社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－DBS－140（Ⅱ）（90）
2006／7／14

Fat-Btreeにおける B-linkを用いた並行性制御手法

吉原朋宏[†1]　小林大[†1,†2]
田口亮[†3]　横田治夫[†4,†1]

B-linkは単一Btree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．しかし，並列Btree全体へB-linkを適用し，サイドポインタの一貫性保持することは難しい．本稿では，B-linkを用いた並列Btree構造Fat-Btreeにおける新たな並行性制御手法を提案する．B-linkを用いることで，Xラッチの獲得数や同時獲得範囲を小さくすることが可能である．Fat-Btreeを採用している自律ディスクに提案手法を実装し，従来手法と比較を行う．更新要求の割合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，高更新環境において特に有効であることを示す．

A Concurrency Control Method Using the B-link on the Fat-Btree

Tomohiro Yoshihara,[†1] Dai Kobayashi,[†1,†2] Ryo Taguchi[†3] and Haruo Yokota[†4,†1]

The B-link can achieve excellent concurrency control, which uses links to chain all nodes at each level together. In the B-link algorithm, neither readers nor updaters latch-couple on their way down to a leaf node and they acquire the latch only on one node at a time. However, it is difficult to guarantee the consistency of the side pointers if the B-link is applied to whole of a parallel Btree structure. In this paper, a new concurrency control method using the B-link on the Fat-Btree, a parallel Btree structure, to reduce the frequency of X latches and the range of X latches at a time. To compare the performance of the proposed method and the conventional method, we implemented them on an autonomous-disk system adopting the Fat-Btree. The experimental results with changing update ratio indicate that the proposed method always improves the system throughput, and are especially effective for higher update ratio configuration.

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)]．

†1 東京工業大学大学院情報理工学研究科計算工学専攻
Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†2 日本学術振興会特別研究員 DC
Research Fellow (DC), Japan Society for the Promotion of Science
†3 日本放送協会放送技術研究所
Science and Technical Research Laboratories, Japan Broadcasting Corporation
†4 東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology

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)]．ディレクトリ構成としてFat-Btreeを用いることで，他の並列Btree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．

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[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 にリクエスト移譲が行われるとき，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．

また，B-link は単一 Btree 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 ラッチを獲得回数および範囲を小さくすることが可能である．しかし，並列 Btree ではサイドポインタの一貫性保持が難しく，B-link の並列 Btree への適用は困難であった．

本稿では，並列 Btree 上での B-link を利用を可能とする新たな並行性制御手法を提案する．この手法は，B-link アルゴリズムと LCFB を組み合わせた手法である☆．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．また，自律ディスク上に LCFB および提案手法を実装し，更新要求の割合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．

☆ LCFB だけでなく，INC-OPT や MARK-OPT と組み合わせることも可能である．

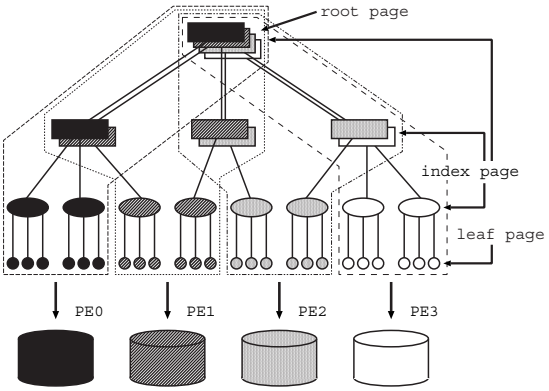


図 1 Fat-Btree
Fig. 1 Fat-Btree.

2. Fat-Btree と並行性制御

2.1 Fat-Btree 構造

Fat-Btree[4] 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，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．

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

表 1　ラッチマトリクス
Table 1　Latch matrix.

