並列Btree構造Fat-Btreeにおけるリクエスト委譲コストを削減する並行性制御手法

吉原　朋宏[†]　小林　大[†]　田口　亮[††]　横田　治夫[†††,†]

† 東京工業大学大学院情報理工学研究科計算工学専攻　〒 152–8552 東京都目黒区大岡山 2–12–1
†† 日本放送協会放送技術研究所　〒 157–8510 東京都世田谷区砧 1–10–11
††† 東京工業大学学術国際情報センター　〒 152–8550 東京都目黒区大岡山 2–12–1
E-mail: †{yoshihara,daik}@de.cs.titech.ac.jp, ††taguchi.r-cs@nhk.or.jp, †††, †yokota@cs.titech.ac.jp

あらまし　Fat-Btreeに適した並行性制御手法として提案してきたMARK-OPTは，楽観的処理中にラッチカップリングを行うため，Fat-Btreeにおけるリクエスト委譲時のコストが高い．本稿では，楽観的処理中にラッチカップリングを行わないことで，Fat-BtreeにおいてMARK-OPTより高い処理性能を得ることが可能な新手法を提案する．提案手法では，非ラッチカップリングに伴う経路誤りを検出するため，ページ両端に境界値を設けている．また，Fat-Btreeでは，ページスプリット時にもページの削除が発生する可能性があるが，削除ページと非削除ページを交換することにより，通常の削除より効率的な処理を行う．Fat-Btreeを採用している自律ディスクに提案手法を実装し，様々な環境における実験から，提案手法が常にシステムスループットを改善し，大規模システムおよび高アクセス負荷環境において特に有効であることを示す．

キーワード　アクセスパス，並行制御とリカバリ，並列・分散DB

A Concurrency Control Method to Reduce the Cost of Request Transfers on the Fat-Btree, Parallel Btree Structure

Tomohiro YOSHIHARA[†], Dai KOBAYASHI[†], Ryo TAGUCHI[††], and Haruo YOKOTA[†††,†]

† Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
2–12–1 Ookayama, Meguro-ku, Tokyo 152–8552, Japan
†† Science and Technical Research Laboratories, Japan Broadcasting Corporation
1–10–11 Kinuta, Setagaya-ku, Tokyo 157–8510, Japan
††† Global Scientific Information and Computing Center, Tokyo Institute of Technology
2–12–1 Ookayama, Meguro-ku, Tokyo 152–8550, Japan
E-mail: †{yoshihara,daik}@de.cs.titech.ac.jp, ††taguchi.r-cs@nhk.or.jp, †††, †yokota@cs.titech.ac.jp

Abstract　As an effective concurrency control method for the Fat-Btree, the MARK-OPT method has been proposed. However, the latch-couplings in the method enlarge the costs of request transfers on the Fat-Btree. In this paper, a new concurrency control method not latch-coupling during optimistic operations latch coupling is proposed to improve the system performance. To detect access path errors in the proposed method, the index pages have boundary values at both ends. In the Fat-Btree, a page split may cause a page deletion. Swap of deleted page and non-deleted page in the method is more effective than normal page deletion. To compare the performance of the proposed method and the MARK-OPT method, we implemented them on an autonomous-disk system adopting the Fat-Btree. The experimental results on varied environments indicate that the proposed method always improves the system throughput, and are especially effective for large scale configuration and high workloads.

Key words　Access Path, Concurrency Control and Recovery, Parallel · Distributed DB

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 Btree を用いる研究がある．並列 Btree 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 Btree 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 Btree 構造として Fat-Btree が提案されている [4]〜[8]．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 [7] 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 Btree 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 [9]．無共有並列計算機向けに設計されている INC-OPT は，従来の Btree の並行性制御方式である B-OPT [10] や ARIES/IM [11] 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 [9]．しかし，INC-OPT は Btree 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の再降下（リスタート）が多数必要となり，システム処理性能の低下をもたらす．

