並列B-tree構造における負荷分散処理時の並行性制御の評価

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
†† 日本学術振興会特別研究員 DC
††† 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　並列データベースや分散ストレージシステムのインデックス構造に並列 B-tree 構造を用いることは有効であり，我々はこれまで並列 B-tree 構造に適した並行性制御手法を提案してきた．また，データマイグレーションは並列データベースや分散ストレージシステムにおける負荷分散を行うための有効な技術であるが，並列 B-tree 構造を用いている場合，データマイグレーション時の並行性制御も重要となる．本稿では，我々が提案してきた並行性制御手法をデータマイグレーション時の並行性制御に適用する．並列 B-tree 構造である Fat-Btree を採用している自律ディスクにおいて，データマイグレーションが発生する環境で実験を行い，それらの手法が常にスループットを改善し，データマイグレーション時の並行性制御として有効であることを示す．

キーワード　並行性制御，アクセスパス，並列 · 分散 DB，並列 B-tree，データマイグレーション

Evaluation of Concurrecy Control for Load-Balancing Operations on a Parallel B-tree 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
†† Research Fellow (DC), Japan Society for the Promotion of Science
††† 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　We have proposed concurrency control methods for parallel B-tree structures, which are used in parallel databases and distributed storage systems. On the other hand, data migration is efficient to handle the access skew in these systems. Migrating data managed by parallel B-tree structures requires an efficient concurrency control for data migration. In this paper, we apply our methods to data migration. To compare the performance of our methods and MARK-OPT, we implemented them on an autonomous-disk system adopting the Fat-Btree, a parallel B-tree structure. The experiments on environments with data migration indicate that our methods always improves the system throughput, and are also effective as concurrency controls for data migration.

Key words　Concurrency Control, Access Path, Parallel · Distributed DB, Parallel B-tree, Data Migration

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，そのた

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]．ディレクトリ構成として Fat-Btree を用いることで，他の並列 B-tree を用いるより高速なアクセスが可能であることが実験により明らかにされている．また，アクセス集中による負荷の偏りが発生した場合には，その負荷偏りを除去する必要がある．データマイグレーションは並列データベースや分散ストレージシステムにおける負荷分散を行うための有効な技術であるが，Fat-Btree では葉ページ単位の移動により，効率的なデータマイグレーションを行うことができる．

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 [5] や MARK-OPT 方式 [6] が提案されている．しかし，INC-OPT や 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 にリクエストを移譲するとき，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り，リクエスト移譲コストが大きい．

[7] において，この問題を解決する新たな並行性制御方式が提案されている．本稿ではこの方式を LCFB 方式を呼ぶ．LCFB 方式はラッチカップリングを行わないことで，リクエスト移譲時の通信コストを減らし，ラッチカップリングを行っている MARK-OPT より優れた性能を得ることが可能である．ラッチカップリング省略に伴い発生する問題を解決するために，インデックスページへの境界値の設置，ページ削除の遅延などを行っている．

B-link アルゴリズムと LCFB を組み合わせた並行性制御手法が提案されている [8]．本稿ではこの方式を LCFB-link 方式と呼ぶ．B-link は単一 B-tree 上で優れた並行性制御が実現できることが知られている．B-link は，サイドポインタにより隣のインデックスノードへの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，X ラッチの獲得回数および範囲を小さくすることが可能である．LCFB におけるラッチカップリングの省略により複数 PE 間に跨る木降下コストを減らし，単一 PE において B-link を用いることにより X ラッチの獲得回数を減らすことができる．

本稿では，LCFB 及び LCFB-link をデータマイグレーション時の並行性制御に適用する．また，自律ディスク上にデータマイグレーション時の並行性制御として，MARK-OPT，LCFB および LCFB-link を実装し，データマイグレーションを発生させた環境での実験から，LCFB 及び LCFB-link のデータマイグレーション時の並行性制御としての効果を示す．

以下に本稿の構成を述べる．まず 2 節で Fat-Btree とデータマイグレーションについて述べる．次に 3 節で従来の並行性制御方式と我々の提案する並行性制御について述べる．4 節では，それらのデータマイグレーション時の並行性制御への適用について述べる．5 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 6 節でまとめと今後の課題を述べる．

2. Fat-Btree とデータマイグレーション

2.1 Fat-Btree 構造

Fat-Btree [4] は，B^+-tree 全体をページ単位で PE 間に分配する並列 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 B-tree 構造と比較して高速に行うことができる．