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

ドの適合性は表 1 に示される[8]．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．

2.3　従来の並列 Btree 並行性制御方式

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[5] が提案されている．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．

2.4　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を行う．また，経路誤りを検出するために，ページ両端の境界値が設けられているの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．

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

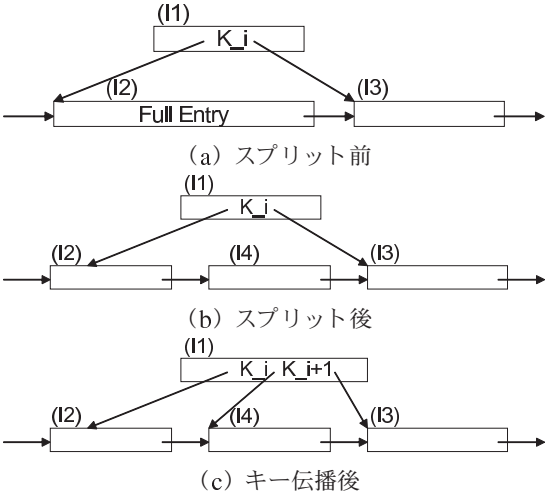
目的の葉ページまで辿り着けば，葉ページに S ラッチを獲得しデータを読む．

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．この手続きの厳密な定義は[7] を参照されたい．

2.5　B^{link}-tree

B^{link}-tree[9),10)] 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 2 は B^{link}-tree のノードを示し，図 3 は B^{link}-tree におけるページスプリットの例を示している．“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．

図 2　B^{link}-tree のノード
Fig. 2　B^{link}-tree node.



（a）スプリット前
（b）スプリット後
（c）キー伝播後

図 3　B^{link}-tree のページスプリット
Fig. 3　A B^{link}-tree page split.

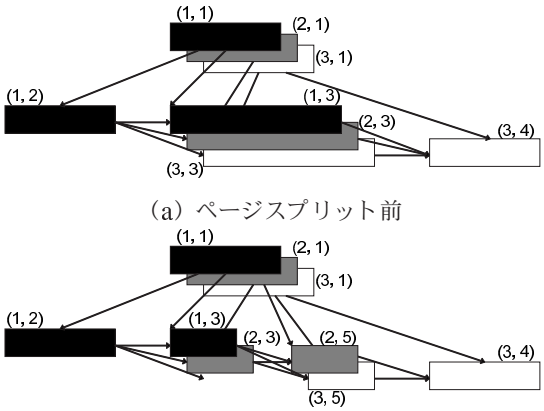
B-link アルゴリズムの検索時のプロトコルは，まず，ルートページを IS モードでラッチした後，以下の処理を繰り返す．

(1)　親ページのキーを比較して，子または兄弟ページのポインタを取得
(2)　親ページのラッチを解放し，子または兄弟ページの 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．B-link アルゴリズムでの更新処理時のプロトコルは，検索と同様の手順で IS モードを用いて葉ページまで辿る．ただし，葉ページでは X ラッチを獲得する．もし，SMO が起きないならば葉ページを更新して終了する．もし SMO が起きるならば，葉ページで SMO を実行する．葉ページのラッチを解放した後，親ノードへと SMO を拡大する．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. 提 案 手 法

単一 Btree 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 向けの新たな並行性制御方式を提案する．



（a）ページスプリット前
（b）ページスプリット後

図 4　B-link を用いたときの Fat-Btree のページスプリット
Fig. 4　A page split on the Fat-Btree with B-link.

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 Btree 全体で B-link アルゴリズムを用いることは難しい．提案手法では，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 Btree で利用することを可能にする．これにより，並列 Btree 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．

3.1　B-link の Fat-Btree への適用

B-link は，サイドポインタにより隣のインデックスノードをリンクしている．また，Fat-Btree では子ページが無くなったページのコピーは消去される．そのため，図 4 のようにページスプリットが発生するとき，インデックスページ（1，2）からページ（3，3）へのリンクを削除しなければならない．よって，その更新のためにページ（1，2）に X ラッチを獲得することが必要となる．葉ノードまでの経路外に存在する（1，2）に X ラッチを獲得しようとすれば，非常に効率が悪い処理が必要となり，B-link を用いている利点が失われてしまう．

この問題を解決するために，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 5）．図 4 の例か

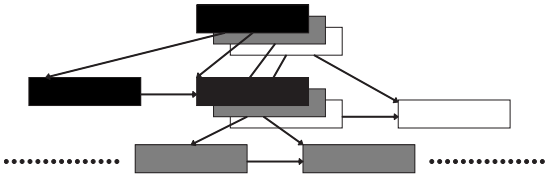


図 5　Fat-B^{link}-tree
Fig. 5　Fat-B^{link}-tree.