そこで我々は，並列 Btree 構造の新たな並行性制御として，INC-OPT を改良した MARK-OPT 方式を提案している [12]．MARK-OPT は，楽観的なラッチカップリング中に SMO が発生するポイントのマーキングを行う．これにより，従来の INC-OPT と比較して，SMO 発生時のリスタート回数を少なく抑え，SMO のための排他ラッチ獲得時間を短縮することができる．分散ディレクトリ構造として並列 Btree である Fat-Btree を用いている自律ディスク [13]〜[15] 上での実験から，MARK-OPT が並列 Btree 上において優れた性能を示すことが明らかにされている [12]．また，データマイグレーションは，アクセス偏り除去のための有効な手段であるが，MARK-OPT がデータマイグレーションの並行性制御としても有効であることも明らかにされている [16]．しかし，MARK-OPT は木の降下中にラッチカップリングを用いているため，Fat-Btree では，他の PE にリクエスト委譲が行われるとき，3 回のネットワークを介したメッセージ転送が必要となる．このように，分散環境でラッチカップリングを用いることはコストが大きい．また，システム規模が大きくなるに従い，リクエスト委譲の頻度は大きくなるため，規模拡大によるスループット向上が小さくなる．

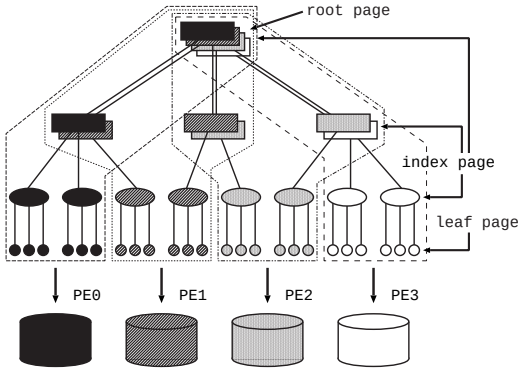


図 1　Fat-Btree 構造

本稿では，MARK-OPT と比べて，リクエスト委譲時のメッセージ転送回数の削減することが可能である Fat-Btree に適した並行性制御方式を提案する．また，自律ディスク上に MARK-OPT および提案手法を実装し，システム規模を変化させた場合，更新要求の割合を変化させた場合およびクライアントのスレッド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．

2. Fat-Btree と並行性制御

2.1 Fat-Btree 構造

Fat-Btree [4]〜[8] は，B^+-tree 全体をページ単位で PE 間で分配する並列 Btree の一種であり，データページである Btree 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 Btree 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，Btree のルートページ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．高い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 Btree 構造と比較して高速に行

表 1　ラッチマトリクス

Mode	IS	IX	S	SIX	X
IS	○	○	○	○	
IX	○	○			
S	○		○		
SIX	○				
X					

うことができる．

2.2　リクエスト委譲

Fat-Btree では，検索処理を実行する PE 上にディレクトリが一部しか存在しない．もし，必要なローカルインデックスページがその PE 上に存在しない場合は，そのインデックスページを格納している PE にリクエストを委譲するのだが，そのためには，他 PE のリモートページへのポインタを保持する必要がある．また，リクエスト委譲時，各 PE 同士はネットワークによって接続されているため，ネットワークを介したメッセージ転送が発生することになる．

2.3　並行性制御

木構造の一貫性を保証するために，Btree に並行性制御は必須である．通常，Btree および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．

2.3.1　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 [17]．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．IS，S モードのラッチは，検索処理に用いられ，S とは共有を意味する．IX，X モードのラッチは，更新処理に用いられ，X とは排他を意味する．SIX モードのラッチは，本稿では用いられない．

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，コピーを持つすべての PE でラッチを獲得する必要がある．

2.3.2　Fat-Btree の並行性制御

Fat-Btree を含む並列 Btree 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 [9], [12]．

［条件 2.1］ 並列 Btree 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 Btree の並行性制御方式として，B-OPT [10]，OPT-DLOCK [18] および ARIES/IM [11] 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，これらは Fat-Btree に適さない．

上記の条件を満たす並行性制御方式として INC-OPT 方式が提案されている [9]．Fat-Btree を含めた並列 Btree に適した並行性制御方式として設計されている INC-OPT 方式は，従来の Btree の並行性制御方式である B-OPT [10] や ARIES/IM [11] 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 [9]．

しかし，INC-OPT は，SMO 発生時にリスタートを繰り返しながらラッチ範囲を順次広げていくという手順をとるため，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．

2.4　MARK-OPT 方式

MARK-OPT（MARKing OPTimistic）方式 [12] は，並列 Btree 向けとして我々が提案した並行性制御手法である．従来の INC-OPT は，広範囲に SMO が発生するとき，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，広範囲に SMO が発生するときでも，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 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(注1)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．

