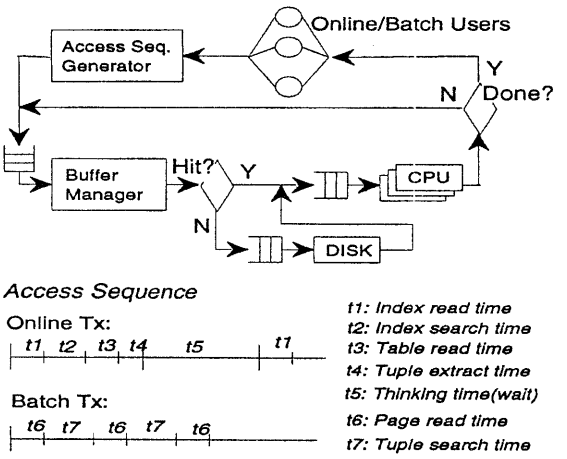

図2．シミュレーションモデル

表1．評価パラメータ

Database Model	
Index size	1,000 pages
Table size	10,000 pages
Transaction Model	
Online users	0 to 5
Batch users	0 to 1
Online Transaction	
t1: Index read time	depend on buffer mgr
t2: Index search time	2 mSec
t3: Table read time	depend on buffer mgr.
t4: Tuple extract time	1 mSec.
t5: Thinking time	100 mSec.
Access pattern	
Index	normally dist. m: 500, std: 100
Table	normally dist. m:5000, std:1000
Batch Transaction	
t6: Page read time	depend on buffer mgr.
t7: Tuple search time	10 mSec./page
Access pattern	Sequential
Number of pages	100 pages/transaction
Workload Model	
Disk access time	16 mSec.
Buffer search time	0.01 mSec./page

し，本評価ではインデックスのシーケンシャルアクセスは行なわないので，結果として３種類の優先度を用いてバッファを管理する．

バッファ容量を500ページと2000ページとしてシミュレーションした．インデックス容量が1000ページであるから，バッファ容量500ページの場合は，インデックスのホットアクセススポットだけがバッファリングされ，バッファ容量が2000ページの場合は，インデックス全体とテーブルのホットアクセススポットがバッファされることが期待される．

オンラインユーザ数を変えた時のオンラインの応答時間を図３に示す．オンラインは，インデックスとテーブルを正規分布でアクセスすることから，オンラインユーザ数を増やすとより顕著なアクセススポットができる．したがって，オンラインユーザ数が増えると等価的にバッファのヒット率が向上しオンラインのレスポンスが向上する．（図３(a)参照）

次に，図３のオンラインの応答時間を比較して優先度制御の効果を評価する．
データタイプによる優先度制御を行なう(b)は(a)と比べて，バッチトランザクションの影響が小さい．特に，バッファ容量が500ページの場合は，オンラインユーザ数に関係無くバッチの影響を受けていない．しかし，バッファが2000ページの場合は，同一優先度であるテーブルのホットスポットページがバッチのシーケンシャルアクセスによって追い出されるために，オンラインはバッチと混在実行すると２０％程度の応答時間の悪化がみられた．これに対して，データタイプとアクセスタイプを統合した優先度を与える(c)では，いずれのバッファ容量とオンラインユーザ数でもバッチの影響を全く受けていない．

表２は，図２と同様の条件で測定したバッチの応答時間である．バッチは，所定のページ数（100ページ）をシーケンシャルにアクセスするので，テーブル全体がバッファできるほど大容量のバッファでない限り，アクセスするページがバッファリングされていることは期待できない．図３の結果によれば，優先度制御の違いによる応答時間の差はない．シミュレーションでは，他のディスクやCPU資源を十分に備えていることから，オンラインユーザ数が増えてもバッチの応答時間は悪化していない．

本評価では簡単のため並行処理制御を省略したが，ここで扱うトランザクションは更新を含まないのでデータベースの一貫性が損なわれることはない．また，１つのトランザクションでアクセスするページ数はデータベースサイズに比べて小さくかつ同時ユーザ数も比較的少ないので，並行処理制御を考慮した場合にもほぼ同様の結果が得られると考えられる．