2.2 データマイグレーション

データマイグレーションは，並列データベースやストレージシステムにおいてアクセス偏り除去のための有効な手段である．分散配置された PE に対して，各データのアクセス負荷を考慮してデータ配置を決定し，データ移動によりデータ再配置を行うことで，システムの性能を大きく低下させるアクセス集中を回避することができる．

2.3 Fat-Btree のデータマイグレーション

値域分配方式では，ある PE（移動元）から論理的に隣接する左右の PE（移動先）にのみデータマイグレーションを行うことができ，インデックスに Fat-Btree を用いる場合，移動元 PE でインデックスが端である葉ページをそのまま，移動先 PE に移すことで，データマイグレーションを行うことができる (図 2)．この場合，実際にデータマイグレーションに関わる PE は，隣接する 2PE だけで済むので，他の PE の処理を妨害することなくデータマイグレーションを実現できる．

以下，ノードを移動するアルゴリズム [4] の概要を述べる．この葉ページの移動の際，移動先の PE がその葉ページの上位のインデックスページを持たない場合，移動対象となる葉ページ

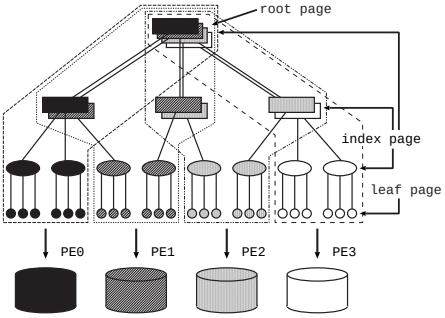

図 1　Fat-Btree 構造

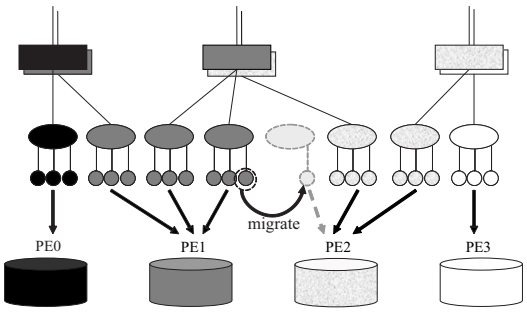

図 2　Fat-Btree におけるデータ移動の方法

表 1　ラッチマトリクス

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

からルートページまでのアクセスパス上のインデックスページを，移動先の PE が持っていないページ分は移動元の PE からコピーする．逆に葉ページの移動によって移動元の PE から移動した葉ページの上位のインデックスページからその PE 内の下位のインデックスページのエントリーがなくなる場合，そのインデックスページを削除する．この操作でそれより上位のインデックスページからローカルページのエントリーがなくなった場合，再帰的に上位のインデックスページを削除する．

3.　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 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．

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 [9]．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，コピーを持つすべての PE でラッチを獲得する必要がある．

3.1　従来の並列 B-tree 並行性制御方式

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 [5] が提案されている．しかし，INC-OPT は，SMO（Structure Modification Operation）発生時にルートページからの木の再降下である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，リスタートが多数必要になり，システム処理性能の低下をもたらす．

この問題を解決する並行性制御方式として，MARK-OPT 方式 [6] が提案されている．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．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 INC-OPT より高いスループットを獲得できることが明らかにされている．

3.2　データマイグレーション時の並行性制御

データマイグレーションでは他の更新処理と同様に SMO 発生範囲に X ラッチを獲得することが必要である．

挿入に関してはノードのエントリー制限を超えた場合に SMO が発生し，削除に関してはノードのエントリーが存在しなくなった場合に SMO が発生する．それに対し，データマイグレーションは葉ページを移動するため，葉ページとその親のインデックスページでは必ず SMO が発生する．それより上位のページで SMO が発生するのは，データマイグレーション先の PE でローカルエントリーが発生して，必要になったインデックスページが作成される場合，または，データマイグレーション元の PE のインデックスページでローカルエントリーがなくなり，不要になったインデックスページが削除される場合である．

このように，挿入及び削除に関しては，X ラッチ獲得範囲を判断する条件が 1 つのみであるのに対し，データマイグレーションに関しては，2 つの条件が存在する．それぞれの条件は独立であるため，MARK-OPT をデータマイグレーション時の並行性制御に適用する場合，マーキングも 2 種類用いる必要がある．2 種類のマーキングを更新しながら木を降下し，リスタート時には SMO 発生範囲が広い条件に合わせるために小さいほうの値を用いる．

