並列ディレクトリ構造Fat-Btreeの並行性制御の改善とその評価

吉原　朋宏† 　小林　大†† 　田口　亮††† 　上原　年博††† 　横田　治夫††,††††

† 東京工業大学 工学部 情報工学科
†† 東京工業大学 大学院 情報理工学研究科 計算工学専攻
††† NHK 放送技術研究所
†††† 東京工業大学 学術国際情報センター
E-mail: †yoshihara@de.cs.titech.ac.jp, ††daik@de.cs.titech.ac.jp, †††{taguchi.r-cs,uehara.t-jy}@nhk.or.jp,
††††yokota@cs.titech.ac.jp

あらまし　無共有並列計算機に適した分散ディレクトリ構造である Fat-Btree の並行性制御として新たな手法を提案する．提案手法は，楽観的なラッチカップリング中に SMO(structure modification operation) が起きるポイントをマークすることで，従来の INC-OPT 方式と比べて，リスタート回数を少なく抑え，SMO のための X ラッチ獲得時間を短くすることができる．また，木の構造の変化を扱う 3 種類の手法を提案する．Fat-Btree を採用している自律ディスクに提案手法を実装し, 従来手法との比較を行う．更新要求の割合を変化させた場合とシステムの規模を変化させた場合の実験から，提案手法が常にシステムスループットを改善し，高更新環境および大規模システムにおいて特に有効であることを示す.

キーワード　インデックス, 並行制御とリカバリ最適化, 並列・分散 DB, Fat-Btree

Improvement and Evaluation of Concurrency Control Method for the Fat-Btree, Parallel Directory Structure

Tomohiro YOSHIHARA†, Dai KOBAYASHI††, Ryo TAGUCHI†††, Toshihiro UEHARA†††, and Haruo YOKOTA††,††††

† Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology
†† Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
††† NHK Science & Technical Research Laboratories
†††† Global Scientific Information & Computing Center, Tokyo Institute of Technology
E-mail: †yoshihara@de.cs.titech.ac.jp, ††daik@de.cs.titech.ac.jp, †††{taguchi.r-cs,uehara.t-jy}@nhk.or.jp,
††††yokota@cs.titech.ac.jp

Abstract　We propose a new concurrency control method for parallel Btree structures, such as the Fat-Btree, used in the shared-nothing environment. The proposed method marks a structure modification operation (SMO) occurrence point during the optimistic latch coupling to reduce the frequency of restarts and to shorten the duration of an X latch for an SMO compared with the conventional INC-OPT method. We also propose three variations of the method for handling a tree structure change. To compare the performance of these methods and the INC-OPT method, we implemented them on an autonomous-disk system adopting the Fat-Btree. The experimental results with changing update ratio and system size indicate that the proposed methods always improve the system throughput, and are especially effective for higher update ratio and large scale configuration.

Key words　index, concurrency control and recovery optimization, parallel・distributed DB, Fat-Btree

1. はじめに

データベース用無共有並列計算機における検索，更新処理は，参照されるデータが配置されている各 PE(Processing Element) 上で並列に実行されることが望ましい．アクセス集中による負荷の偏りが存在する場合，負荷が大きい PE がボトルネックとなり，全体の処理性能が低下してしまう．したがって，各 PE で負荷を均等化することは処理性能の向上につながり，負荷を均等にするためのデータ分配方式が重要になる. [1], [2]

従来のデータ分配方式にはハッシュ分配方式や値域分配方式 [3] などがあるが，ハッシュ分配方式では領域指定された問い合わせや，連続したアクセスの I/O 回数を削減するクラスタ化に対応できないという欠点がある．一方，値域分配方式では，静的決定された分割境界に沿って分割するため，データ更新によってデータ配置の偏りが生じたときに均一化するコストが非常に大きくなる欠点がある．

データ分配方式として値域分配方式を採用した上で，インデックス構造に並列 B-Tree を用いる研究がある．並列 B-Tree を利用することによって，両方式の欠点が解消でき，同時に高速アクセスが可能になる．しかし，従来の並列 B-tree ではディレクトリ更新によるスループットの低下や，少数の PE へのアクセスの集中といった問題が生じる．

これらの問題を解決するために，新しい並列 B-Tree 構造として Fat-Btree が提案されている [4]〜[12]．ディレクトリ構成として Fat-Btree を用いることで，単一キー問い合わせ，レンジ問い合わせが並列に高速実行できることが確率モデルの検証 [4], および nCUBE3 [6] や LAN 環境での PC クラスタ [12] 上への実装による実験により明らかにされている．

Fat-Btree を含めた並列 B-Tree に適した並行性制御方式として INC-OPT 方式が提案されている [7], [10]．無共有並列計算機向けに設計されている INC-OPT は，従来の B-Tree の並行性制御方式である B-OPT [13] や ARIES/IM [14] より優れたパフォーマンスを示すことが明らかにされている [7], [10]．しかし，INC-OPT は B-Tree の構造変化を起こす操作 (SMO:Structure Modification Operation) が広範囲で行われるとき，ルートページからの木降下リスタートが多数必要となり，システムの処理性能の低下をもたらす．

本稿では，INC-OPT と比べて，SMO 発生時のリスタート回数を少なく抑えることが可能である新しい Fat-Btree の並行性制御 MARK-OPT 方式を提案する．また，MARK-OPT の拡張として，更新処理プロトコルの第 2 フェーズの木構造の変化に着目した INC-MARK-OPT 方式，2P-INT-MARK-OPT 方式および 2P-REP-MARK-OPT 方式の 3 つの手法を提案する．さらに，分散ディレクトリとして Fat-Btree を採用している自律ディスク [15]〜[17] 上に INC-OPT および提案 4 手法を実装し，更新要求の割合を変化させた場合と台数を変化させた場合のスループットを測定し，提案手法の効果を示す．

以下に本稿の構成を述べる．まず 2. 節で Fat-Btree と従来の並行性制御方式について述べる．次に 3. 節では提案する並行性制御方式について述べる．4. 節では提案手法と従来手法の実験による評価について述べる．最後に 5. 節でまとめと今後の課題を述べる．

