データベースシステム　125－70
（２００１．７．１９）

並列ディレクトリ構造Fat-Btreeの並行性制御とその評価

宮崎 純
miyazaki@jaist.ac.jp
北陸先端科学技術大学院大学
情報科学研究科
〒923–1292 石川県能美郡辰口町旭台 1–1

横田 治夫
yokota@cs.titech.ac.jp
東京工業大学
学術国際情報センター 情報基盤部門
〒152–8552 東京都目黒区大岡山 2–12–1

あ ら ま し　我々が提案している無共有並列計算機向けのディレクトリ構造である Fat-Btree に適した並行性制御 INC-OPT 方式を提案する．従来の B-tree の並行性制御方式に比べて INC-OPT は並列 B-tree 構造の性質と非常に親和性がよい．これは，INC-OPT 方式が無共有並列計算機向けに設計されているのに対して，従来の方式がいずれも単一プロセッサもしくは SMP を仮定して設計されているからである．nCUBE3 上を用いた実験を通して，INC-OPT は更新操作が多い場合でもスループットの低下が小さく，また，SIB と CWB とを比較しても，Fat-Btree は高いスループットが得られることを示す.

和文キーワード 並列データベース，B-tree，並行性制御，性能評価，無共有並列計算機

Concurrency Control and Its Evaluation of the Fat-Btree

Jun MIYAZAKI
miyazaki@jaist.ac.jp
School of Information Science,
Japan Advanced Institute of
Science and Technology
1–1 Asahidai, Tatsunokuchi,
Ishikawa 923–1292, Japan

Haruo YOKOTA
yokota@cs.titech.ac.jp
Global Scientific Information and
Computing Center,
Tokyo Institute of Technology,
2–12–1 Ookayama, Meguro,
Tokyo 152–8550, Japan.

Abstract　We propose a concurrency control, named INC-OPT method, for the Fat-Btree. The Fat-Btree is designed as an index structure for shared-nothing parallel databases. The INC-OPT is a better concurrency control for parallel B-trees than conventional ones, because the INC-OPT primarily takes account of shared-nothing environments whereas the others were for a uni-processor or an SMP. In this paper, we give a practical result using an nCUBE3 machine that the INC-OPT enables the Fat-Btree to keep high throughput even under the high update ratio. Also, we show that the Fat-Btree outperforms other parallel B-trees such as an SIB and a CWB.

英文 key words
parallel database, B-tree, concurrency control, performance evaluation, shared-nothing parallel computers

1 はじめに

データベース用無共有並列計算機では，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上で並列に実行されることが望ましい．各PE間で負荷を均等にすることは処理性能向上につながり，負荷を均等化する為のデータの分配方式が重要となる [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値域分割では可能な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，分割の基準値を静的に決定されるので，データ更新によってデータ配置の偏りが生じた時に均一化するコストが非常に大きくなる欠点がある．

これらの両方式の欠点を解消する為に，データ分配方式として並列B-treeを利用する研究がある [4]．並列B-treeを利用することによって，両方式の欠点を解消でき，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ではディレクトリの更新時に問題が生じる．アクセスを分配するため全てのPE(Processor Element) にディレクトリ全体をコピーして配置 (Copy-Whole-Btree: 以下CWB方式と呼ぶ) すると，ディレクトリ更新時に全PE上のコピーをロックして更新する必要があり，システムのスループットを大きく低下させる．一方，更新を簡略化するために1つのPEのみに全ディレクトリを配置 (Single-Index-Btree: 以下SIB方式と呼ぶ) すれば，そのPEにアクセスが集中しボトルネックとなる．

そこで我々は，新しい並列B-tree構造としてFat-Btreeを提案している [5, 6, 7]．ディレクトリ構成としてFat-Btreeを用いることにより，従来の並列B-treeで生じるディレクトリ更新によるスループット低下や，少数のPEへのアクセス集中といった問題点を解決できることが確率モデルにより明らかにされている [5, 7]．また，レンジ問い合わせが並列に高速実行できる [8]．しかし [5, 7] では，並行性制御のオーバヘッドを無視していた．また [9] では，並行性制御方式としてB-OPT[10] を採用したが，B-OPTは並列B-treeに対して，更新時，特にB-tree構造変化を起こす操作 (*SMO*: Structure Modification Operation) 時に大きな性能低下をもたらす．他にも，優れたB-treeの並行性制御方式と見なされているARIES/IMも，並列B-treeには適さない．