3.3　LCFB 方式

LCFB 方式 [7] は，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である．

MARK-OPT では，目的の葉ページまでの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 間でリクエスト移譲が発生するとき，一回のリクエスト移譲ごとに，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，リクエスト移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．

LCFB のように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，LCFB はルートページからリスタートする．また，経路誤りを検出するため

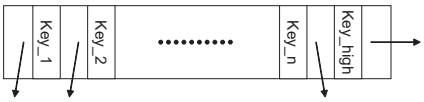

図 3　B^{link}-tree のノード

に，ページ両端の境界値が設けられているインデックスページを用いる．このような境界値をもつ構造を用いることで，経路誤りの有無を判別できる．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 [10] 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，ローカルエントリーが無くなったページにコピーし，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．

3.4　B^{link}-tree

B^{link}-tree [11], [12] では，各レベルの全ノードがリンクしている．各ノードは，同レベルの次ノードを指すこのサイドポインタとポインタに割り当てらたキー（high key）を保持している．B^{link}-tree におけるポインタは，ページスプリットを 2 フェーズ（分割と適切な親へのインデックスエントリーの挿入）で処理することを可能にする．図 3 は B^{link}-tree のノードを示している．“high key” より大きい検索キーを持ったプロセスは，右リンクを用いて適切なページを獲得する．このような木を横へ辿る操作はリンクチェイスと呼ばれる．また，B-link 方式では，検索，更新プロセスともに，葉ノードまでの過程でラッチカップリングを用いない．また，SMO を実行するとき，各ページで X ラッチを獲得するが，ラッチカップリングは用いない．B-link アルゴリズムでは，同時に 1 つのラッチしか獲得されない．

3.5　LCFB-link 方式

LCFB-link 方式 [8] は，単一 B-tree 上で優れた並行性制御を実現できる B-link を Fat-Btree に適用し，Fat-Btree 向けの並行性制御方式である．

LCFB では，ラッチカップリングを省略することにより木の降下コストを削減し，高速な検索を可能にした．しかし，更新処理では SMO に関連する全ノードに X ラッチを獲得する必要がある．そのため，更新処理時のコストは依然大きい．X ラッチの獲得回数や範囲が小さく，更新コストが小さい手法として，B-link アルゴリズムが知られているが，並列 B-tree 全体で B-link アルゴリズムを用いることは難しい．LCFB-link では，LCFB と組み合わせることにより，更新コストの小さい B-link を並列 B-tree で利用することを可能にする．これにより，並列 B-tree において，ラッチカップリング省略による効率的な検索だけでなく，ラッチ獲得数及び範囲を小さくすることによる効率的な更新が可能になる．

LCFB-link では，同じ PE 内のページ間はサイドポインタでリンクさせ，異なる PE のページ間はサイドポインタを用いない（図 4）．スプリット時に削除されない同じ PE 内の (1, 2) から (1, 3) のみをリンクさせ，(1, 2) から異なる PE の (2, 3)，(3, 3) をリンクさせないことで，(1, 3)，(2, 3)，(3, 3) のスプリット時には (3, 3) が削除されるが，サイドポインタの削除は発生しない．これにより，ルートページから葉ページまでの経路内のページのみ更新により，サイドポインタの一貫性保持が可能になる．

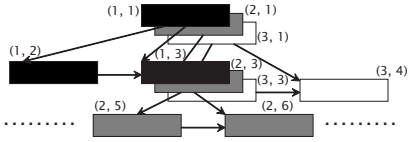

図 4　Fat-B^{link}-tree

図 4 の構造を用いた場合，コピーページが存在するインデックスノードでは，サイドポインタのないページが存在する．したがって，コピーページが存在するノードでは，B-link アルゴリズムを用いることはできない．そこで，LCFB-link は，コピーページが存在するノードの処理に他の手法を用いる．この手法として，全 SMO に必要な X ラッチを獲得した後に更新を行う LCFB を選択することで，コピーページが存在するインデックスノードでも更新処理を行うことが可能である．

4.　LCFB 方式のデータマイグレーション時の処理

4.1　LCFB 方式

LCFB のデータマイグレーション時の並行性制御への適用は容易である．不要になったページの削除以外は，LCFB の更新処理とデータマイグレーション時の MARK-OPT の処理を組み合わせるだけである．