5　ＤＢＭＳプロトタイプの構成法

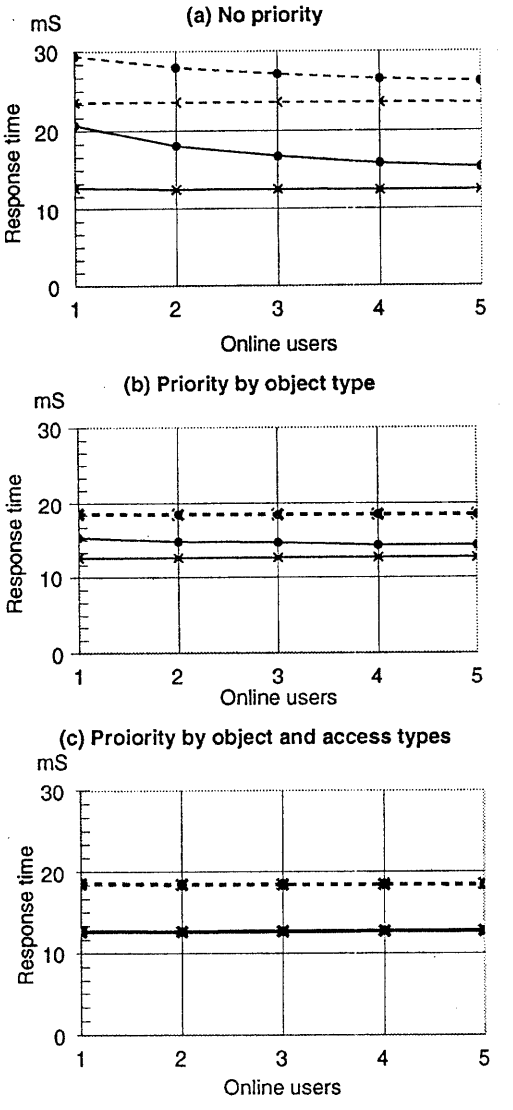

図３．オンライントランザクションの応答時間

表２．バッチトランザクションの応答時間

方式	Buffer pages	Online users 0	1	2	3	4	5
(a)	500	2.41	2.43	2.43	2.44	2.44	2.44
	2000	2.14	2.19	2.19	2.21	2.20	2.20
(b)	500	2.41	2.50	2.50	2.50	2.50	2.50
	2000	2.14	2.24	2.24	2.27	2.27	2.27
(c)	500	2.41	2.50	2.50	2.50	2.50	2.50
	2000	2.14	2.24	2.25	2.25	2.25	2.25

5.1 バッファサブシステムの構成

バッファ容量の飛躍的な増大に対して，バッファ探索コストを増大させないバッファ構成が必要となる．図4は，単一LRUリストで実現した図3(c)のバッファ管理に探索コスト0.01m秒/ページを考慮した場合の応答時間である．バッファサイズが大きくなるとバッファのヒット率が向上する(図3(c))にも関わらず，探索コストの増大によって応答時間が悪化している．LRUリストの探索・更新は，複数プロセッサでの同時処理ができないクリティカルな処理である．このため，LRUリストを分割した複数のサブリストで管理する方法が提案されている[Effe84][Jauh90]．バッファ管理の優先度別にリストを構成する[Jauh90]の方法は，低優先度の追い出し対象ページを容易に探索できる利点があるが，リストの分割に制約があるため，ここでは，ページ番号でリストを分割する[Effe84]の方法を基本にバッファサブシステムを構成する．その構成を図5に示す．

アプリケーションによるバッファの探索はページ番号をキーとして行なうことから，検索対象とするリストをページ番号のハッシュ値で決定し，選択されたリストだけをロックして探索・更新を行なうことで，複数アプリケーションによる同時検索を実現できる．一方，バッファ置換のための追い出しページは優先度が低く最も最近に利用されていないページであるから，以下の手順で探索する．まず，追い出し対象ページを探索するリストをラウンドロビンで決定する．選択リスト内に最低優先度のページが存在すれば，それらのページを一括して追い出す．もし存在しなければラウンドロビンに次のリストを選択し，最低優先度のページを追い出すことを試みる．全てのリストを一巡しても追い出し対象ページが存在しなければ，優先度を上げて，上記ラウンドロビンによるリスト単位の追い出しを繰り返す．