図 6　LCFB と組み合わせているときの B^{link}-tree のノード
Fig. 6　B^{link}-tree node with the LCFB.

らもわかるように，インデックスノードのスプリットで削除されるページは，PE 内で左端のページであり，（3，3）のように他の PE からのサイドリンクのみが存在する．したがって，PE 内のみサイドポインタを用いている場合，インデックスページスプリットにおいてサイドポインタの削除は起きないため，上記の問題は発生しない．

また， LCFB と組み合わせるため，インデックスページは図 6 のような構造になる．通常の B^{link}-tree との違いは，下限境界値が存在することである．

3.2　処理プロトコル切替

図 5 の構造を用いた場合，コピーページの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，コピーページの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，提案手法は，コピーページの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，コピーページの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．

LCFB は，同時に複数ノードでラッチを獲得するため，B-link アルゴリズムに比べて更新コストが大きい．しかし，コピーページの存在するノードは非常に少ない☆．そのようなノードの大多数は，ルートノード等の上位ノードであり，ほとんどの更新処理範囲は下位ノードのみであるため，多くの更新処理は，B-link アルゴリズムのみを用いて行われる☆☆．よって，LCFB と組み合わせたことによるシステム性能低下は低いと考えられる．

3.3　検　　索

提案手法では，コピーページが存在するノードでは，LCFB を用い，コピーページが存在しないノードでは，B-link アルゴリズムを用いる． LCFB のようにラッチカップリングを行わない手法と組み合わせた場合には，常にラッチカップリングを行わない☆☆☆．また，インデックスページでは，IS モードのラッチを獲得し，葉ページでは，S モードのラッチを獲得する．

また，LCFB では経路誤りが発生することがある．そのような場合，LCFB では正しい経路に復帰するためにリスタートを行うが，提案手法では，リンクポインタで復帰できる場合には，リンクポインタを辿ることで復帰する．

3.4　更　　新

提案手法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： 葉ページまでの降下は検索と同様である．ただし，インデックスページでは，IX モードのラッチを獲得し，葉ページでは，X モードのラッチを獲得する．もし，SMO が起きないならば葉ページを更新して終了．SMO が起きるならば，B-link のアルゴリズムに従ってボトムアップに X ラッチを獲得し，更新操作を実行する．SMO がコピーページが存在するノードまで及ぶならば，第 2 フェーズに移行する．

第 3 フェーズ： LCFB を用いて，残った SMO に十分な X ラッチを獲得する．その後，第 1 フェーズで最後に実行された更新操作の結果を適切な親ノードに伝播させ，そのノードより上位の残った SMO を実行する．　他のプロセスにより木構造が変更され，コピーページの存在しないノードに X ラッチを獲得した場合，それより上位のノードラッチは解放する．第 1 フェーズに移行し，葉ノードにタプルを挿入する代わりに，インデックスノードにスプリットしたエントリーを挿入する．