LCFB のデータマイグレーション時のプロトコルは，ラッチの獲得及び解放に関しては他の更新処理と同様である．ただし，マーキングに関しては MARK-OPT 同様に 2 種類のマーキングが必要となる．また，LCFB のデータマイグレーションにおいて，必要なページが作成されることに関しては，特に問題は発生しない．ラッチカップリングを行っていないために作成されたばかりのそのページにアクセスできなくても，コピーページが存在すれば，そのページを経由して目的の葉ページに辿り着くことができるためである．このとき，そのページにアクセスできなかったために，リクエスト移譲の回数が増える可能性があるが，非常に稀であるため，大きな影響はないと考える．それに対し，不必要なページの削除に関しては，LCFB で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可能性がある．そのため，LCFB のページ削除時に用いられるドレイン手法と delete_bit を組み合わせた手法を用いる．これにより，データマイグレーションを行ったときにも，目的の葉ページへの正しい経路を保証する．

4.2　LCFB-link 方式

LCFB-link 方式を用いてデータマイグレーションを行おうとすると，同じ PE 内のページのみをサイドポインタでリンクしている構造では，B-link の特徴であるサイドポインタの一貫性を保持するためのコストが非常に高い．LCFB-link 方式を用いて効率的なデータマイグレーションを行うためには，この問題を解決しなければならない．

4.2.1　マイグレーション時のサイドポインタの更新

B-link を用いている場合のデータマイグレーションにおいて

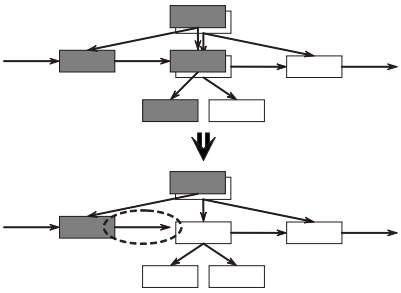

図 5　右 PE へのマイグレーションにおけるインデックスページの削除

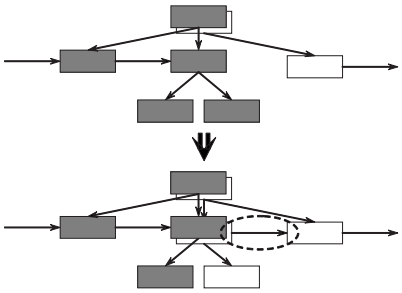

図 6　右 PE へのマイグレーションにおけるインデックスページの作成

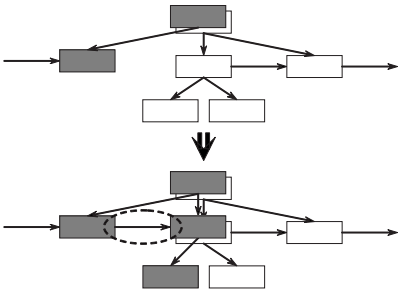

図 7　左 PE へのマイグレーションにおけるインデックスページの作成

は，必要になったページの作成および不必要になったページの削除におけるサイドポインタの更新が問題になる．B-link のサイドポインタは単方向リンクであるため，論理的に右の PE へのマイグレーションか左の PE へのマイグレーションかによって，異なる問題が発生する．

まず，右 PE へのマイグレーションについて述べる．右 PE へのマイグレーションでは，不必要になったページを削除する場合は，図 5 のように，図中の破線で囲われた左ページからのサイドポインタが存在するため，そのポインタを削除しなければならない．しかし，この左のページは，ルートページから葉ページまでの経路上になく，左隣にあるページはサイドポインタによっても辿れないため，このページを更新するためのコストは非常に高い．

この問題を解決するために，各 PE の各レベルの最右ページの右隣にダミーページを設ける．マイグレーションによって最右ページが削除される場合には，このダミーページを削除し，本来削除されるべきページは削除せず，削除されるダミーページの代わりにダミーページに設定する．これにより，経路外のページの更新は発生せず，効率的にサイドポインタの更新を行うことが可能になる．ダミーページであるか否かは，1 であるときにダミーページであることを示すビット（dummy_bit）によって区別される．ダミーページは，dummy_bit が 1 にセットされている以外は，通常のページと同様である．そして，マイグレーションによって，各ページはダミーページになったり通常のページになったりする．

また，右 PE へのマイグレーションにおいて必要になったページを作成するとき，図 6 のように，図中の破線で囲われた作成されたページから右のページへのサイドポインタを作成する必要がある．しかし，この右ページへのポインタ情報は，親ページから得られる場合もあるが，最悪の場合はルートページから再び辿らないと得られないため，ポインタを作成するコストが非常に高い．