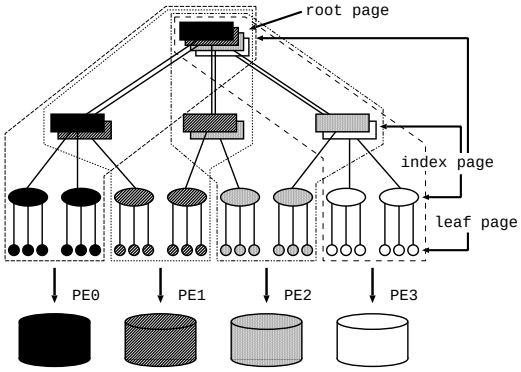

図 1　Fat-Btree の例

2. Fat-Btree と並行性制御

2.1 Fat-Btree 構造

Fat-Btree は，B^+-Tree 全体をページ単位で PE 間で分配する並列 B-Tree の一種であり，データページである B-Tree の葉ページを各 PE に均等に分配する．ディレクトリ部分である B-Tree の葉ページ以外は，各 PE に配置されている葉ページへのアクセスパスを含むインデックスページにのみ再帰的に配置する．これにより，各 PE のディスクに格納されるのは，B-Tree のルートから均等に分配された葉ページまでの部分木である (図 1).

更新頻度が高い下位のインデックスページほどそのコピーを持つ PE が減少していくため，ディレクトリ更新時に同期が必要となる PE 数が少なくなり，PE 間の局所的な通信によって更新処理を行うことができる．

また，各 PE では格納している葉ページの探索に必要のないインデックスページを持たないため，各 PE でインデックスページのキャッシュを行った場合にキャッシュのヒット率を高く保つことができる．高キャッシュヒット率により，更新処理だけでなく，探索処理も従来の並列 B-Tree 構造と比較して高速に行うことができる．

2.2 並行性制御

木構造の一貫性を保証するために，B-Tree に並行性制御は必須である．通常，B-Tree や他のアクセスパスには，ロックの代わりにデッドロック検出機構を持たない高速かつ単純なラッチが用いられる．ラッチはセマフォの一種である．従って，アクセスパスの並行性制御はデッドロックフリーでなければならない．

2.2.1 ラッチモード

本稿では，5 種類のモードを持つラッチを仮定する．各モードは IS，IX，S，SIX，X であり，これらのモードの適合性は表 1 に示される [18]．表中の“ ○ ”は同時に複数のラッチが獲得可能なモードである．

あるインデックスページの複製が複数の PE に存在する場合を考える．もしラッチ処理が各 PE で分散して行われるならば，表 1 より，IS および IX モードはローカル PE のみで獲得するだけでよい．しかし，S，SIX および X モードは，コピーを持

表 1　ラッチマトリクス

mode	IS	IX	S	SIX	X
IS	○	○	○	○	×
IX	○	○	×	×	×
S	○	×	○	×	×
SIX	○	×	×	×	×
X	×	×	×	×	×

つすべての PE でラッチを獲得する必要がある．すなわち，S，SIX および X モードのラッチはデータのコピーが存在するとき，PE 間の同期オーバーヘッドが生じる．さらに各 PE に対してランダムにラッチを獲得しようとすればデッドロックの可能性がある．デッドロックを回避するためには PE 番号の昇順または他の順序に従いラッチを獲得する方法が有効である．しかし，これはラッチに関わる PE 数に比例して同期完了までの時間が長くなることを意味している．

2.2.2　Fat-Btree の並行性制御

前述のとおり，アクセスパスの並行性制御はデッドロックフリーでなければならない．

また，多数の PE にコピーを持つ Fat-Btree の上位ページにおいて，S，SIX および X ラッチを獲得することはできる限り避けなければならない．なぜなら，並列 B-Tree における S，SIX および X ラッチの獲得は PE 間の同期が必要で，多くの PE にコピーを持つページでのそれらのラッチの獲得は，PE 間の同期オーバーヘッドが増大させてしまうからである．

表 1 のラッチを用いる方式と異なる並行性制御方式として，木ラッチと呼ばれる B-Tree 全体のラッチを用いて，SMO 発生時の B-Tree の並行性制御を行う方式が存在する [14]．その方式を分散環境における並列 B-Tree で実現するためには，ある特定の PE で木ラッチを管理するのが最も簡単であるが，PE 数が増加すればその PE にラッチリクエストが集中するため，その PE がボトルネックとなる．また，木全体をラッチするため，PE 間の同期オーバーヘッドも大きい．

以上のことを要約すると，Fat-Btree を含む並列 B-Tree の並行性制御は，一般に次の条件を満たすことが望まれる [7], [10].

［条件 2.1］ 並列 B-Tree の並行性制御には以下の三点を満たすことが要求される．

（1） デッドロックフリーである
（2） ルートおよび木の上位ページにできる限り S，SIX および X ラッチを獲得しない
（3） 短時間であっても木全体をラッチすべきではない

優れた B-Tree の並行性制御方式として，B-OPT [13]，OPT-DLOCK [19] および ARIES/IM [14] などがあるが，これらは条件 2.1 を満たしていない．B-OPT は（2）を，OPT-DLOCK は（1）を満たさず，ARIES/IM は（3）を満たさない．よって，これらの並行性制御は並列 B-Tree である Fat-Btree に適さない．

2.2.3　INC-OPT 方式

Fat-Btree を含めた並列 B-Tree 向けの並行性制御方式として INC-OPT 方式は提案された [7], [10]．INC-OPT は条件 2.1 を満たしており，並列 B-Tree の性質に合致している．

INC-OPT の検索時のプロトコルは，IS モードによるラッチカップリングが使用される．まず，ルートページを IS モードでラッチした後，以下の処理を繰り返す．

（1） 親ページのキーを比較して，子ページのポインタを取得
（2） 子ページの IS ラッチを取り，親ページのラッチを解放

葉ページまで辿り着けば，葉ページに S ラッチを獲得しデータを読む．

INC-OPT での更新処理時のプロトコルは，二つのフェーズで行われる．

第 1 フェーズ: IX ラッチカップリングで葉ページまで辿る (葉ページは X ラッチが獲得される)．もし，SMO が起きないならば葉ページを更新して終了．もし葉ページがスプリットを起こし SMO が起こるならば，葉ページの X ラッチを解放し，第 2 フェーズに移行する．