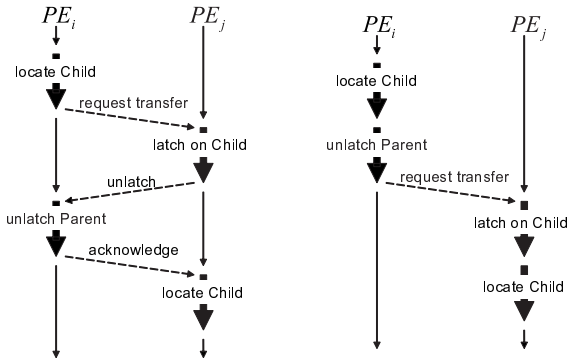
（1） 親ページのキーを比較して，子ページのポインタを取得
（2） 子ページの IS ラッチを取り，親ページのラッチを解放

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．

MARK-OPT での更新処理時のプロトコルは，次のように二つのフェーズで行われる．

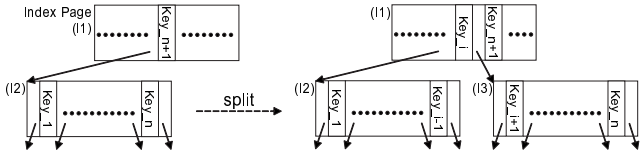
第 1 フェーズ： IX ラッチカップリングで葉ページまで辿る（葉ページは X ラッチが獲得される）．フルエントリー（次にエントリーが増えたときにスプリットする状態のこと）ではないインデックスページが存在したら，そのページのルートページからの高さをマークする．マークする高さは，フルエントリーではないページに遭遇するたびに逐次更新される．もし，SMO が起きないならば葉ページを更新して終了．もし葉ページがスプリットを起こし SMO が起こるならば，葉ページの X ラッチを解放し，第 2 フェーズに移行する．

(注1)：[17] においては，ラッチカップリングはクラビングと呼ばれている．



（a）ラッチカップリング使用時　（b）ラッチカップリング不使用時

図 2　リクエスト委譲プロトコル



（a）スプリット前　（b）スプリット後

図 3　インデックスページのスプリット

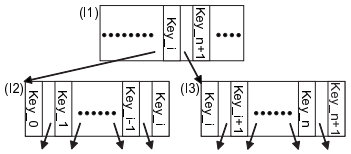


図 4　境界値のあるインデックスページのデータ構造

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．

この手続きの厳密な定義は [12] を参照されたい．

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木の高さ H に抑えられる．短期間で SMO 範囲が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．

3. 提 案 手 法

Fat-Btree 向けの並行性制御方式として提案されている MARK-OPT の Fat-Btree 上での性能を改善する並行性制御方式を提案する．

MARK-OPT では，目的の葉ページまで経路においてラッチカップリングが用いられる．そのため，Fat-Btree における PE 間でのリクエスト委譲が発生するとき，一回のリクエスト委譲ごとに，図 2（a）のように逐次的なネットワーク通信が三回必要となる．それに対し提案手法は，MARK-OPT で用いられているラッチカップリングを行わないことにより，図 2（b）のようにリクエスト委譲時のネットワーク通信を一回に抑えることができるので，ネットワークを介した通信回数を削減することが可能である．よって，ネットワークを介した通信回数を削減することによるレスポンスタイム向上が期待できる．

しかし，ラッチカップリング以外の基本的な処理は 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 Btree に適した並行性制御であるといえる．

3.1　経路誤りの検出

ラッチカップリングを用いない場合，誤った場所へのポインタに従う可能性がある．このとき，提案手法はルートページからリスタートを行う．

経路誤りが発生する手法として B-link [19], [20] を用いた手法や ARIES/IM [11] がある．B-link を用いた手法では，同時に 1 つのページにしかラッチを獲得しないため，経路誤りが発生する．このとき，B-link という同じ高さの兄弟ページをつなぐポインタによって正しい経路に復帰する．また，ARIES/IM は，目的の葉ページまでの経路ではラッチカップリングを行うが，SMO 発生時に，更新のための X ラッチ獲得にはラッチカップリングを用いないため，経路誤りが発生する．このとき，LSN（Log Sequence Number）の値が変更されていないページまで後退することで正しい経路に復帰する．これらの手法は，ボトムアップに SMO のための X ラッチを獲得し，更新を行うため，トップダウンに行われる木の降下中に多くの経路誤りが発生する．それに対し，提案手法はトップダウンに X ラッチを獲得するため，それらの手法より経路誤りが発生する確率が低い(注2)．