この問題を解決するために，論理的に隣り合う PE 間で左 PE の最右ページから右 PE の最左ページへのサイドポイントをリンクさせる．先の問題を解決するためにダミーページを設けているため，最右ページであるダミーページからのサイドポインタとなる．このポインタにより，上位ページからの再降下を行うことなく更新を行うことができる．また，PE 間をポインタでリンクさせたことにより，ポインタの変更が問題になる．この変更は，左 PE へのマイグレーション時に，ポインタでリンクしていた移動元の PE でインデックスページが削除される場合にのみ発生する．このとき，削除されるページの右ページへのポインタに変更する必要がある．また，そのような場合には，変更されるポインタのあるダミーページの両隣のインデックスページに X ラッチが獲得されるので，ダミーページの更新も容易である．

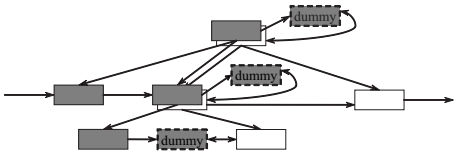

図 8　ダミーページを備えた Fat-B^{link}-tree

続いて，左 PE へのマイグレーションについて述べる．左の PE へのマイグレーションでは，必要なページを作成するとき，図 7 のように，図中の破線で囲われた作成されたページに左のページからサイドポインタを作成する必要がある．しかし，この左のページは，ルートページから葉ページまでの経路上になく，左隣にあるページはサイドポインタによっても辿れないため，このページを更新するためのコストは非常に高い．

この問題を解決するために，論理的に隣り合う PE 間で左 PE の最右ページから右 PE の最左ページへのサイドポイントをリンクさせる．一つ目の問題を解決するためにダミーページを設けているため，最右ページであるダミーページへのサイドポインタとなる．このポインタを辿ることにより更新を行えるため，他の経路を辿る必要がなくなる．

上記の解決方法をすべて適用した場合の B-link を備えた Fat-Btree の構造は図 8 のようになる．破線枠のページがダミーページである．この図からもわかるように PE 間に双方向リンクが存在する．このリンクはマイグレーションのためだけに使用されるのだが，上記の解決方法では，双方向リンクの両側のページにそれぞれの方向から X ラッチを獲得する．そのため，デッドロックが発生する可能性がある．デッドロックを回避するため，左右どちらかの PE にラッチ獲得順序の権利を表すラッチを設ける．これにより，マイグレーションの効率が悪くなることが考えられるが，ある PE 間のマイグレーションが左右双方向同時に発生すること自体が問題である．基本的にはそのような命令は発生せず，このラッチによる待ちはないと考える．

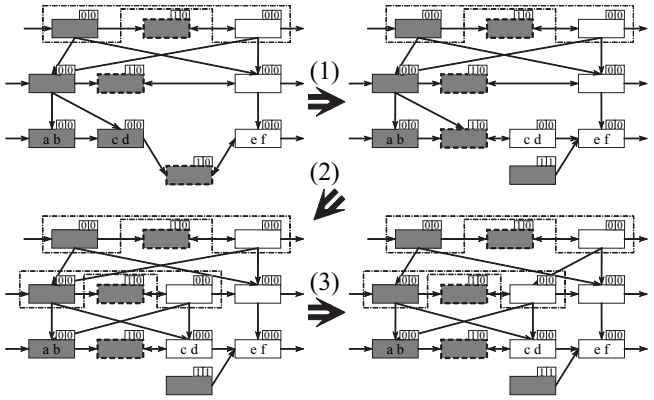

図 9　左の PE から右の PE へのマイグレーション

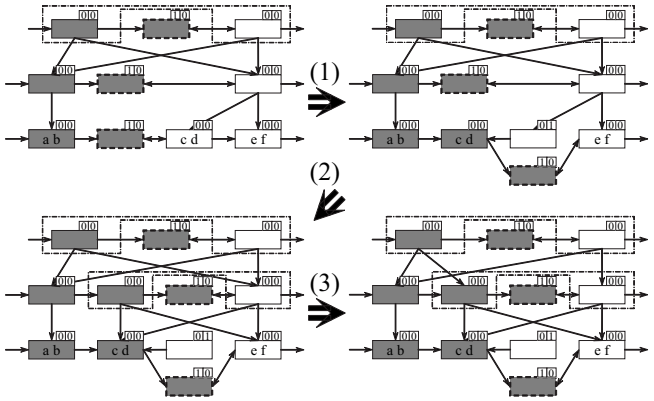

図 10　右の PE から左の PE へのマイグレーション

4.2.2　ダミーページを用いたマイグレーション

上記で説明した解決方法を用いた場合のデータマイグレーションの処理を図 9, 10 の例を用いて示す．図 9, 10 における各ページの右肩の 2 つの数字は，左は dummy_bit，右は delete_bit を示す．また，破線枠のページはダミーページであり，鎖線で囲われたページ同士はコピー関係にある．