第 2 フェーズ: 葉ページとその親ページを X モードでラッチする．もし親ページもスプリットするならば，同様にして X ラッチの範囲を拡大していく．十分な X ラッチが獲得されたならば更新操作を実行する．

この手続きの厳密な定義は [7], [10] を参照されたい．

INC-OPT は，更新処理のプロトコルからわかるように，SMO 発生時のリスタートが複数回必要になることがある．ルートページまで更新が及ぶ場合には，木の高さと同じフェーズ数が必要となる．このことは更新命令のレスポンスタイムを増加させるとともに，上位ページで複数回 X ラッチを獲得することで，システム全体のスループットも低下させる．

3. 提 案 手 法

3.1 MARK-OPT 方式

並列 B-Tree である Fat-Btree 向けの並行性制御として，MARK-OPT(MARKing OPTimistic) 方式を提案する．従来の INC-OPT は，広範囲に SMO が発生するときに，X ラッチ拡大のために複数回のリスタートが必要だった．それに対し MARK-OPT は，SMO が発生する木の高さをマークすることにより，広範囲の SMO 発生時も，ほとんどの場合 1 回のリスタートで更新フェーズに移ることができるので，リスタート回数を少なく抑えることが可能である．よって，リスタート回数を少なく抑えることによるレスポンスタイムの向上と，中間の X ラッチ拡大フェーズの除去による不必要な X ラッチの獲得の減少から，高更新環境において従来手法より高いスループットを得ることが期待できる．MARK-OPT は INC-OPT の拡張であるため，検索処理時のプロトコルは同様であるが，更新処理時のプロトコルが異なる．

MARK-OPT での更新処理時のプロトコルは次のように行われる．

第 1 フェーズ: IX ラッチカップリングで葉ページまで辿る (葉ページは X ラッチが獲得される)．フルエントリー (次にエントリーが増えたときにスプリットする状態のこと) では

ないインデックスページが存在したら，そのページのルートページからの高さをマークする．マークする高さは，フルエントリーではないページに遭遇するたびに逐次更新される．もし，SMOが起きないならば葉ページを更新して終了．もし葉ページがスプリットを起こし SMOが起こるならば，葉ページの X ラッチを解放し，第 2 フェーズに移行する．

第 2 フェーズ: 第 1 フェーズと同様に木の高さのマークは行う．前フェーズでマークした高さ以下のインデックスページと葉ページを X モードでラッチする．もし獲得した X ラッチの範囲が SMO の範囲に対して十分でなければ，同様にマークを用いて X ラッチの範囲を変化させていく．十分な X ラッチが獲得されたならば更新操作を実行する．

上記の手続きを厳密に定義する．木の高さを H とすれば，ページのレベル (h) はルートが 1，葉ページが H となる．また，l を X ラッチを獲得し始めるページのレベルとすれば，l は初め H にセットされる．そして，木の降下時にマークする値は m にセットされる．このとき，MARK-OPT プロトコルは図 2 で定義される．

```
↑   1   l := H;
    2   Parent := null; Child := ROOT; h := 1; m := 1;
    3   while h < l do begin
    4       Child に IX ラッチ，Parent のラッチ解放;
第  5       if Child がフルエントリーではない then
1   6           m := h;              /* マーキング */
フ  7       NewChild を決定;
ェ  8       Parent := Child; Child := NewChild; h := h + 1;
イ  9   end;
ズ  10  Child とそのコピーに X ラッチ;
    11  Parent のラッチ解放;
    12  if Child がフルエントリー then
    13      すべてのラッチを解放; l := m;
    14  else
↓   15      更新操作を実行; 全ラッチ解放し終了;
↑   16  Parent := null; Child := ROOT; h := 1; m := 1;
    17  while h < l do begin
    18      Child に IX ラッチ，Parent のラッチ解放;
    19      if Child がフルエントリーではない then
    20          m := h;              /* マーキング */
    21      NewChild を決定;
    22      Parent := Child; Child := NewChild; h := h + 1;
    23  end;
第  24  Child とそのコピーに X ラッチ;
2   25  Parent のラッチ解放;
フ  26  NewChild を決定;
ェ  27  Parent := Child; Child := NewChild; h := h + 1;
イ  28  while h ≦ H do begin
ズ  29      Child とそのコピーに X ラッチ;
    30      NewChild を決定;
    31      Parent := Child; Child := NewChild; h := h + 1;
    32  end;
    33  if 獲得した X ラッチが SMO に不十分 then
    34      すべてのラッチを解放; l := m; goto 15;
    35  else
↓   36      更新操作 (SMO) を実行; 全ラッチ解放し終了;
```

図 2　MARK-OPT プロトコル

MARK-OPT はマークした前フェーズの状態をもとにラッチ範囲を決定している．そのため，前フェーズからの木構造の変化による SMO 範囲の拡大により，複数回リスタートが必要となる場合もあるが，INC-OPT 同様に最大フェーズ数は高々H に抑えられる．SMO 範囲が複数のレベルに渡って同時に拡大することは極めて稀であり，多くの場合は 1 回のリスタートで処理を完了できる．よって，上位ページまで SMOが及ぶときでも，INC-OPT のように多くのリスタートを必要としない．

また，MARK-OPT は次の 3 点からわかるように条件 2.1 を満たしている．

（1） トップダウンにラッチを獲得するためデッドロックフリーである

（2） S，SIX ラッチをインデックスページには使用せず，SMO に関係しない不必要な X ラッチも獲得しない

（3） 木全体のラッチは存在しない

すなわち，MARK-OPT プロトコルは並列 B-Tree の更新の性質に合致している．

3.2　MARK-OPT 方式の拡張

MARK-OPT 方式の拡張として，INC-MARK-OPT 方式，2P-INT-MARK-OPT 方式および 2P-REP-MARK-OPT 方式の三つの並行性制御方式を提案する．各方式は更新処理プロトコルの第 2 フェーズのみが異なる．MARK-OPT では，木構造が変化しても処理に変更がない．それに対し拡張方式は，そのフェーズで最初に X ラッチ獲得するページの状態から，関連する部分木の前フェーズから構造変化の有無の判断を行う．それによって，木の構造が変化していたと判断された (最上位ページの状態が変化した) 場合は各方式で異なる処理を行う．その処理の違いをまとめたものが表 2 である．行は木構造が変化したと判断された以降に行う処理フェーズから区分し，列はそのときのリスタートの有無から区分されている．