さらに，経路誤りが起こったときのレスポンスタイムの悪化も小さい．なぜなら，リスタート前のアクセスにより主記憶上にキャッシュされている確率が高いからである．また，リクエスト委譲の発生により，現 PE 上ではアクセスしていないページも存在するが，そのようなページはたいていアクセス頻度の高い木の上位ページであり，これも主記憶上にキャッシュされている確率が高い．

図 3 のスプリットにおいて，経路誤りが発生する例を示す．スプリット前の図 3（a）において，検索キーが “Key_i” であるプロセス a が存在したとする．プロセス a は，(I1）のラッチ獲得後，(I1）から次のページが（I2）であるという情報を得る．プロセス a はラッチカップリングを用いていないため，(I2）のラッチを獲得する前に（I1）のラッチを解放することになる．そのため，(I2）のラッチを獲得する前に，他のプロセスによって図 3（b）のように更新される可能性がある．しかし，プロセス a は（I2）のラッチを獲得し，そのページを検索することになるが，検索キー “Key_i” を含むキー範囲は，更新によって（I3）に移っている．すなわち，プロセス a は正しい経路である（I3）ではなく，誤った経路である（I2）を辿っていることになる．

このような場合，提案手法では，プロセス a はルートページ

(注2)：経路誤りの確率は非常に低いため，正しい経路に復帰するための手続きによって性能に差はでないと考えられる．節 4.5 においてその値を示す．

からリスタートを行う．しかし，図 3 のような通常の Btree のインデックスページのデータ構造では，経路誤りを検出できない．プロセス a の検索キー “Key_i” は（I2）のキー範囲外であるが，そのことを（I2）だけでは判別できないため，次のような問題が発生する．検索キーが Key_i-1 より大きいときに経路誤り有と判断し，ルートページからのリスタートや上位ページに後退という手法をとると，検索キーが Key_i-1 と Key_i の間であるときのように正しい経路であった場合，そのページから先に進めなくなる．一方，経路誤り無と判断し，その経路を辿り続けると，検索キーが Key_i のときのように誤った経路であった場合，誤ったアクセスパスを辿ってしまう．

そこで，経路誤りを検出するために図 4 の構造のインデックスページを用いる．このデータ構造には，通常のデータ構造には存在しないページ両端の境界値が設けられている．このような境界値をもつ構造を用いることで，境界値を基にそのページだけで正確なキーレンジが判別できる．先ほどの例でも，検索キー “Key_i” が境界値 Key_i 以上であることを基に経路誤りであることを判別でき，上記のような問題は発生しない．

3.2 検　　索

提案手法の検索処理時のプロトコルは MARK-OPT と異なり，ラッチカップリングを用いない．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） 経路誤り検出時，親ページのラッチを解放し，ルートページから再処理
（2） 親ページのキーを比較して，子ページのポインタを取得
（3） 親ページのラッチを解放し，子ページの IS ラッチを獲得

目的の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．

3.3 更　　新

提案手法の更新処理時のプロトコルは，MARK-OPT と同様に二つのフェーズで行われるが，楽観的処理中にラッチカップリングは用いない．また，経路誤り検出によってリスタートした場合，その降下でのマークは無効になる．

更新処理時のプロトコルを厳密に定義する．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，提案手法のプロトコルは図 5 で定義される．

3.3.1 削　　除

タプルの削除に伴いページのエントリーが存在しなくなった(注3)ページは削除される．正しいポインタの参照を保証するラッチカップリングを行っている場合には，削除されるページへのポインタが獲得されることはないので，このことにより特に問題は発生しない．しかし，提案手法はラッチカップリングを行っていないため，削除したページへのアクセスが発生する