本論文では，まず従来のB-treeの並行性制御方式がなぜ並列B-treeに向かないのかについて議論し，その問題点を解決する新しい並行性制御方式INC-OPTを提案する．その後，無共有並列計算機nCUBE3にARIES/IM，B-OPTおよびINC-OPTを実装し，INC-OPTが更新操作 (SMO) が多い場合にもスループットを保つことができることを示す．

2 並列B-tree

2.1 従来の並列B-tree構造

並列B-treeは，完全一致検索だけでなくレンジ検索が容易であるというB-treeが持つ性質の他に，並列化によりI/Oバンド幅を大きくできることが魅力である [4]．従来の並列B-tree構造を分類すると，2種類に大別できる．

- Single-Index-Btree (SIB): データを格納する葉ページを各PEに分散するが，木は一つである (図1)．以降簡単のため，木構造，つまり全てのインデックスページは1つのPEに格納されているとする．SIBはページのコピーを持たないため，更新操作は高速であるが，木構造が一つのため検索は低速である．

- Copy-Whole-Btree (CWB): データを格納する葉ページを各PEに分散し，ディレクトリを格納するインデックスページは全てのPEが木構造全体を持つようにコピーし，それぞれのPEに格納する (図2)．CWBはページのコピーを持つため，複数のパスが確保でき検索は高速となるが，その反面更新時にはコピー間の同期をとるオーバヘッドが大きい．

従来の並列B-treeの研究は，主に検索の高速化にのみ注目され，更新処理の高速化にはあまり触れられていなかった．検索も更新も多いOLTPなど分野では，従来の並列B-treeでは不十分である．そこで我々は，更新処理にも強い，新しい並列B-treeであるFat-Btreeを提案している [5, 6, 7].

2.2 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の一種である．Fat-Btreeは，B-treeの葉ページ (データページ) 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

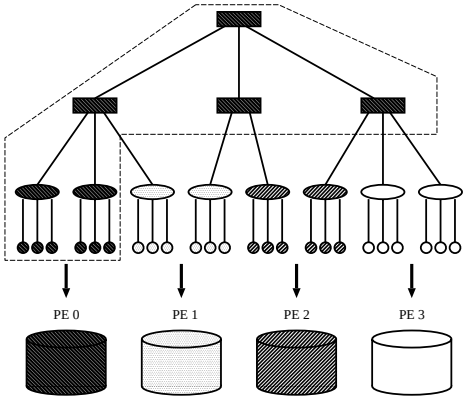

図 1: Single-Index-Btree

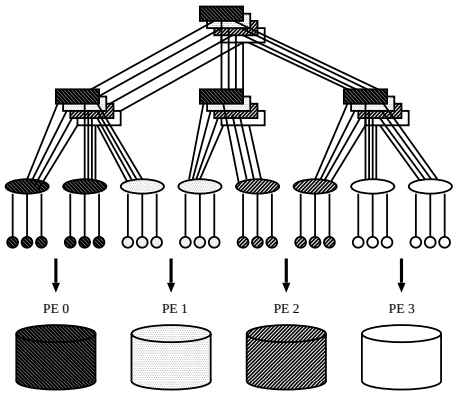

図 2: Copy-Whole-Btree

ページへのアクセスパスを含むインデックスページのみを再帰的に配置する．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 3)．Fat-Btree の名前は，並列計算機向け相互結合網の一つである，Fat-Tree[11] と B-tree とを融合したことに因んでいる．Fat-Tree は，ルートに近付くほどパスが多くなる木構造であり，トラフィックが集中するルート付近のバンド幅を大きくすることが目標となっている．これは，B-tree を含む木構造一般に応用することが可能である．

Fat-Btree では，多数の葉ページへのパスを含む上位のインデックスページは，コピーされ多くの PE に配置される．特にルートページは，全ての葉ページへのパスを含むので，全ての PE にコピーされる．一方，少数しか葉ページへのパスを持たない下位のインデックスページは，他の PE にコピーを持つ確率が低く，コピーが存在してもその数は少ない．この Fat-Btree の性質は，検索，