表 2　フェーズ間の木構造変化に対する処理による各手法の比較

	リスタートしない	リスタートする
第 2 フェーズを続行	MARK-OPT/INC-OPT	INC-MARK-OPT
第 1 フェーズに移行	2P-INT-MARK-OPT	2P-REP-MARK-OPT

しかし，常に木の高さのマーキングは行う等，基本的な処理は MARK-OPT と同じである．よって，条件 2.1 は満たしており，MARK-OPT と同様に並列 B-Tree に適した並行性制御であるといえる．

3.2.1　INC-MARK-OPT 方式

INC-MARK-OPT(INCremental MARKing OPTimistic)方式は，第 2 フェーズにおいて木の構造が変化したと判断された場合にリスタートを行う．葉ページまで辿っていないので，次のフェーズで用いる状態マーク情報は完全ではないが，そこまでの情報をもとに X ラッチの範囲が決定する．

INC-MARK-OPT での更新処理時のプロトコルの第 2 フェーズは次のようになる．まず，前フェーズでマークした高さのインデックスページを X モードでラッチする．そのページがフルエントリーでなければ，MARK-OPT と同様の処理を葉ページまで行う．もしページがフルエントリーであればリスタートし，十分な X ラッチを獲得するまで第 2 フェーズの処理を繰り返す．図 2 での定義と同じ変数を用いて，この手続きを厳密に定義したものが図 3 である．枠内が MARK-OPT からの追加部分である．

INC-MARK-OPT は，木の構造が変化したと判断されたら直ちにリスタートするので，実際に SMO の範囲が拡大している場合には，MARK-OPT より不必要な X ラッチを獲得しない．一方で，SMO の範囲が縮小しているときも，木の構造が変化したと判断されるので，その場合不必要な X ラッチの獲得が増

```
↑   1  l := H;
    2  Parent := null; Child := ROOT; h := 1; m := 1;
    3  while h < l do begin
    4      Child に IX ラッチ，Parent のラッチ解放;
   第 5      if Child がフルエントリーではない then
   1 6          m := h;              /* マーキング */
   フ 7      NewChild を決定;
   ェ 8      Parent := Child; Child := NewChild; h := h + 1;
   ｜ 9  end;
   ズ 10 Child とそのコピーに X ラッチ;
   11 Parent のラッチ解放;
   12 if Child がフルエントリー then
   13     すべてのラッチを解放; l := m;
   14 else
↓  15     更新操作を実行; 全ラッチ解放し終了;
↑  16 Parent := null; Child := ROOT; h := 1; m := 1;
   17 while h < l do begin
   18     Child に IX ラッチ，Parent のラッチ解放;
   19     if Child がフルエントリーではない then
   20         m := h;              /* マーキング */
   21     NewChild を決定;
   22     Parent := Child; Child := NewChild; h := h + 1;
   23 end;
   第 24 Child とそのコピーに X ラッチ;
   2 25 Parent のラッチ解放;
   フ 26 if Child がフルエントリーである then
   ェ 27     すべてのラッチを解放; l := m; goto 16;
   ｜ 28 NewChild を決定;
   ズ 29 Parent := Child; Child := NewChild; h := h + 1;
   30 while h ≦ H do begin
   31     Child とそのコピーに X ラッチ;
   32     NewChild を決定;
   33     Parent := Child; Child := NewChild; h := h + 1;
   34 end;
   35 if 獲得した X ラッチが SMO に不十分 then
   36     すべてのラッチを解放; l := m; goto 15;
   37 else
↓  38     更新操作 (SMO) を実行; 全ラッチ解放し終了;
```

図 3　INC-MARK-OPT プロトコル

```
↑   1  l := H;
    2  Parent := null; Child := ROOT; h := 1; m := 1;
    3  while h < l do begin
    4      Child に IX ラッチ，Parent のラッチ解放;
   第 5      if Child がフルエントリーではない then
   1 6          m := h;              /* マーキング */
   フ 7      NewChild を決定;
   ェ 8      Parent := Child; Child := NewChild; h := h + 1;
   ｜ 9  end;
   ズ 10 Child とそのコピーに X ラッチ;
   11 Parent のラッチ解放;
   12 if Child がフルエントリー then
   13     すべてのラッチを解放; l := m;
   14 else
↓  15     更新操作を実行; 全ラッチ解放し終了;
↑  16 Parent := null; Child := ROOT; h := 1; m := 1;
   17 while h < l do begin
   18     Child に IX ラッチ，Parent のラッチ解放;
   19     if Child がフルエントリーではない then
   20         m := h;              /* マーキング */
   21     NewChild を決定;
   22     Parent := Child; Child := NewChild; h := h + 1;
   23 end;
   第 24 Child とそのコピーに X ラッチ;
   2 25 Parent のラッチ解放;
   フ 26 if Child がフルエントリーである then
   ェ 27     l := H; goto 7;
   ｜ 28 NewChild を決定;
   ズ 29 Parent := Child; Child := NewChild; h := h + 1;
   30 while h ≦ H do begin
   31     Child とそのコピーに X ラッチ;
   32     NewChild を決定;
   33     Parent := Child; Child := NewChild; h := h + 1;
   34 end;
   35 if 獲得した X ラッチが SMO に不十分 then
   36     すべてのラッチを解放; l := m; goto 15;
   37 else
↓  38     更新操作 (SMO) を実行; 全ラッチ解放し終了;
```