(注3)：実用的なワークロードでは，エントリーが fanout の半分未満になったページをマージするより，エントリーが存在しなくなったページをマージ（削除）する方が効率が良い [21].

```
                1  l := H;
↑               2  Parent := null; Child := ROOT; h := 1; m := 1;
                3  while h < l do begin
                4      IX latch on Child;
                5      if Access path error is detected then
                6          Unlatch Child; goto 2;
                7      if Child is not full then
                8          m := h;              /* Marking */
                9      Determine NewChild;
               10      Parent := Child; Child := NewChild; h := h + 1;
               11      Unlatch Parent;
               12  end;
               13  X latch on Child and its copies;
               14  if Access path error is detected then
               15      Release all granted latches; goto 2;
               16  if Child is full then
               17      Release all granted latches; l := m;
               18  else begin
               19      Update;
↓              20      Release all granted latches; operation completion;
               21  end;
↑              22  Parent := null; Child := ROOT; h := 1; m := 1;
               23  while h < l do begin
               24      IX latch on Child;
               25      if Access path error is detected then
               26          Unlatch Child; goto 22;
               27      if Child is not full then
               28          m := h;              /* Marking */
               29      Determine NewChild;
               30      Parent := Child; Child := NewChild; h := h + 1;
               31      Unlatch Parent;
               32  end;
               33  X latch on Child and its copies;
               34  if Access path error is detected then
               35      Release all granted latches; goto 22;
               36  Determine NewChild;
               37  Parent := Child; Child := NewChild; h := h + 1;
               38  while h ≦ H do begin
               39      X latch on Child and its copies;
               40      Determine NewChild;
               41      Parent := Child; Child := NewChild; h := h + 1;
               42  end;
               43  if X latch are not sufficient for SMOs then
               44      Release all granted latches; l := m; goto 22;
               45  else begin
               46      Update including the SMOs;
↓              47      Release all granted latches; operation completion;
               48  end;
```

The first phase / The second phase

図 5　提案プロトコル

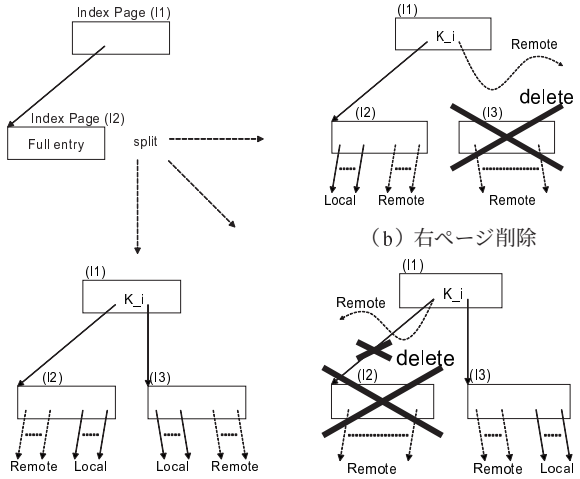
可能性があり，この問題への対処が必要となってくる．

この問題を解決するために，ドレイン手法 [22] と delete_bit という特別なビットを組み合わせた手法を用いる．通常時，delete_bit は 0 にセットされていて，削除されるべきページのみ delete_bit を 1 にセットすることで，他のプロセスに経路誤りを知らせる．delete_bit が 1 にセットされたページはすぐには削除されず，ドレイン手法によって削除される．ドレイン手法では，削除されるべきノードへのポインタが存在したときに，すでに木の降下を行っていたすべてのプロセスが終了した後，そのページの削除を行う．

3.3.2 挿　　入

タプルの挿入に伴うページのエントリー超過によりインデックスページでスプリットが発生した場合，単一 Btree と異なり，Fat-Btree では以下に述べるようなページの削除が起こる可能性がある．Fat-Btree では，ページのスプリットによって，エントリーがリモートページへのポインタのみになることがある．リモートページへのポインタのみを保持するページは，自 PE の葉ページへのアクセスパスを含まないことを意味するので，Fat-Btree の規則により削除される．

スプリット発生時のページ削除の有無から図 6 のような三つのパターンに分けられる．パターン（a）はページの削除が起こらないため，問題は発生しない．パターン（b）はページの削除が起こるが，そのページへのポインタは存在しないため，問題は発生しない．パターン（c）はページの削除が起こり，そ



（a）ページ削除無　　（b）右ページ削除　　（c）左ページ削除

図 6　Fat-Btree におけるインデックスページのスプリット

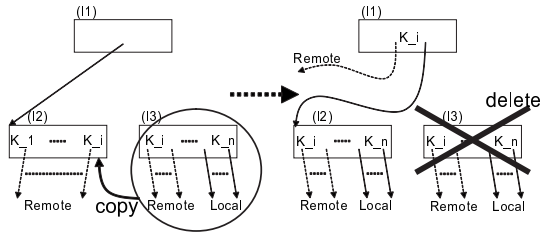


図 7　スプリット時の削除ページの変更

のページへのポインタも存在するため，節 3.3.1 と同様の問題が発生する．

パターン（c）のスプリット時の問題を解決するために，提案手法では次のような手続きを用いる．図 7 のように右ページのエントリーを左ページにコピーし，左ページは削除せず，右ページを削除するようにする．右ページへのポインタは存在しないので，スプリット発生時の削除ページへのアクセスという問題は解決される．