アプリケーションとのインタフェースとして以下の4種類の関数を用意した．バッファの生成は，共有メモリ上に確保するバッファの面数（フレーム数）と分割リストの本数を指定して行なう．プロトタイプでは，ページ番号をリスト数で割った剰余でリストを決定することにした．バッファ管理の優先度は，ページを解放する時に指定する方式とすることによって，バッチトランザクションが大量の中間結果を一時的に保持する際に，その一部分を優先度を指定して解放できるようにした．

```
bufCreate( frames, entries )
bufDelete()
bufFix( pageid )
bufRelease( pageid, priority )
```

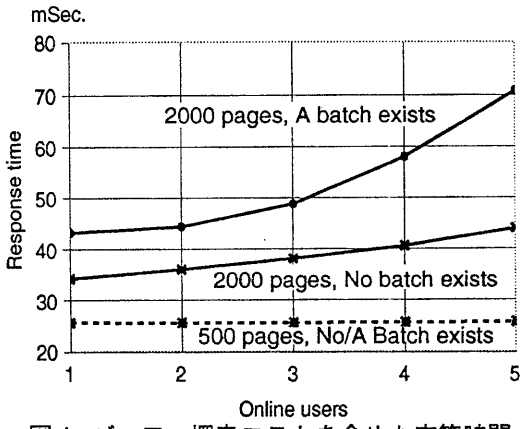

図4．バッファ探索コストを含めた応答時間

5.2 バッチトランザクションの並列処理法

(a)単純問合せ

検索条件でデータベースをスキャンする問合せは，典型的なバッチトランザクションである．図1に示したプロセス構成において，ＤＢリーダ，複数の選択オペレータとフェッチオペレータがパイプラインとパラレルの2種類の並列処理を行なう．これらの並列処理は，トランザクション管理プロセスによって制御される．

(1)ＤＢリーダは検索対象テーブルを格納しているページを読み出して複数の選択オペレータに配分する．一度に配分するページ数は，選択オペレータ数と残りページ数等から算出し，最初は多くのページを，処理が進につれて少ないページ数を配分する[Hira91].

(2)選択オペレータは，配分されたページを問合せ条件で検索し，条件に合致した行を出力ページに書き出す．一杯になった出力ページは次のフェッチオペレータに渡し，空のページを返却してもらうことで処理を継続する．

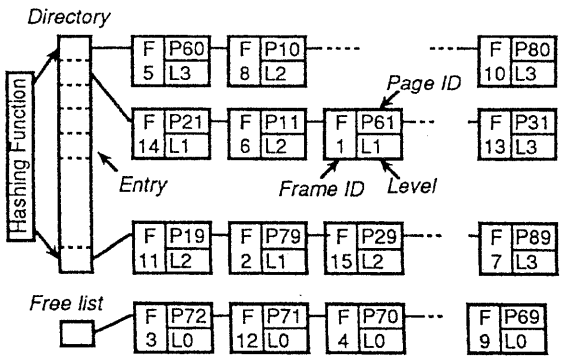

図5．バッファサブシステムの構成

(3)フェッチオペレータは，選択オペレータから出力ページを受け取ると同時に，クライアントプロセスに結果を渡す処理を繰り返す．ここで，検索結果の一時表を格納できる最大ページ数をあらかじめトランザクション管理プロセスが設定しておく．これにより，一時表が一杯になると(2)や(1)の処理が中断されて，不必要な中間結果のディスク書き出しが抑止される．

(b)結合を含む問合せ

ハイブリッド・ハッシュ結合法[Schn89]は，第１表の選択結果をハッシュ関数によって複数のバケットに分割格納し，バケット間で独立に第２表との突合せ処理が行なえる並列処理に適した手法である．以下では，優先度管理を行なうバッファサブシステムを用いて中間結果であるバケットを格納する手法を示す．この容量は，選択条件に依存するために，あらかじめ十分な容量のバッファを割り当てておくことができない．中間結果が割当て容量を超えた場合は，バケット単位でバッファから追い出してディスクに書き出す処理が必要であり，この中間結果の格納管理がプログラムを複雑にしていた．提案する優先度付きのバッファサブシステムを用いることで，中間結果の格納管理をサブシステムで行なえる．