図 4　2P-INT-MARK-OPT プロトコル

```
↑   1  l := H;
    2  Parent := null; Child := ROOT; h := 1; m := 1;
    3  while h < l do begin
    4      Child に IX ラッチ，Parent のラッチ解放;
   第 5      if Child がフルエントリーではない then
   1 6          m := h;              /* マーキング */
   フ 7      NewChild を決定;
   ェ 8      Parent := Child; Child := NewChild; h := h + 1;
   ｜ 9  end;
   ズ 10 Child とそのコピーに X ラッチ;
   11 Parent のラッチ解放;
   12 if Child がフルエントリー then
   13     すべてのラッチを解放; l := m;
   14 else
↓  15     更新操作を実行; 全ラッチ解放し終了;
↑  16 Parent := null; Child := ROOT; h := 1; m := 1;
   17 while h < l do begin
   18     Child に IX ラッチ，Parent のラッチ解放;
   19     if Child がフルエントリーではない then
   20         m := h;              /* マーキング */
   21     NewChild を決定;
   22     Parent := Child; Child := NewChild; h := h + 1;
   23 end;
   第 24 Child とそのコピーに X ラッチ;
   2 25 Parent のラッチ解放;
   フ 26 if Child がフルエントリーである then
   ェ 27     すべてのラッチを解放; goto 1;
   ｜ 28 NewChild を決定;
   ズ 29 Parent := Child; Child := NewChild; h := h + 1;
   30 while h ≦ H do begin
   31     Child とそのコピーに X ラッチ;
   32     NewChild を決定;
   33     Parent := Child; Child := NewChild; h := h + 1;
   34 end;
   35 if 獲得した X ラッチが SMO に不十分 then
   36     すべてのラッチを解放; l := m; goto 15;
   37 else
↓  38     更新操作 (SMO) を実行; 全ラッチ解放し終了;
```

図 5　2P-REP-MARK-OPT プロトコル

える．また，リスタート回数は多くなるが，獲得する X ラッチの範囲は拡大していくので，MARK-OPT と同様に最大フェーズ数は高々H である．

3.2.2　2P-INT-MARK-OPT 方式

2P-INT-MARK-OPT(2-Phase INTegrated MARKing OPTimistic) 方式は，木の構造が変化したと判断された場合に，そのページ以下のページでは，IX ラッチによるラッチカップリングを行う．すなわち，第 1 フェーズに戻るということである．木の状態のマークは第 2 フェーズでも行っているので，マーキング情報が不完全になるという問題が発生することなく第 1 フェーズに移行できる．

2P-INT-MARK-OPTでの更新処理時のプロトコルの第 2 フェーズは次のようになる．まず，前フェーズでマークした高さのインデックスページで X ラッチを獲得する．そのページがフルエントリーでなければ，MARK-OPT と同様の処理を葉ページまで行う．もしページがフルエントリーであれば，そのページから第 1 フェーズに戻る．図 2 と同じ変数を用いて，この手続きを厳密に定義したものが図 4 である．枠内が MARK-OPT からの追加部分である．

2P-INT-MARK-OPT は，木の構造が変化したと判断されてもリスタートは行わず，第 1 フェーズにフェーズチェンジ (プロトコルのフェーズを移行すること) し処理を行う．よって，INC-MARK-OPT よりリスタート回数は少なく，MARK-OPT より不必要な X ラッチの獲得も行わない．また，INC-MARK-OPT と違い，SMO の範囲が縮小した場合にも，インデックスで不必要な X ラッチの獲得は行わない．一方で，木の高さ H を超えるリスタートが必要となることもあり，レスポンスタイムが著しく悪化することも考えられるが，そのようなことはほとんど起こりえない．また，最悪の場合は処理が終わらないが，そのようなことは起こりえないので，SMO 発生時も少数のリスタートで済む．

3.2.3　2P-REP-MARK-OPT 方式

2P-REP-MARK-OPT(2-Phase REPetitive MARKing OPTimistic) 方式は，木の構造が変化したと判断された場合にリスタートを行う．INC-MARK-OPT と異なり，リスタート後は第 1 フェーズに戻って処理を行う．

2P-REP-MARK-OPT での更新処理時のプロトコルの第 2 フェーズは次のようになる．まず，前フェーズでマークした高さのインデックスページを X モードでラッチする．そのページがフルエントリーでなければ，MARK-OPT と同様の処理を葉ページまで行う．もしページがフルエントリーであればリスタートを行い，ルートページから第 1 フェーズの処理を行う．図 2 と同じ変数を用いて，この手続きを厳密に定義したものが図 5 である．枠内が MARK-OPT からの追加部分である．

2P-REP-MARK-OPT は木の構造が変化したと判断されてたら，すぐにリスタートし，その後第 1 フェーズに戻り処理を行う．よって，提案手法の中で不必要な X ラッチの獲得は最も行わないが，リスタート回数は最も多くなる．一方で，2P-INT-MARK-OPT と同様，木の高さ H を超えるリスタートが必要となることもあるが，そのようなことはほとんど起こりえないので，SMO 発生時も少数のリスタートで済む．

3.3　提案方式の比較

各提案方式間で更新処理時における特徴の比較を行う．リスタート回数と 1 フェーズあたりの X ラッチ獲得時間から 1 処理での X ラッチ獲得時間が求まり，その時間というものが提案方式のアルゴリズムの性能に非常に影響する．よって，ここでは

リスタート回数および 1 フェーズあたりの X ラッチ獲得時間により各提案方式の比較を行う．

まず，リスタート回数で各提案方式を比較する．MARK-OPT より，拡張方式はリスタート回数が多くなり，その中でも 2P-REP-MARK-OPT は最も多くなる．

続いて，1 フェーズあたりの X ラッチ獲得時間で各提案方式を比較する．各拡張方式は，最終的に更新が行われるフェーズ以外に複数のページで同時 X ラッチを獲得するフェーズは存在しない．よって，MARK-OPT と比較して，拡張方式は 1 フェーズあたりの X ラッチ獲得時間は短くなる．INC-MARK-OPT は他の拡張方式より上位ページから X ラッチを獲得する機会が多くなるので，X ラッチ期間は長くなる．

また，実験による提案手法間の比較を 4.6 節にて行う．

4. 実　　験

提案する新たな並行性制御 MARK-OPT とその拡張方式の有効性を示すために，Fat-Btree を採用している自律ディスク [15]〜[17] に提案手法を実装し実験を行う．