スプリット時のページ削除の問題には，節 3.3.1 で述べる手法によって解決することも可能である．しかし，節 3.3.1 の手法では，ページにアクセスしたプロセスが必ずリスタートが発生するのに対し，本節の手法では，ページのレンジから外れたときのみリスタートが発生するので（確率は二分の一），こちらのほうが効率的である．

3.4 正 当 性

Btree の並行性制御は，目的の葉ページまでの経路を保証しなければならない．その経路でラッチカップリングを行わない場合，経路誤りや削除されるページへのアクセスが発生し，正しい経路が保証されない．よって，ラッチカップリングを用いない手法では，誤った経路を検出し，正しい経路へと復帰する手段が必要となる．

提案手法では，ページの両端に境界値を設け，そのページのキーレンジを明確にすることで，経路誤りの検出を可能にしている．さらに，経路誤りの検出後には，正しい経路に復帰することが必要である．提案手法では，検出後にルートページから再降下を行う．ルートページは必ず正しい経路であるため，正しい経路に復帰することが可能である．よって，提案手法は経路誤りの問題を解決可能である．

提案手法では，削除されるべきページをすぐには削除せず，delete_bit を 1 にセットすることによって，削除されるべきページの検出を可能にしている．さらに，検出後は経路誤りの問題と同様に，ルートページから再降下を行うことで，正しい経路に復帰することが可能である．よって，提案手法は削除されるページへのアクセス発生問題を解決可能である．

また，ページスプリット時に削除ページを交換する手続きは，削除されるページへのアクセス発生問題の経路誤り問題への置換である．よって，経路誤りの問題と同様に解決可能である．

以上より，誤った経路を検出することができ，正しい経路へと復帰する手段が存在するため，提案手法は目的の葉ページまでの経路を保証している．

表 2　実験システムの構成

No. of Nodes:	4–64 (Storages), 16 (Clients)
CPU:	AMD Athlon XP-M 1800+ (1.53GHz)
Memory:	PC2100 DDR SDRAM 1GB
Network	1000BASE-T
Hard Drive:	TOSHIBA MK3019GAX (30GB, 5400rpm, 2.5inch)
OS:	Linux 2.4.20
Java VM:	Sun J2SE SDK 1.5.0_03 Server VM

表 3　実験システムのパラメータ

Page size:	4KB
Tuple size:	350B
Max No. of entries in an index node (fanout):	64
Max No. of tuples in a leaf node:	8

4. 実　　験

提案する新たな並行性制御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 [13]〜[15] 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 2 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．

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．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 の構成パラメータを表 3 に示す．

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，自律ディスク 1 台あたり 5120 タプルの初期 Fat-Btree を構築した後，