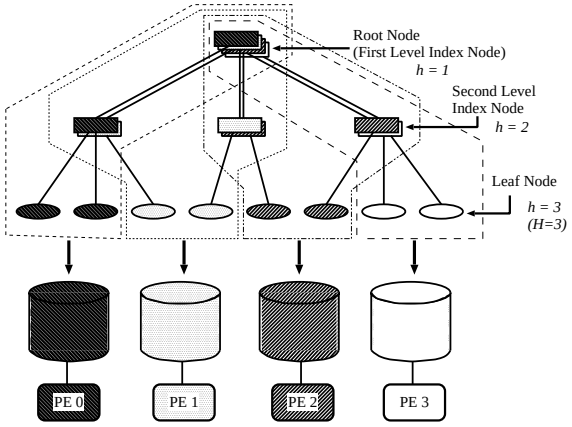

図 3: Fat-Btree

更新の両方に貢献する．なぜならば，Fat-Btree では，

- 上位のインデックスページは高い確率で参照され，ページのコピー数が多いため複数の PE で並列に木を辿ることが可能である．特に，ルートページは参照される確率は 1 であるが，全ての PE にコピーが置かれるため，全ての PE でアクセスリクエストを受け取り，並列に処理することができる．アクセスリクエストに偏りがなければ，コピーの少ない下位のインデックスページへ操作が進んでも全 PE に一様に分散される．
- インデックスページの更新は，ページがオーバフロー (アンダーフロー) するときのスプリット (統合) 操作により引き起こされ，木の下部から上部へとボトムアップに行われる．パス中の全てページがオーバフローやアンダーフローする確率は低いため，下位のインデックスページほど高い確率で更新処理が行われる．Fat-Btree では，下位のインデックスページほどコピーを持つ確率が低いため，更新操作に伴う PE 間の同期オーバヘッドは小さい．一方，ルートページにまで更新操作が及ぶときには，全 PE に存在するページの同期が必要であるが，その確率は極めて低い．

換言すれば，Fat-Btree は，検索に関しては CWB の性質を，更新に関しては SIB の性質を有し，両者の良い特性のみを継承している．

2.2.1 データ構造

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 に処理を委譲する．円滑に処理を引き継ぐためには，リモートページへのポインタを保持する必要がある．ローカルとリモートのページへのポインタを統一的に扱うために，ディスク上の物理ページ番号と PE 番号を組み合わせた**論理ページ番号**を用いる．この論理ページ番号により，全ての PE 上の全てのページが一意に認識される．

インデックスページはコピーされて複数の PE で保持されることがある．もし同一 PE 上にコピーが存在する場合は，親ページから同一 PE 内の子ページへ直接ポインタでリンクされる．同一 PE 内に子ページが存在せず，他の PE に子ページが唯一存在する時も，親ページから直接ポインタで子ページにリンクされる．そうではなく，同一 PE に子ページが存在せず，他の複数の PE に子ページが存在する場合は，その複数のポインタは**ポインタページ** [6] に格納される (図 4 参照)．

2.2.2 インデックスページのキャッシュ

通常の DBMS では，頻繁に参照されるページは主記憶上にキャッシュされる．B-tree に限定すれば，木の上位ページほど参照される確率が高いため，これらのページをキャッシュするのが賢明である．B-tree のキャッシュ置換方式はいくつかあるが [12]，本稿では LRU を仮定する．

並列 B-tree に関しては，Fat-Btree のキャッシュの効果は非常に高い．なぜなら，各 PE は部分木しか持たないので，システム全体から見れば，より多くのインデックスページがどこかの PE にキャッシュされていることになる．Fat-Btree では，同一 PE に存在しないページの参照は他の PE に転送されて実行されるため，どこかの PE で参照されるべきページがキャッシュされていることは，アクセス時間の短縮に結び付く．昨今の計算機は，ディスクへのアクセス時間よりもネットワークを介したメッセージ転送時間の方が一桁以上高速である．つまり，ローカルディスクを参照するよりも，もし他の PE 上のキャッシュにデータが存在する確率が高いならば，その PE に処理を委譲するのは賢明である．たとえ主記憶にデータ存在せず結局ディスクを参照することになったとしても，メッセージ転送コストは無視できるほど小さい．