提案手法の更新処理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

☆ 4 節の実験で用いた Fat-Btree において，コピーページの存在するインデックスノードは全インデックスノードの約 7.0%であった．

☆☆ 提案手法を用いた 4 節の実験において，コピーページの存在するノードに SMO が及ぶ更新処理は全更新処理の約 0.9%であった．

☆☆☆ INC-OPT や MARK-OPT のようにラッチカップリングを行う手法と組み合わせた場合には，上位ノードではラッチカップリングを行い，コピーページが存在しないノードまで降下してきたら，ラッチカップリングを行わない処理に切り替わる．

```
1   l := H; r := H;
2   Parent := null; Child := ROOT; h := 1; m := 1;
3   while h < min(l, r) do begin
4       Unlatch Parent, IX latch on Child;
5       if Access path error is detected then
6           Unlatch Child; goto 2;
7       Determine NewChild;
8       Parent := Child; Child := NewChild;
9       if Child is not sibling of Parent then begin
10          if Parent is safe then
11              m := h;              /* Marking */
12          push(Parent); h := h + 1;
13      end;
14  end;
15  if Child has no copy then begin
16      Unlatch Parent, X latch on Child;
17      while Child is not target do begin
18          Determine NewChild;
19          Parent := Child; Child := NewChild;
20          Unlatch Parent, X latch on Child;
21      end;
22      while Child is unsafe do begin
23          Update including SMO on Child;
24          Unlatch Child;
25          Child := pop(); h := h - 1;
26          X latch on Child;
27          if Child has its copies then
28              Unlatch Child; l := m; r := h; goto 2;
29      end;
30      Update; Unlatch Child;
31  end;
32  else begin
33      X latch on Child and its copies, Unlatch Parent;
34      while h < r do begin
35          Determine NewChild;
36          Parent := Child; Child := NewChild;
37          if Parent is safe then
38              m := h;              /* Marking */
39          push(Parent); h := h + 1;
40          if Child has no copy then
41              Release all granted latches; l := H; goto 3;
42          X latch on Child and its copies
43      end;
44      if X latches are not sufficient for SMOs then
45          Release all granted latches; l := m; goto 2;
46      else begin
47          Update including all SMOs;
48          Release all granted latches;
49      end;
50  end;
```

図 7 提案プロトコル
Fig. 7 Proposed protocol.

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．最後に，第 1 フェーズの更新を伝播させるべき親のレベルを r とする．このとき，LCFB と組み合わせた提案手法のプロトコルは図 7 で定義される．

3.5 正 当 性

Btree の並行性制御は，更新発生時にも目的の葉ページまでの経路を保証しなければならない．

B-link のアルゴリズムによる更新はノードごとにボトムアップに処理される．このとき，あるノードが分割された後，親にエントリー更新が伝播されるまで，木の構造としては不完全な状態にある．しかし，分割してできた新しいノードへ移ったエントリーには，サイドポインタで新ノードへリンクしているため，アクセス可能である．そのため，B-link のアルゴリズムを用いているコピーページの存在しないノードでは問題は発生しない．

一方で，コピーページの存在するノードでは，全 SMO に必要な X ラッチを獲得した後に更新を行う LCFB を用いる．したがって，この更新により問題が発生することはない☆．

コピーページの存在するノードまで SMO が及ぶ場合，不完全な状態でリスタートし，LCFB に処理方法を変更する．このときも不完全な状態ではあるが，サイドポインタでリンクされているため，そのノード以下の全葉ページへの経路は保証されており，問題は発生しない．また，LCFB による X ラッチ範囲がコピーページの存在しないノードまで及んだ場合，コピーページの存在するノードで獲得したラッチは解放し，B-link アルゴリズムによる処理方法に変更する．上記の切り替えをラッチ獲得ごとにコピーページの有無を判別しているため，同一ノードで二つアルゴリズムが混同することはなく，それにより問題が発生することはない．

以上より，B-link アルゴリズムおよび LCFB は，それぞれの更新によって問題が発生することはなく，その処理が混同されるノードは存在しないため，提案手法は更新発生時にも目的の葉ページまでの経路を保証している．

4. 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[11] に提案手法を実装し実験を行う． 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ネットワークスイッチを用いて，実験環境を構成した． 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 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．

☆ LCFB はラッチカップリングを行っていないため，経路誤り等の問題が発生するが，LCFB ではそれを解決する処理が行われている[7]．

表 2　実験システムの構成
Table 2　Experimental environment.

#Nodes:	64 (Storages), 16 (Clients)
CPU:	AMD Athlon XP-M 1800+ (1.53GHz)
Memory:	PC2100 DDR SDRAM 1GB
HDD:	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
Table 3　Parameters used for the experiments.