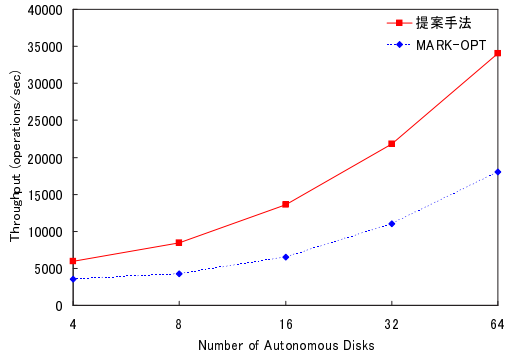


図 8　自律ディスクの台数を変化させたときの並行性制御方式の比較

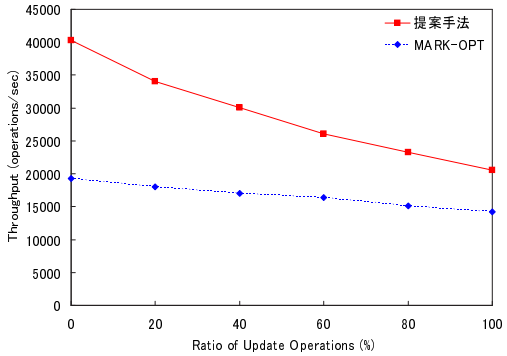


図 10　更新要求の割合を変化させたときの並行性制御方式の比較

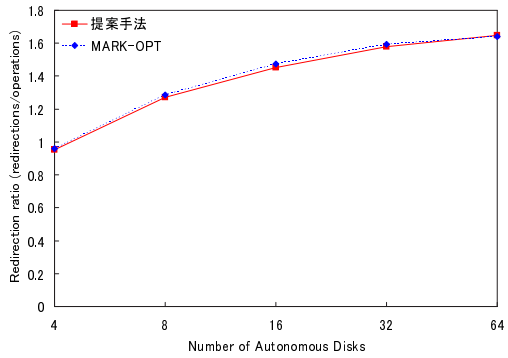


図 9　Fat-Btree におけるリクエスト委譲頻度による比較

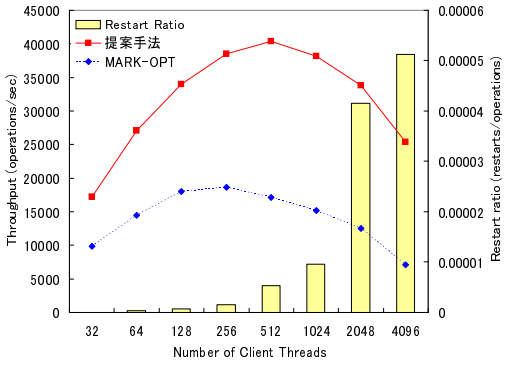


図 11　クライアントのスレッド数を変化させたときの比較

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．

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．また，Fat-Btree におけるリクエスト委譲の回数も測定する．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．最後に，クライアントのスレッド数を変化させたときのスループットおよび経路誤りによるリスタート回数の測定を行う．

4.3　自律ディスクの台数を変化させたときの比較

クライアント PC（1 台あたり並行して 8 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 8 は，更新比率を 20%とし，横軸として自律ディスクの台数を 4 から 64 に変化させ，提案手法と MARK-OPT を，スループットを縦軸として比較したグラフで，図 9 は，PE 間のリクエスト委譲回数を縦軸として比較したグラフである．

MARK-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，図 9 からもわかるように，台数増加に従い Fat-Btree におけるリクエスト委譲頻度が増加したからである．ラッチカップリングを行っている MARK-OPT は，リクエスト委譲に 3 回のネットワーク通信が必要であるため，リクエスト委譲頻度が増加はネットワーク通信オーバーヘッドを増大させる．それに対し，提案手法はリクエスト委譲を 1 回のネットワーク通信で行えるため，台数増加に伴うスループットの上昇が大きい．また，今回の提案によって，委譲の頻度は変化しない．

4.4　更新要求の割合を変化させたときの比較

クライアント PC（1 台あたり並行して 8 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 10 は，自律ディスクの台数を 64 とし，横軸として更新操作の割合を 0%から 100%に変化させ，提案手法と MARK-OPT を，スループットを縦軸として比較したグラフである．

更新要求の割合が小さいときには，提案手法は MARK-OPT のスループットを大きく改善しているが，更新要求の割合が増えるに従い，MARK-OPT と提案手法の差は縮まる．これは，全体の処理時間の多くを占める更新処理に対して，提案手法も MARK-OPT と同じ処理を行うためである．しかし，更新処理の割合が 100%のときでも，提案手法は MARK-OPT の約 1.45 倍のスループットが得られた．

4.5　クライアントのスレッド数を変化させたときの比較

クライアント PC 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 11 は，自律ディスクの台数を 64，更新比率を 20%とし，横軸としてクライアントのスレッド数を 32 から 4096 に変化させ，提案手法と MARK-OPT のスループット，及びそのときの経路誤りによるリスタートの回数を全リクエスト数で割った値を縦軸としたものである．

MARK-OPT は，クライアントのスレッド数の増加に伴うスループット増加は小さく，また，スレッド数 512 からスループットが低下し始める．これは，Fat-Btree におけるリクエスト委譲時のネットワーク通信回数が多いことで，アクセス高負荷時の PC 対する負荷が大きいためである．それに対し，提案手法は

クライアントのスレッド数の増加に伴うスループット増加は大きく，また，スレッド数 512 までスループットが増加し続ける．これは，MARK-OPT よりネットワーク通信回数が少ないことで，PC 対する負荷が小さいためである．

また，クライアントのスレッド数が増えるに従い，提案手法のリスタート率は高くなる．しかし，スレッド数が 4096 のときでも，システム性能に大きな影響がないほど，その値は十分に低い．