一方，SIB はインデックスページを保持する PE が一つであるので，限られたキャッシュ容量に多くのインデックスページを保持することはできない．CWB は，各 PE が同じインデックスページのコピーを持つので，どの PE も同様なページをキャッシュしているだけで，システム全体から見れば，多くの**異なる**インデックスページをキャッシュしている訳ではない．つまり，他の PE に処理を転送してもキャッシュにはヒットしない．以上の理由により，SIB，CWB ともにキャッシュのヒット率は低い．

3 Fat-Btree の並行性制御

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，B-tree 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．

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 [13]．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．

表 1: ラッチマトリックス

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

複数の PE にデータのコピーが存在する場合を考える．もしラッチの処理が各 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，X モードは，コピーを持つ全ての PE でラッチを獲得する必要がある．すなわち，S，SIX，X モードのラッチはデータのコピーが存在する時，PE 間の同期オーバヘッドが生じる．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ので，デッドロックを回避するために PE 番号の昇順または他の順序に従いラッチを獲得しなければならない．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．

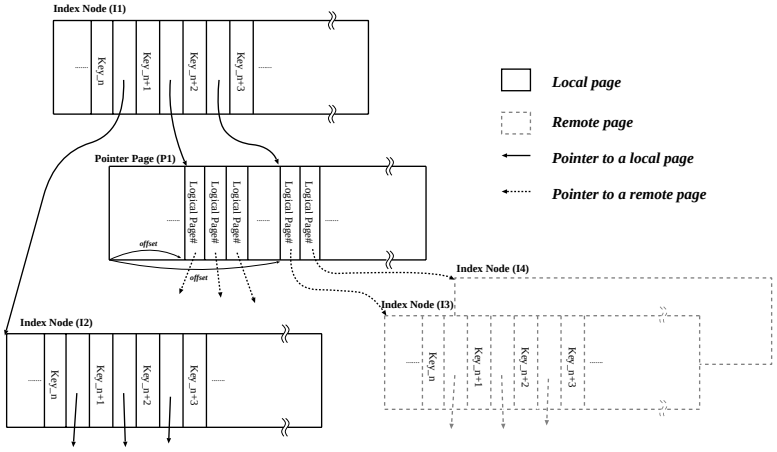

図 4: Fat-Btree のデータ構造

3.2 従来の B-tree の並行性制御

優れた B-tree の並行性制御方式として，B-OPT [10]，B-link [14]，OPT-DLOCK [15]，ARIES/IM [16] などがある．

B-link は，サイドポインタにより隣のインデックスノードをリンクすることを仮定している [14]. 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．この時，サイドポインタを一貫的に維持しようとすれば，デッドロックの可能性がある．OPT-DLOCK も IX から X モードへの変換を伴うためデッドロックの可能性がある [15]. 一方，B-OPT，ARIES/IM はデッドロックフリーである．

B-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．

B-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: ルートから SIX ラッチを葉ページまで取得する．次に SMO に関わるページのみを X ラッチに変換し更新処理を行う．それ以外のページのラッチはすぐに解放される．

第 2 フェーズは，B-SIX プロトコルと呼ばれる [10]．この間，IS ラッチを用いる検索は，SIX ラッチと衝突しないため，X ラッチに無関係なパスの検索は同時実行できる．SIX ラッチも X ラッチも無関係なパスは更新処理も同時実行可能である．

3.1 節で言及したように，ルートページは全 PE にコピーが存在するので，SIX ラッチ獲得には全 PE で同期が必要である．同様に上位のページについても，多数の PE にコピーを持つので，同期オーバヘッドは大きい．つまり，Fat-Btree の上位ページには同期が必要な S，SIX，X ラッチを獲得することを可能な限り避けなければならない．

一方，ARIES/IM では各ページに SM_Bit と呼ぶフラグを持ち，SM_Bit がセットされているページは SMO がそのページで進行中である可能性を示す．さらに**木ラッチ**と呼ばれる B-tree 全体のラッチを用いて，SMO が起きても一貫的な B-tree 構造を保証する [16].