4.1 実験環境

実験は，我々の提案する分散ストレージ技術である自律ディスクの模擬実装上で行う．これは Linux クラスタ上に Java を用いて模擬実装されている．今回の実験では表 3 に示す構成の PC と十分なバックボーン性能を持つネットワークスイッチを用いて，実験環境を構成した．

表 3　実験システムの構成

ノード	32-128 台 (Storage)，32 台 (Clients)
CPU	AMD Athlon XP-M1800+ (1.53GHz)
メモリ	PC2100 DDR SDRAM 1GB
ディスク	TOSHIBA MK3019GAX (30GB，5400rpm，2.5inch)
OS	Linux 2.4.20
Java VM	Sun J2SE SDK 1.4.2_04 Server VM

4.1.1 初期 Fat-Btree の構築

本来自律ディスクにはデータマイグレーションによる負荷分散機能 [16], [17] を備えているが，今回の実験用の自律ディスクには実装されていない．よって，Fat-Btree の葉ページを移動させることによる動的な負荷分散 [11] を行うことはできない．この場合，どのようにして挿入するタプルを均等に PE に分散させるかが問題になってくる．そこで今回は初期 Fat-Btree として各 PE に葉ページを 1 ページずつ作成し，これらの葉ページのキーをその PE に格納されるキーの下限値とする方法をとった．PE 番号の昇順に，初期葉ページのキーの値が大きくなるようにしておくことにより，以後の挿入処理では各 PE に格納される葉ページが静的に分割される．この初期 Fat-Btree に対してランダムな要素を繰り返し挿入したものを実験に用いる．今回の実験における Fat-Btree の構成パラメータは表 4 に示す．このようなパラメータに設定したのは，提案手法の効果を明確にするためである．

4.2 実験方法

このような環境下の自律ディスク上に上記の手法で初期 Fat-Btree を構築した後，各クライアント PC から同時にリクエストを送信する．キーはランダムに選び，検索と挿入操作を実行したときのスループットから性能を評価する．操作を送るのは，ランダムに選択した PE(自律ディスク) に対してである．

表 4　実験システムのパラメータ

ページサイズ	4KB
タプルサイズ	400B
インデックスページのキー数	最大 4-64
葉ページのタプル数	最大 1-8

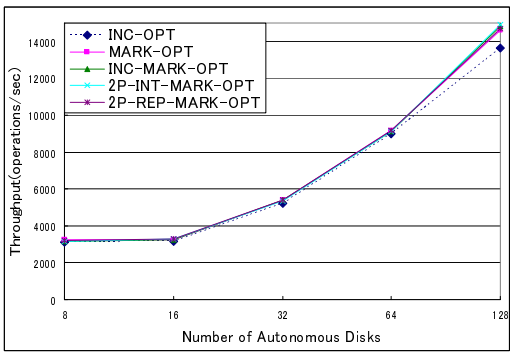

図 6　自律ディスクの台数を変化させたときの並行性制御方式の比較

以下に本実験の概要を述べる．まず，自律ディスクの台数を変化させたときの各手法のスループットを測定し，従来手法と提案手法の比較を行う．次に，更新要求の割合を変化させたときのスループットの測定を行う．また，リスタート回数の測定も行い，各手法間の比較を行う．最後に，SMO 発生が多い環境におけるスループット，レスポンスタイム，リスタートおよび X ラッチ獲得の回数の測定を行い，各手法間の比較を行う．

4.3 自律ディスクの台数を変化させたときの比較

32 台のクライアント PC(1 台あたり並行して 4 スレッド) からリクエストを送信し，10 秒間操作したときのスループットを測定した．図 6 は，更新比率 20%，自律ディスク 1 台数あたりのデータベースサイズ 3200 タプルのときの Fat-Btree を，横軸として自律ディスクの台数を 8 から 128 に変化させ，スループットを縦軸として比較したグラフである．破線グラフは INC-OPT の値を表し，棒線グラフは提案 4 手法の値を表し，提案 4 手法はほぼ同じ値である．

INC-OPT は，台数増加に伴うスループットの上昇が小さい．これは，上位ページに SMO が及ぶ場合複数回のリスタートが発生し，毎フェーズ X ラッチを獲得することによる PE 間同期オーバーヘッドが大きくなるからである．それに対し，提案手法は上位ページに SMO が及ぶときでも，ほとんどの場合 1 回のリスタートのみで済み，X ラッチの獲得は最小限になるので，PE 間同期オーバーヘッドが小さい．よって，提案手法は台数増加に伴うスループットの上昇が大きい．

4.4 更新要求の割合を変化させたときの比較

32 台のクライアント PC(1 台あたり並行して 4 スレッド) からリクエストを送信し，10 秒間操作したときのスループットを測定した．図 7 は，自律ディスク台数 64，自律ディスク 1 台数あたりのデータベースサイズ 3200 タプルのときの Fat-Btree を，横軸として更新操作の割合を 0%から 100%に変化させ，INC-OPT と提案 4 手法を，スループットを縦軸として比較したグラフである．破線グラフは INC-OPT の値を表し，棒線グラ