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 Btree の並行性制御として新手法を提案した．これは，Fat-Btree におけるリクエスト委譲時の通信コストを減らすために，楽観的処理中にラッチカップリングを行わないように MARK-OPT を拡張した並行性制御方式である．無共有並列計算機を用いた分散環境では，ラッチカップリングはコストが大きい．これを除去することによって，提案手法は高いシステムスループットを獲得できる．提案手法では，ラッチカップリングを行わないため，経路誤りという問題が発生するが，ページの両端に境界値を設けることで，問題を解決している．また，削除ページへのアクセスという問題も発生するが，ドレイン手法と delete_bit を組み合わせることで，この問題を解決している．スプリット時に発生するページの削除には，そのページを指すポインタがまだない非削除ページのエントリーをコピーし，指すポインタがないページの方を削除することで，より効率的に処理を行っている．さらに自律ディスク上での実験により，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た．

今後の課題として，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 [19], [20] 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．

さらに，MARK-OPT や提案手法と同様に INC-OPT 方式の拡張である PO-P 方式 [23] との比較を検討している．PO-P 方式は，INC-OPT と PO [24] というトップダウン手法を組み合わせた並列 Btree に適した楽観的な並行性制御方式である．

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）及び東京工業大学 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] George Copeland, William Alexander, Ellen Boughter, and Tom Keller. Data Placement in Bubba. In *Proc. of ACM SIGMOD'88*, pp. 99–108, 1988.

[2] Shahram Ghandeharizadeh and David J. DeWitt. Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. In *Proc. of VLDB'90*, pp. 481–492, 1990.

[3] David J. DeWitt and Jim Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, Vol. 35, No. 6, pp. 85–98, 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 AI97-44, DE97-77, 1997.

[5] Haruo Yokota, Yasuhiko Kanemasa, and Jun Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *Proc. of ICDE'99*, pp. 448–457, 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. 情処研報 DBS-119-68, 1999.

[7] 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. DEWS2001，7A-7, 2001.

[8] Jun Miyazaki, Yohei Abe, and Haruo Yokota. Availabilities and Costs of Reliable Fat-Btrees. In *Proc. of PRDC2004*, pp. 163–172, 2004.

[9] Jun Miyazaki and Haruo Yokota. Concurrency Control and Performance Evaluation of Parallel B-tree Structures. *IEICE Trans. Inf. Syst.*, Vol. E85-D, No. 8, pp. 1269–1283, 2002.

[10] Rudolf Bayer and Mario Schkolnick. Concurrency of Operations on B-trees. *Acta Inf.*, Vol. 9, No. 1, pp. 1–21, 1977.

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

[12] 吉原朋宏, 小林大, 田口亮, 上原年博, 横田治夫. 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. DEWS2005, 2A-o4, 2005.

[13] Haruo Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of DANTE'99*, pp. 435–442, 1999.

[14] Daisuke Ito and Haruo Yokota. Automatic Reconfiguration of an Autonomous Disk Cluster. In *Proc. of PRDC2001*, pp. 169–172, 2001.

[15] Haruo Yokota and Ryota Abe. Secondary Storage Configuration for Advanced Data Engineering. In *Nontraditional Database Syst.*, chapter 14, pp. 212–230. Taylor & Francis, 2002.

[16] 吉原朋宏, 渡邊明嗣, 小林大, 田口亮, 上原年博, 横田治夫. 並列 Btree 構造における負荷分散処理の並行性制御への影響. 情処研報 DBS-137-32, 2005.

[17] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1992.

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

[19] Philip L. Lehman and S. Bing Yao. Efficient Locking for Concurrent Operations on B-trees. *ACM Trans. Database Syst.*, Vol. 6, No. 4, pp. 650–670, 1981.

[20] Vladimir Lanin and Dennis Shasha. A Symmetric Concurrent B-tree Algorithm. In *Proc. of FJCC'86*, pp. 380–389, 1986.

[21] Theodore Johnson and Dennis Shasha. Utilization of B-trees with Inserts, Deletes and Modifies. In *Proc. of PODS'89*, pp. 235–246, 1989.

[22] H. T. Kung and Philip L. Lehman. Concurrent Manipulation of Binary Search Trees. *ACM Trans. Database Syst.*, Vol. 5, No. 3, pp. 354–382, 1980.

[23] Damdinsuren Amarmend, Masayoshi Aritsugi, and Yoshinari Kanamori. PO-P: A Concurrency Control Protocol for Parallel B-trees. *IPSJ Trans. Databases*, Vol. 45, No. SIG14, pp. 30–38, 2004.

[24] Y. Mond and Yoav Raz. Concurrency Control in B+-Trees Databases Using Preparatory Operations. In *Proc. of VLDB'85*, pp. 331–334, 1985.