検索のプロトコルは，B-OPT とほぼ同一である．しかし，SM_Bit がセットされているページに遭遇した時は，現在獲得しているラッチを解放し，S モードの木ラッチを獲得することにより SMO の終了を待つ．この木ラッチはすぐに解放し，検索を再開する．

一方更新プロトコルは多少複雑である．

1. まず，IX ラッチカップリングで葉ページまで辿る[1]．もし SMO が起きないならば，葉ページを更新して終了．

2. もし SMO が起こるのならば，SMO に関わるページをキャッシュに固定する．

3. X モードの木ラッチを獲得し，葉ページの SM_Bit をセット，葉ページの更新，最後に葉ページのラッチを解放する．

4. 親ページの X ラッチを獲得し，SM_Bit をセット，ページを更新し，ラッチを解放する．この操作を SMO の影響を受けるページにボトムアップに繰り返す．

5. セットした全ての SM_Bit をリセットし[2]，X モードの木ラッチを解放する．

ARIES/IM がルートページや上位ページの X ラッチを獲得するのは，ルートページが SMO に巻き込まれるときである．しかし，ある更新操作が SMO を処理している間は，X モードの木ラッチが獲得されているため，その更新操作とは全く独立のパスに対する SMO であっても，同時に実行できない．つまり，全ての SMO を起こす更新操作は逐次に実行される[3]．また，木ラッチを分散環境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．

以上の議論を要約すれば，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．

条件 3.1 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．

- デッドロックフリーである
- ルートおよび木の上位ページに可能な限り *X*，*SIX*，*S* ラッチを獲得しない
- 短時間であっても木全体をラッチすべきではない

□

[1]途中で SM_Bit がセットされたページに遭遇すれば，検索時と同様の処理を行う．

[2]SM_Bit がセットされたページに遭遇した操作が，S モードの短時間の木ラッチを獲得することにより SMO が終了したことを判定し，この操作に SM_Bit をリセットさせる方法もある [16]．

[3]木ラッチを木ロックに変更し，SMO が実際に起こるまでは IX モードの木ロックをとり，SMO 実行直前に X モードに変換する方法で複数の SMO が扱えるが，しかし，このロック変換はデッドロックの可能性がある [16]．

3.3 INC-OPT 方式

並列 B-tree 向けの INC-OPT(INCremental OPTimistic) 並行性制御方式を提案する．INC-OPT 方式は B-OPT の拡張であるが，第 2 フェーズとして B-SIX を用いない．もし第 1 フェーズで葉ページがスプリットを起こすならば一度全てのラッチを解放し，第 2 フェーズは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 INC-OPT が 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．この時，INC-OPT プロトコルは図 5 で定義される．

```
1  l := H;
2  Parent := null; Child := ROOT; h := 1;
3  while h < l do begin
4      Child に IX ラッチ，Parent のラッチ解放;
5      NewChild を決定;
6      Parent := Child; Child := NewChild; h := h + 1;
7  end;
8  while h ≤ H do
9      Child とそのコピーに X ラッチ; h := h + 1;
10 if 獲得した X ラッチが SMO に不十分 then begin
11     全てのラッチを解放;
12     l := l - 1; goto 2;
13 end;
14 else begin
15     更新操作 (SMO) を実行;
16     全てのラッチを解放;
17 end;
```

図 5: INC-OPT プロトコル

この INC-OPT は，(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ので，条件 3.1 を満たしている．最大フェーズ数は高々H に抑えられ，かつ SMO が上位ページまで影響するようなケースは極めて稀である．多くの場合は下位のページのみで SMO が実行される．すなわち，INC-OPT プロトコルは B-tree の更新の性質に非常に合致している．

4 性能評価

Fat-Btree の評価を無共有並列計算機 nCUBE3 上で行った．評価に使用した nCUBE3 のうち

64PE(I/O PE) は2系統のSCSIバスを持ち，各バスにはハードディスク (Seagate SN31230N) が合計128台接続されているが，各PEの一系統のハードディスクのみを用い，もう一方のハードディスクはデータのバックアップ用に用いた．nCUBE3の諸元は表2の通りである．

実装した実験システムのパラメタは，表3で示される．なお，キャッシュのサイズは，PE数を変化させたときキャッシュの影響を公平にするため，システム全体で1600ページとした．また，実験に使用したB-treeの各ページ内のデータの占有率は，0.5から1.0の間の値を各ページごとに乱数で決定した．この占有率を用いれば，更新操作の10%がSMOを起こす．