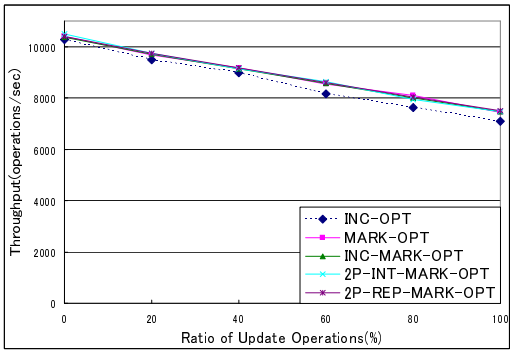

図 7　更新要求の割合を変化させたときの並行性制御方式の比較

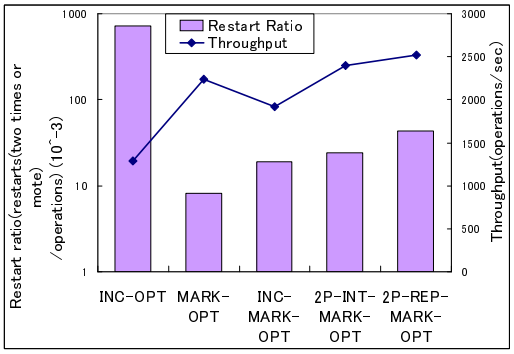

図 9　SMO 発生が多い環境における各提案方式の比較

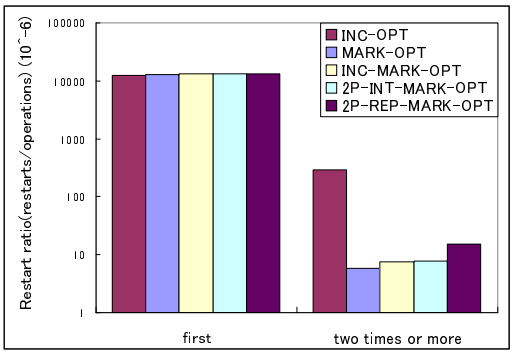

図 8　並行性制御方式のリスタート率の比較

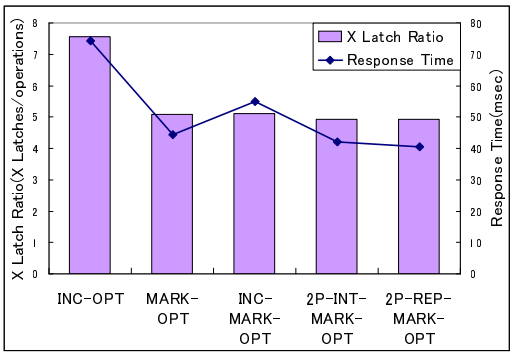

図 10　レスポンスタイムと X ラッチ獲得率による各提案方式の比較

フは提案 4 手法の値を表し，提案 4 手法はほぼ同じ値である．

更新要求の割合が 20%や 40%ときには，各方式の間でスループットの差は小さい．これは，要求に占める割合が大きい検索に対して，各方式とも同じ処理を行うためである．

更新要求の割合が増えるに従い，INC-OPT のスループットが低下していくのに対し，提案手法も徐々にスループットは低下するものの，INC-OPT よりは高いスループットを維持している．これは，INC-OPT と比べて，提案手法が SMO 発生時のリスタート回数を少なく抑えているからである．また，提案 4 方式間でスループットの差異は見られなかった．

4.5　並行性制御方式によるリスタート率の比較

4.4 節の実験において，リスタート回数の測定も行った．図 8 は，更新操作の割合 100%，自律ディスク台数 64，自律ディスク 1 台数あたりのデータベースサイズ 3200 タプルのときの Fat-Btree の，1 命令あたりのリスタート率を比較したグラフである．左のグラフは各更新処理において最初に発生したリスタートの回数，右のグラフは 2 回目以降のリスタートの回数を全命令数で割った値である．

どの並行性制御方式も初回のリスタート率はほぼ等しい．これは，更新プロトコルの第 1 フェーズの処理に差異がないからである．

2 回目以降のリスタート率は，X ラッチの範囲を徐々に拡大していく INC-OPT だけが多い．一方，提案方式はどの方式も同様に少ない．相対的には，MARK-OPT，INC-MARK-OPT と 2P-INT-MARK-OPT，2P-REP-MARK-OPT の順に率は高い．

このように，再リスタートが少ないということは，木の状態のマーキングによる X ラッチ範囲の決定が有効であること示す．また，各提案方式で実験結果にほとんど差異がなかったのは，2 回目以降のリスタートの絶対数が非常に少なく，性能にほとんど影響がでなかったからである．拡張方式は，SMO が多く起こる環境を想定して提案したものであるため，そうでない場合，各方式に差異はでない．

4.6　SMO 発生が多い環境における比較

これまで SMO の発生が少ない環境での実験を行ってきたため，提案手法間でスループットの差がほとんどなかった．現実的ではないが，提案手法間の比較を行うために SMOが多発する環境で実験を行う．そのような環境を実現するために，Fat-Btree の 1 ページあたりのエントリー数を減らし，クライアントの並行スレッド数を増やすことでアクセス負荷を大きくし，更新比率を 100%とした．

32 台のクライアント PC(1 台あたり並行して 16 スレッド) からリクエストを送信し，2 分間操作したときのスループットおよびリクエスト回数を測定した．図 9 は，更新比率 100%，自律ディスク台数 64，自律ディスク 1 台数あたりのデータベースサイズ 33 タプル，インデックスページのキー数 4，葉ページのタプル数 1 のときの Fat-Btree，提案 4 手法のスループットと 1 命令あたりのリスタート回数 (2 回目以降) を縦軸として比較したグラフで，図 10 は，レスポンスタイムと 1 命令あたりの X ラッチの獲得回数を縦軸として比較したグラフである．今実験のレスポンスタイムが悪いのは SMO の発生が多い環境で行われているためであり，通常の環境でのレスポンスタイムは 4-5msec 程度である．

INC-OPT は，他の実験と同様に各提案手法よりスループットが低い．これは，今実験の環境においても，提案手法と比較し

て複数回リスタートが非常に多いからである．

提案手法間の比較では，木構造の変化に対してリスタートをしない MARK-OPT が，複数回リスタートが最も少ない．直ちにリスタートする INC-MARK-OPT とリスタートはしないが第 1 フェーズに戻る 2P-INT-MARK-OPT がほぼ同じ回数で続く．そして，リスタートかつ第 1 フェーズに戻る 2P-REP-MARK-OPT が提案手法の中で最も多い．しかし，リスタート回数が最も多い 2P-REP-MARK-OPT のスループットが最も高く，2P-INT-MARK-OPT, MARK-OPT の順にわずかの差で続き．INC-MARK-OPT は，リスタート回数は 2P-INT-MARK-OPT と同程度だが，提案手法の中ではスループットが最も低い．図 10 より X ラッチの獲得回数の差がわずかであることから，この結果は X ラッチ 1 つあたりの獲得時間の差によるものであると思われる．