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

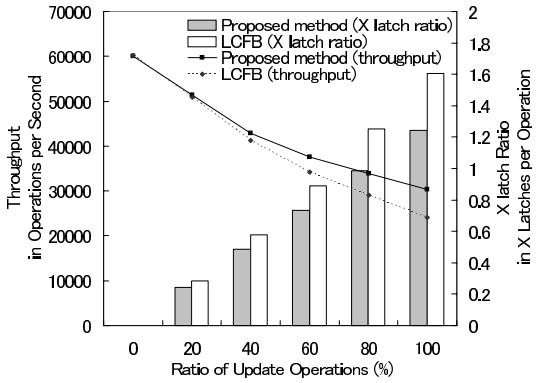


図 8　更新要求の割合を変化させたときの比較
Fig. 8　Comparison with changing update ratio.

クライアント PC（1 台あたり並行して 64 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，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 LCFB のスループット及び 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．

更新要求の割合が増えるに従い，LCFB のスループットが大きく低下していくのに対し，更新要求が多い場合でも提案手法は高いスループットを維持している．これは，図 8 からもわかるように，提案手法の X ラッチ獲得頻度のほうが小さいからである．全 SMO に必要なラッチを獲得した後に更新処理を行う LCFB は，その過程で同ページに X ラッチを獲得する可能性が高い．それに対し，提案手法は単一 PE 内の処理に B-link を用いていることにより，同ページに X ラッチを獲得する可能性は極めて低く，X ラッチ獲得数が減らすことが可能である．

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 Btree の並行性制御として新手法を提案した．これは，単一 Btree 上で優れた並行性制御を実現できる B-link を並列 Btree に適用し，従来の LCFB と B-link のアルゴリズムを組み合わせた並行性制御方式である．提案手法では，単一 PE における更新処理は B-link のアルゴリズムを用い，複数の PE に及ぶ更新処理は LCFB を用いる．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，MARK-OPT と同様に INC-OPT 方式の拡張である PO-P 方式[12] との比較を検討している．PO-P 方式は，INC-OPT と PO[13] 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　B-link 及び 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 (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．

参　考　文　献

1) Copeland, G., Alexander, W., Boughter, E. and Keller, T.: Data Placement in Bubba, *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, ACM SIGMOD, pp. 99–108 (1988).
2) Ghandeharizadeh, S. and DeWitt, D. J.: Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines, *Proceedings of 16th International Conference on Very Large Data Bases (VLDB'90)*, pp. 481–492 (1990).
3) DeWitt, D.J. and Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems, *Communications of the ACM*, Vol. 35, No.6, pp.85–98 (1992).
4) Yokota, H., Kanemasa, Y. and Miyazaki, J.: Fat-Btree: An Update-Conscious Parallel Directory Structure, *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*, IEEE Computer Society, pp.448–457 (1999).
5) Miyazaki, J. and Yokota, H.: 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，第 16 回データ工学ワークショップ（DEWS2005）論文集，2A-o4，電子情報通信学会データ工学研究専門委員会 (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，第 16 回データ工学ワークショップ（DEWS2006）論文集，7C-o3，電子情報通信学会データ工学研究専門委員会 (2006).
8) Gray, J. and Reuter, A.: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Francisco, California, USA (1992).
9) Lehman, P.L. and Yao, S.B.: Efficient Locking for Concurrent Operations on B-trees, *ACM Transactions on Database Systems*, Vol.6, No.4, pp.650–670 (1981).
10) Lanin, V. and Shasha, D.: A Symmetric Concurrent B-tree Algorithm, *Proceedings of the Fall Joint Computer Conference (FJCC'86)*, ACM and IEEE, pp.380–389 (1986).
11) Yokota, H.: Autonomous Disks for Advanced Database Applications, *Proceedings of 1999 International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, IEEE Computer Society, pp.435–442 (1999).
12) Amarmend, D., Aritsugi, M. and Kanamori, Y.: PO-P: A Concurrency Control Protocol for Parallel B-trees, *IPSJ Transactions on Databases*, Vol.45, No.SIG14, pp.30–38 (2004).
13) Mond, Y. and Raz, Y.: Concurrency Control in B+-Trees Databases Using Preparatory Operations, *Proceedings of 11th International Conference on Very Large Data Bases (VLDB'85)*, pp.331–334 (1985).