(1)第１表の選択結果はバッファ上の一時表としてページ単位で格納すると同時に，従来のハイブリッドハッシュ法と同様にバケット単位に管理リストを作成する．一時表として格納できる最大ページ数は，トランザクション管理プロセスが予め設定しておく．

(2)中間結果の容量が指定された最大ページ数に達した場合は，(1)で作成したバケット単位の管理リストを用いて適当なバケットをバッファから解放する．解放されたページは，バッファサブシステムによって適当な時期にディスクに書き戻されてバッファから追い出される．解放したページ数を使用して中間結果の作成と格納を継続する．

6　おわりに

本稿では，従来独立に処理していた同一データベースに対するオンラインとバッチトランザクションを混在実行するための課題を明らかにし，メモリ共有型マルチプロセッサを用いた並列処理法と，そこでのバッファ管理法について示した．アプリケーションが操作する際のデータタイプとアクセスタイプとからバッファ管理の優先度を決定する優先度付きバッファ管理法が，オンライントランザクションの実行性能を低下させずにバッチを実行するのに適していることをシミュレーションで検証した．この方法は，バッチトランザクションの実行過程で必要となる中間結果の格納に適していることを，プロトタイプＤＢＭＳの設計を通して確認した．

参考文献

[Chou85] Chou, H.-T. and DeWitt, D. J.: "An Evaluation of Buffer Management Strategies for Relational Database Systems", Int'l Conf. on Very Large Data Bases, pp. 127-140, 1985

[Effe84] W. Effelsberg and T. Haerder: "Principles of Database Buffer Management", ACM Trans. on Database Systems, Vol.9, No.4, pp. 560-595, Dec. 1984

[Hira92] 平野泰宏, 佐藤哲司, 他: "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", 信学論 Vol. J75-D-I, No. 3, pp. 152-159, Mar. 1992

[Inou91] Inoue, U., Satoh, T. et al: "Rinda: A Relational Database Processor with Hardware Specialized for Searching and Sorting", IEEE Micro, Vol.11, No. 6, pp. 61-70, Dec. 1991

[Jauh90] R. Jauhari, M. J. Carey and M. Livny: "Priority-Hints: An Algorithm for Priority-Based Buffer Management", 16th Int'l Conf. on Very Large Data Bases, pp. 708-721, 1990

[Kata91] 片岡良治, 佐藤哲司, 井上潮: "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", 信学論 Vol. J74-D-I, No. 3, pp. 224-231, Mar. 1991

[Murp91] Murphy, M. C. and Shan, M.-C.: "Execution Plan Balancing", 7th Int'l Conf. on Data Engineering, pp. 698-706, Apr. 1991

[Nech88] Neches, P. M.: "The Ynet: An Interconnect Structure for a Highly Concurrent Data Base Computer System", Proc. 2nd Symp. Frontiers of Massively Parallel Computation, pp. 429-435, 1988

[Sacc86] Sacco, M. and Schkolnick, M. :"Buffer Management in Relational Database Systems", ACM Trans. on Database Systems, Vol. 11, No. 4, pp. 473-498, 1986

[Sato92] Satoh, T., Hirano, Y., et. al.: "Design and Implementation of Parallel Database Processing on a Shared Memory Multiprocessor System", Proc. of 2nd Far-East Workshop on Future Database Systems, pp. 337-346, Apr. 1992

[Schn89] Schneider, D. A. and DeWitt, D. J.: "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", ACM SIGMOD Record, Vol. 18, No. 2, pp. 110-121, Jun. 1989

[Serl91] Serlin, O: "The History of Debit Credit and the TPC", In The benchmark Handbook for Database and Transaction Processing Systems, Jim Gray ed., pp. 19-114, Morgan Kaufmann Publishers, 1991