以上より提案手法の間の差はわずかであるが，SMO が多発する場合は 2P-REP-MARK-OPT が一番有効であることと言える．

5. まとめと今後の課題

本稿では，無共有並列計算機に適したディレクトリ構造である並列 B-Tree の並行性制御として新手法，MARK-OPT を提案した．これは，SMO 発生によりリスタートが必要となったとき，前フェーズの木の状態に基づき X ラッチの獲得開始位置をマークし，リスタート後の SMO に必要な X ラッチの範囲を決定する並行性制御方式である．また，MARK-OPT の拡張として 3 つの手法を提案した．さらに自律ディスク上での実験により，4 つの提案手法と従来手法との比較を行い，提案手法の有効性を確認し，リスタート回数と X ラッチの獲得時間の関係から 2P-REP-MARK-OPT が最も優れていることを示した．

本稿ではアクセスパターンが各 PE に対して均一であると仮定したが，実際は偏りがある場合が多い．そのときには，自律ディスクの機能である自律的なデータ移動による負荷分散機能も用いて実験を行う必要がある．また，負荷分散時の並行性制御についても検討しなければならない．

さらに，優れた並行性制御を実現できることが知られている B-link [20], [21] の Fat-Btree への適用を検討している．B-link は，サイドポインタにより隣のインデックスノードにリンクをもっている．サイドポインタがあることにより，ラッチカップリングを用いず，単一ノードラッチによる並行性制御を行うことができる．また，リスタートを行うことはなく，すべての処理が 1 フェーズで行われる．B-link の Fat-Btree への適用方法を検討し，適用したものと従来手法との比較が必要である．

謝　　辞

Fat-Btree の並行性制御に関して奈良先端科学技術大学の宮崎純助教授に助言を頂いた．ここに感謝の意を表します．また本研究の一部は，独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST，情報ストレージ研究推進機構 (SRC)，文部科学省科学研究費補助金特定領域研究 (16016232) および東京工業大学 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤構築」の助成により行なわれた．

文　　献

[1] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data Placement in Bubba. In *Proc. of ACM SIGMOD conf. '88*.

[2] S. Ghandeharizadeh and D. J. DeWitt. HybridRange Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. In *Proc. of VLDB Conf. '90*, 1990.

[3] D. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. In *Communications of the ACM*, Vol. 35, pp. 85–98, Jun. 1992.

[4] 金政泰彦 , 宮崎純 , 横田治夫. 並列データベースシステムにおける更新を考慮したディレクトリ構成. 信学技報，電子情報通信学会，DE97-77(AI97-44). 電子情報通信学会, 1997.

[5] Haruo YOKOTA, Yasuhiko KANEMASA, and Jun MIYAZAKI. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *Proc. of IEEE ICDE Conf. '99*, pp. 448–457, Mar. 1999.

[6] 宮崎純 , 横田治夫. 無共有並列計算機向けディレクトリ構造 Fat-Btree の実装とその評価. 情処学会研究会報告，データベースシステム DBS-119-68, pp. 407–412. 情報処理学会, Jul. 1999.

[7] 宮崎純 , 横田治夫. 並列ディレクトリ構造 Fat-Btree の並行性制御とその評価. 情処学会研究会報告，データベースシステム DBS-124-69, pp. 37–44. 情報処理学会, Jul. 2001.

[8] 宮崎純 , 横田治夫. 並列ディレクトリ Fat-Btree のリカバリについて. 信学技報，電子情報通信学会，DE2001-107, pp. 17–24. 電子情報通信学会, Oct. 2001.

[9] 宮崎純 , 横田治夫. 高信頼 Fat-Btree 構成への neighbor-WAL プロトコルの適用. 第 13 回電子情報通信学会データ工学ワークショップ論文集，DEWS2002，C1-2. 電子情報通信学会, Mar. 2002.

[10] Jun MIYAZAKI and Haruo YOKOTA. Concurrency Control and Performance Evaluation of Parallel B-tree Structures. In *IEICE Transactions on Information and Systems*, Vol. E85-D, pp. 1269–1283. The Institute of Electronics, Information and Communication Engineers, Aug. 2002.

[11] 鈴木裕通 , 横田治夫. 並列ディレクトリ構造 Fat-Btree における負荷分散の手法とその実装. 第 11 回電子情報通信学会データ工学ワークショップ論文集，DEWS2000，4B-4. 電子情報通信学会, Mar. 2000.

[12] 風戸広史 , 横田治夫. 並列ディレクトリ構造 Fat-Btree におけるレンジ問い合わせの取り扱い. 第 12 回電子情報通信学会データ工学ワークショップ論文集，DEWS2001，7A-7. 電子情報通信学会, Mar. 2001.

[13] R. Bayer and M. Schkolnick. Concurrency of Operations on B-trees. In *Acta Informatica*, Vol. 9, pp. 1–21, 1977.

[14] C. Mohan and F. Levine. ARIES/IM: an Efficient and High Concurrency Index Management Method Using Write-Ahead Logging. In *Proc. of ACM SIGMOD Conf. '92*, pp. 371–381, 1992.

[15] Haruo Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, pp. 441–448, Nov. 1999.

[16] 風戸広史 , 横田治夫. 自律ディスクへの Fat-Btree の実装. 情処学会研究会報告，データベースシステム DBS-125-71, pp. 45–52. 情報処理学会, Jul. 2001.

[17] 伊藤大輔 , 風戸広史 , 横田治夫. 負荷分散機構と組み合わせた自律ディスクのクラスタ再構築. 信学技報，電子情報通信学会，FTS2001-20, pp. 119–126. 電子情報通信学会, Jul. 2001.

[18] J. Gray and A. Reuter. *Transaction Proceeding: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 1993.

[19] V. Srinivasan and M. J. Carey. Performance of B-Tree Concurrency Control Algorithms. In *Proc. of ACM SIGMOD Conf. '91*, pp. 416–425, 1991.

[20] P. L. Lehman and S. B. Yao. Efficient Locking for Concurrent Operations on B-trees. In *ACM Trans. Database Syst.*, Vol. 6, pp. 650–670, 1981.

[21] V. Lanin and D. Shasha. A Symmetric Concurrent B-tree Algorithm. In *Proc. FJCC*, pp. 380–389, 1986.