表 2: nCUBE3 の諸元

項目	パラメタ
CPU	45MHz カスタム
メモリ	32MB
通信スループット	18.46 MB/s
通信オーバヘッド	127.2 μs
ディスク読み出し	最大 5.18 MB/s
ディスク書き込み	最大 4.21 MB/s

表 3: システムパラメタ

項目	パラメタ
ローカル PE のラッチ	11 μs
リモート PE のラッチ	310 μs
cache サイズ	合計 1600 ページ
cache hit 時のページ読出し	10 μs
cache miss 時のページ読出し	11.7 ms
ページサイズ	4 KB
タップルサイズ	208 B
データベースサイズ	512K – 8M タップル
インデックスページの鍵数	最大 507
葉ページのタップル数	最大 19
木の高さ	3($\leq$2M タップル)
	4($\geq$4M タップル)

以降の実験は，キーをランダムに選び，検索と挿入操作を合計1万回実行したときのスループットを計測する．操作はI/Oプロセッサとは別のPEで生成され，メッセージを介して発行し，そしてその結果を得る．操作を送るのは，SIBではインデックスを持つPEに対して，CWBとFat-Btreeではランダムに選択したPEに対してである．

4.1 並行性制御方式の比較

図6は，PE数64，データベースサイズ2Mタップルの時のFat-Btreeを，横軸として更新操作の割合を0%から30%に変化させ，INC-OPTとB-OPT，ARIES/IMのスループットを縦軸として比較したグラフである．

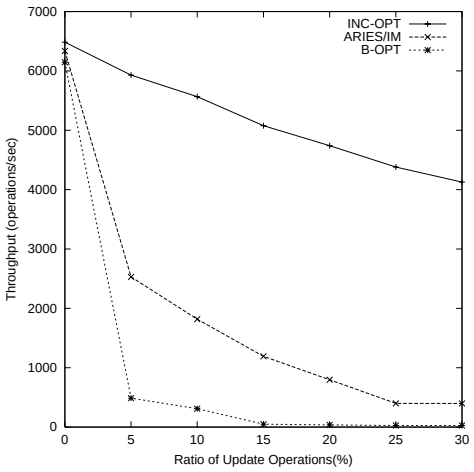

図 6: 64PE時の並行性制御方式の比較 (2Mタップル)

B-OPTは，更新操作が入ると急激にスループットが落ちる．これは，SMOが発生することにより，第2フェーズとしてB-SIXプロトコルが実行され，木の上部がSIXモードによりラッチされるからである．特にルートのラッチは全PEの同期を必要とし，オーバヘッドは著しく大きい．

ARIES/IMは，ルートに対するSIXラッチのような大きなオーバヘッドがないため，更新操作が増えてもB-OPTのように急激なスループットの落ち込みはない．しかし，SMOが発生すれば，木ラッチの獲得，すなわち大域的な同期が必要である．この同期はある特定のPEに処理させているため，SMOが増えればこのPEがボトルネックとなる．このため，更新操作の割合が増えるに従いスループットが低下する．

一方INC-OPTは更新操作が増えるに従い，わずかながらループットが低下するものの，B-OPTやARIES/IMのようなスループット大幅は落ち込みはない．これは並列B-treeに要求される条件3.1を満たしているからである．

4.2 並列 B-tree の比較

更新操作の割合が10%を仮定すると，図6より，たとえSIBやCWBにARIES/IMを適用しても，並行性制御のオーバヘッド (木ラッチ) により，スループットは毎秒2000操作を越えない．そこで，SIBとCWBにもINC-OPTを適用した場

合の Fat-Btree との性能の比較を行う．

図 7 は，横軸としてデータベースサイズを変化させたときの各並列 B-tree のスループットを縦軸に取ったものである．いずれの方式も 2M タップルと 4M タップルの間でスループットの低下が見られる．これは，木の高さが 3 から 4 へ変化したためである．Fat-Btree が特にスループットの低下が著しいのは，葉ページまでに参照されるポインタページ数の増加によるものである．ディスクの読出し回数について 2M と 4M タップル時の比較をすると，CWB では 9.3%の増加であるが，Fat-Btree では 27%の増加となった．しかし，8M タップルのときでも，Fat-Btree は CWB の 1.7 倍のスループットが得られた．