図 9 の右 PE へのマイグレーションの例を説明する．右 PE へのマイグレーションでは，移動元 PE の右端の葉ページまで木を降下した後，以下の処理を行う．箇条書き番号は図中の遷移番号に対応する．

（1） 移動元 PE の右端の葉ページとその右隣のダミー葉ページに X ラッチを獲得する．そして，葉ページのコピーを移動先 PE に作成する．作成したページの左右へのポインタを作成するとともに，移動先 PE の左端だった葉ページに X ラッチを獲得し，左方向へのポインタを削除し，このページのラッチを解放する．その後，移動元の PE において，ダミー葉ページの delete_bit を 1 にセットし，もとの葉ページの dummy_bit を 1 にセットし，これらのページのラッチを解放する．

（2） 親ページとその右隣のダミーインデックスページに X ラッチを獲得した後，新たに作成した葉ページへのポインタを格納するインデックスページが移動先 PE に存在しないため，このインデックスページのコピーを移動先 PE に作成する．そして，葉ページのときと同様，左下の図になるように適切にサイドポインタを更新する．

（3） その親ページにはコピーページが存在するため，リスタートし，ルートページから再降下する．親ページとそのコピーに X ラッチを獲得した後，それらのページを更新し，ラッチを解放する．

表 2　実験システムの構成

#Nodes:	128 (Storages), 16 (Clients)
CPU:	AMD Athlon XP-M 1800+ (1.53 GHz)
Memory:	PC2100 DDR SDRAM 1 GB
Network	1000BASE-T
HDD:	TOSHIBA MK3019GAX (30 GB, 5400 rpm, 2.5 inch)
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:	4 KB
Tuple size:	350 B
Max No. of entries in an index node (fanout):	64
Max No. of tuples in a leaf node:	8

図 10 の左 PE へのマイグレーションの例を説明する．左 PE へのマイグレーションでは，移動元 PE の左端の葉ページまで木を降下した後，以下の処理を行う．箇条書き番号は図中の遷移番号に対応する．

（1） 移動先 PE のダミー葉ページと移動元 PE の左端の葉ページに X ラッチを獲得する．そして，移動元 PE の葉ページのエントリーを移動先 PE のダミーページにコピーし，移動先 PE に新しいダミーページを作成し，元のダミーページのダミーページの dummy_bit を 0 にセットする．そして，右上の図になるように適切にサイドポインタを更新する．

（2） 親ページとそのリンク先のダミーページに X ラッチを獲得した後，新たに作成した葉ページへのポインタを格納するインデックスページが移動先 PE に存在しないため，葉ページのときと同様にダミーページにエントリーをコピーし，新たにダミーページを作成する．そして，左下の図になるようにサイドポインタを更新する．

（3） その親ページにはコピーページが存在するため，リスタートした後，親ページとそのコピーを更新する．

5. 実　　験

提案手法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] に提案手法を実装し実験を行った．自律ディスクは 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 192 ポートのネットワークスイッチを用いて，実験環境を構成した．ストレージノードおよびクライアントは，1 つの同じスイッチに接続したスタートポロジとした．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．

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，各クライアント PC（1 台あたり並行して 64 スレッド）