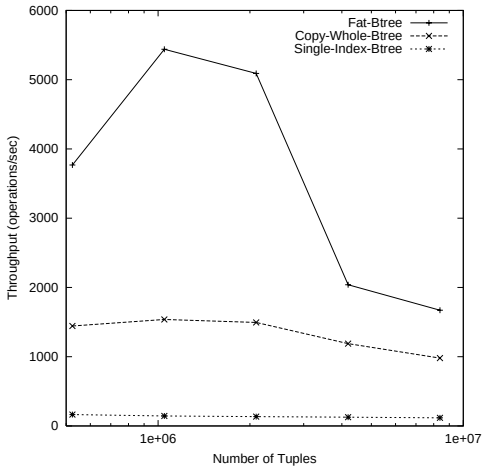

図 7: 64PE 時の各並列 B-tree の比較 (更新操作 10%)

5 おわりに

無共有並列計算機向けのディレクトリ構造である Fat-Btree に適した並行性制御方式 INC-OPT を提案した．従来の並行性制御方式はいずれも，単一プロセッサもしくは SMP を仮定して設計されているのに対して，INC-OPT が並列 B-tree の並行性制御の必要条件を満たし，無共有並列計算機向けに設計されている．このため，INC-OPT は並列 B-tree 構造の性質にうまく合致している．

nCUBE3 上を用いた実験を通して，INC-OPT は他の従来の並行性制御方式と比較して，更新操作が多い場合でも著しくスループットが改善されることを示した．また，SIB と CWB とを比較しても，Fat-Btree は高いスループットが得られることが証明された．

本稿ではアクセスパターンが均一と仮定したが，実際はアクセスパターンは偏っている場合が多い．現在，各 PE のアクセス頻度だけでなくデータ量も考慮したページのマイグレーション方式を検討中である．

参考文献

[1] G. Copeland, W. Alexander, E. Boughter and T. Keller: “Data Placement in Bubba”, Proc. of ACM SIGMOD Conf. ’88, pp. 99–108 (1988).

[2] S. Ghandeharizadeh and D. J. DeWitt: “Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines”, Proc. of VLDB Conf. ’90 (1990).

[3] D. DeWitt and J. Gray: “Parallel Database Systems: The Future of High Performance Database Systems”, Communications of the ACM, **35**, 6, pp. 85–98 (1992).

[4] B. Seeger and P. Larson: “Multi-Disk B-trees”, Proc. of ACM SIGMOD Conf. ’91, pp. 436–445 (1991).

[5] 金政, 宮崎, 横田：“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 DE97–77 (AI97–44) (1997).

[6] 金政, 宮崎, 横田：“更新を考慮した並列ディレクトリ構成 Fat-Btree の実装に関する 考察”, 第 9 回データ工学ワークショップ論文集 (1998).

[7] H. Yokota, Y. Kanemasa and J. Miyazaki: “Fat-Btrees: An Update-Conscious Parallel Directory Structure”, Proc. of IEEE ICDE Conf. ’99, pp. 448–457 (1999).

[8] 風戸, 横田：“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 回データ工学ワークショップ論文集 (2001).

[9] 宮崎, 横田：“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”, 信学技報 DE99–77 (1999).

[10] R. Bayer and M. Schkolnick: “Concurrency of Operations on B-trees”, Acta Informatica, **9**, 1, pp. 1–21 (1977).

[11] C. E. Leiserson: “Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing”, IEEE Transactions on Computer, **34**, 10, pp. 892–901 (1985).

[12] C. Y. Chan, B. C. Ooi and H. Lu: “Extensible Buffer Management of Indexes”, Proc. of VLDB Conf. ’92, pp. 444–454 (1992).

[13] J. Gray and A. Reuter: “Transaction Processing: Concepts and Techniques”, Morgan Kaufmann, San Francisco (1993).

[14] P. Lehman and S. Yao: “Efficient Locking for Concurrent Operations on B-trees”, ACM TODS, **6**, 4, pp. 650–670 (1981).

[15] V. Srinivasan and M. J. Carey: “Performance of B-Tree Concurrency Control Algorithms”, Proc. of ACM SIGMOD Conf. ’91, pp. 416–425 (1991).

[16] C. Mohan and F. Levine: “ARIES/IM: an Efficient and High Concurrency Index Management Method Using Write-Ahead Logging”, Proc. of ACM SIGMOD Conf. ’92, pp. 371–381 (1992).