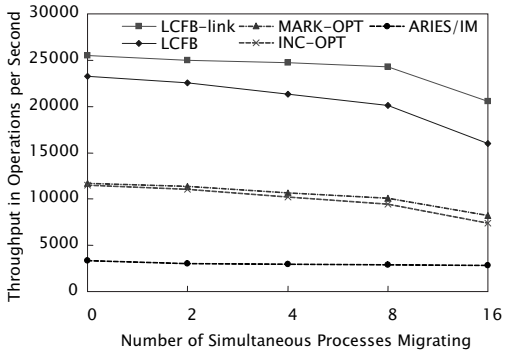

図 11　データマイグレーションの頻度を変化させたときの比較

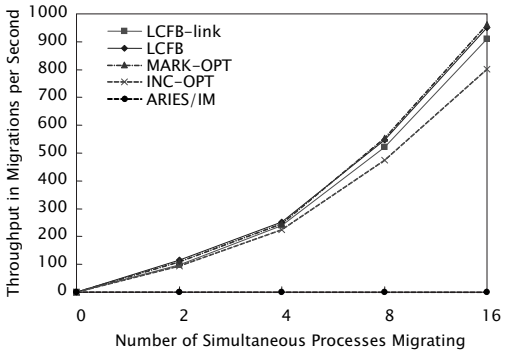

図 13　データマイグレーション処理性能の比較

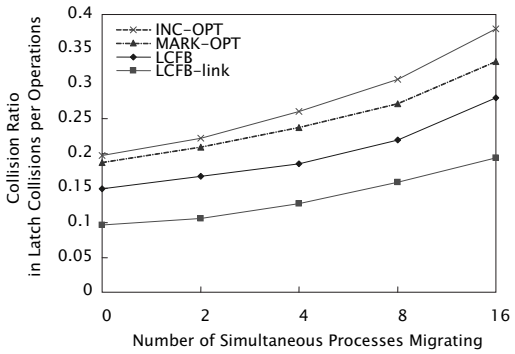

図 12　ラッチ衝突比率の比較

から同時にリクエストを送信した．キーはランダムに選び，ランダムに選択した PE（自律ディスク）に対して，検索と挿入操作（挿入比率 40%）を実行したときのスループットから性能を評価した．また，20 ミリ秒に 1 回ずつ移動元 PE に一方向へのデータマイグレーションのリクエストを送信した．100 回リクエストを送信したら，先ほど移動先であった PE から逆方向へのデータマイグレーションのリクエストを送信した．リクエストを送る PE は，データマイグレーションの並列数に合わせて等間隔に PE を選んだ．

以下に本実験の概要を述べる．まず，データマイグレーションの頻度を変化させたときのスループットの測定し，各手法の比較を行う．また，ラッチ衝突回数の測定も行う．次に，データマイグレーションを一定間隔ではなく，前リクエストのレスポンス後に次のマイグレーションをリクエストすることで，マイグレーションの処理性能の比較を行う．最後に，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する．

5.1　データマイグレーションの頻度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 32，横軸としてデータマイグレーションの同時リクエスト数を 0 から 16 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．また，図 12 は，1 命令あたりのラッチ衝突数を縦軸として比較したグラフである．

ARIES/IM におけるマイグレーションでは常に木ラッチが獲得されるため，常にスループットが低い．また，INC-OPT と比べて，MARK-OPT は常に高いスループットが得られている．これは，図 12 からも分かるように，MARK-OPT が X ラッチ獲得を削減し，ラッチ衝突が減少したためである．その MARK-OPT と比べて，LCFB は常に高いスループットが得られている．マイグレーションではリクエスト移譲は発生しないため，マイグレーションのコストは基本的に同じである．よって，この結果は検索および更新におけるリクエスト移譲コストを削減しているためである．また，図 12 からも分かるようにラッチ衝突率も下がっており，LCFB はトランザクション自身のリクエスト移譲コストを削減するとともに，他のトランザクションの待ち回数も減らしている．

LCFB-link は LCFB と比べてさらに高いスループットを得られている．これは，B-link による挿入処理の効率化が大きく影響している．また，マイグレーション頻度が大きい場合には，低いときに比べて高い改善が見られる．これは図 12 からも分かるように，マイグレーション頻度上昇によるラッチ衝突率の上昇を抑えているためである．LCFB は，葉ページのコピー等の更新時にも SMO の範囲全体に X ラッチを獲得する．それに対し，LCFB-link は 4.2.2 節の例からもわかるように，葉ページの更新時にインデックスページにラッチを獲得しない．よって，LCFB-link ではマイグレーション時のラッチ衝突が減少する．ただし，挿入操作頻度の増加に比べて，データマイグレーション頻度の増加による LCFB に対する LCFB-link のスループットの改善は小さい．これは，データマイグレーションは複数の PE に跨る SMO が必ず発生し，LCFB-link は LCFB を用いた処理が必ず発生するためである．

5.2　データマイグレーション処理性能の比較

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．本節では，20 ミリ秒間隔ではなく，前のリクエストのレスポンスが返ってきたら，次のマイグレーションをリクエストした．図 13 は，自律ディスクの台数を 32，横軸としてデータマイグレーションの同時リクエスト数を 0 から 16 に変化させ，1 秒あたりのデータマイグレーションリクエスト数を縦軸として比較したグラフである．

MARK-OPT と LCFB のマイグレーションの処理性能はほぼ同じである．これは，この 2 手法の処理の大きな違いがラッチカップリングの有無だけで，リクエスト移譲の起きないマイグレーションではその違いは処理時間に影響を及ぼさないためで

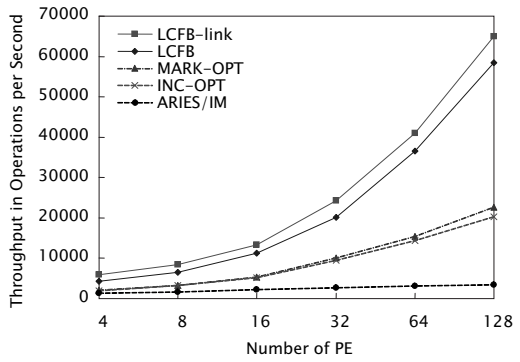

図 14　自律ディスクの台数を変化させたときの比較

ある．それに対し，LCFB-link は，マイグレーション同時リクエスト数が 16 のときに LCFB の約 95%の処理性能である等，他の 2 手法と比べてわずかであるがマイグレーションの処理性能が低い．これは，サイドポインタの一貫性を保持するために，マイグレーションにおいて更新の必要なページ数が増えているためである．ただし，性能低下は無視できる範囲であり，INC-OPT および ARIES/IM よりは高い処理性能が得られている．それに対し，前節の結果の通り，検索，挿入に関しては，LCFB-link が最も高いスループットが得られている．

5.3　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 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．また，データマイグレーションの同時リクエスト数は自律ディスクの台数の 4 分の 1 にした．図 14 は，横軸として自律ディスクの台数を 4 から 128 に変化させ，LCFB-link，LCFB，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．

LCFB や LCFB-link は，ARIES/IM，INC-OPT および MARK-OPT と比べて，台数増加に伴うスループット上昇が大きい．これは，LCFB や LCFB-link はリクエスト移譲のコストが低いからであり，データマイグレーションがある場合においても，その効果が大きいことを示している．したがって，LCFB と LCFB-link はデータマイグレーションの発生する環境においても，ARIES/IM，INC-OPT および MARK-OPT より高い拡張性があることがわかる．

6.　まとめと今後の課題

本稿では，我々の提案してきた並列 B-tree 向け並行性制御手法のデータマイグレーションへの適用方法を示した．delete_bit やダミーページを用いることで，通常の B-tree に対する命令だけでなくデータマイグレーションも処理可能になった．さらに，自律ディスクにおけるデータマイグレーションの発生する環境での実験により，従来手法との比較を行った．データマイグレーションの処理性能に関しては，LCFB と MARK-OPT はほぼ等しく，LCFB-link はやや低いが，無視できる程度であった．それに対し，検索，挿入に関しては，LCFB-link が最も高いシステムスループットが得られた．

本稿では，アクセス偏りのない環境で，クライアントからランダムにリクエストを送信することで，データマイグレーションを発生させたが，本来は負荷値により負荷偏りを検出し，それに基づいたデータマイグレーションを行うべきである．実験に用いた自律ディスクには，それらの操作を自律的に行う機能を備えており，アクセス偏りのある環境で，その機能を用いて実験を行う必要がある．

また，MARK-OPT と同様に INC-OPT 方式の拡張である PO-P 方式 [14] との比較を検討している．PO-P 方式は，INC-OPT と PO [15] というトップダウン手法を組み合わせた並列 B-tree に適した楽観的な並行性制御方式である．

謝　　辞

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 (18049026) 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] George Copeland, William Alexander, Ellen Boughter, and Tom Keller. Data Placement in Bubba. In *Proc. of SIGMOD Conference '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] Haruo Yokota, Yasuhiko Kanemasa, and Jun Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *Proc. of ICDE '99*, pp. 448–457, 1999.

[5] Jun Miyazaki and Haruo Yokota. Concurrency Control and Performance Evaluation of Parallel B-tree Structures. *IEICE Transactions on Information and Systems*, Vol. E85-D, No. 8, pp. 1269–1283, 2002.

[6] 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 論文集，2A-o4, 2005.

[7] 吉原朋宏, 小林大, 田口亮, 横田治夫. 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. DEWS2006 論文集，7C-o3, 2006.

[8] 吉原朋宏, 小林大, 田口亮, 横田治夫. Fat-Btree における B-link を用いた並行性制御手法. 情処研報 DBS-140-90, 情報処理学会, 2006.

[9] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Francisco, California, USA, 1992.

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

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

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

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

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

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