データベースシステム 119−68
（1999. 7. 23）

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

宮崎 純† 横田治夫††

我々の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Fat-Btreeは，従来の単一インデックス方式や，インデックス全コピー方式と比較して，高いスループットが得られることが，確率モデルにより明らかにされている．本論文では，Fat-BtreeをnCUBE3上に実装することにより，様々なオーバヘッドを含めた性能評価を行った．その結果，Fat-Btreeは従来の方式に比較して高い性能が得られることが分かった．

An Implementation and Evaluation of Fat-Btrees

JUN MIYAZAKI† and HARUO YOKOTA††

We have proposed a parallel index structure for shared-nothing machines, named a Fat-Btree, and shown by using a probability model that the Fat-Btree has quite better throughput than both a single index (SIB) and a copy whole (CWB) strategies. In this paper, we implemented the Fat-Btree to evaluate the performance including various overheads that were not considered in the previous research. The results indicate that the read throughput of the Fat-Btree outperforms both the SIB and the CWB.

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Btreeを利用する研究がある[4)]．並列Btree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Btree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Btree構造としてFat-Btreeを提案している[5)~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Btree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)]．しかし[5),7)]では，ロックのオーバヘッドを無視していた．

本論文では，Fat-Btreeを無共有並列計算機nCUBE3に実装し，その実装方式を示すとともに，分散したデータに対するロックのオーバヘッドを考慮した処理性能についての初期評価を明らかにする．さらに，実装についての問題点についても言及する．

2. Fat-Btree

Fat-Btreeは，Btree全体をページ単位でPE間で分配する並列Btreeの一種である．

† 北陸先端科学技術大学院大学 情報科学研究科
School of Information Science, Japan Advanced Institute of Science and Technology
†† 東京工業大学 情報理工学研究科 計算工学専攻
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

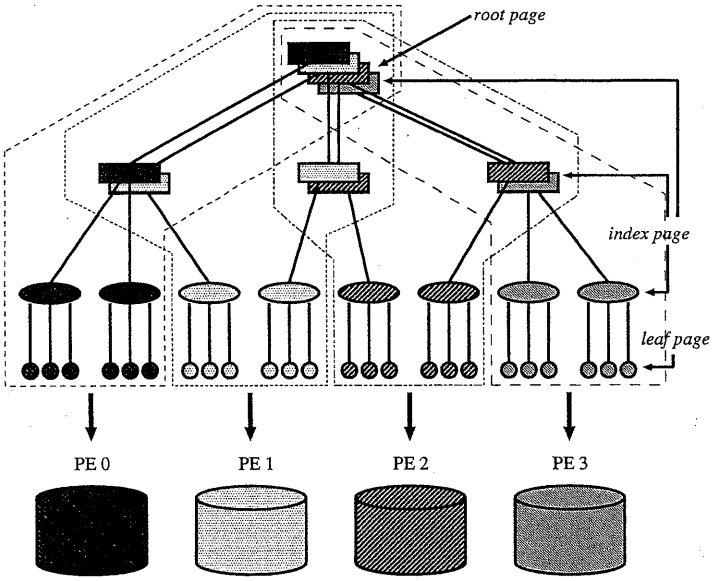

図1　Fat-Btree

Fat-Btree は，Btree 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 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，Btree 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)．これは相互結合網において，ルートに近付くほどパスが多くなる木構造 Fat-Tree[8] に類似するため，提案する Btree 構造を Fat-Btree と名付けた．

Fat-Btree では，多くの葉ページへのパスに共有される比較的上位のインデックスページは，コピーされ多数の PE に配置される．特にルートページは，全ての PE にコピーされる．しかし，ディレクトリの更新は，確率的に比較的下位のインデックスページのみで起こり，それらのインデックスページは少数の PE にしか配置されていない．従って，CWB 方式に比べてディレクトリ更新時の処理が軽くなることが期待される．また，ルートページからその PE に配置されている葉ページまでの全てのパスはその PE 上にあるので，各 PE ごとにデータの検索が可能で，SIB 方式のように特定の PE での処理の集中を避けることができる．

3. Fat-Btree の実装

3.1 データ構造

Fat-Btree ではデータ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，ある PE 上でデータ検索のためにアクセスパスをたどっている時に，必要なインデックスページがその PE 上に存在しなかった場合は，その PE は処理に必要なインデックスページを格納している PE に引き継ぐことになる．円滑に処理を引き継ぐためには，子ページへのポインタを保持する必要がある．

通常の Btree では，インデックスページ内には子ページへのポインタとして子ページの物理ページ番号が保存されるが，Fat-Btree では，複数の PE 上のページにポインタが跨ることがあるため，この物理ページ番号を用いるポインタを採用できない．そこで，複数の PE 上のページを特定するための論理ページを Fat-Btree のポインタとして用いる．また，子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ため，複数のポインタを保持する仕組みが必要である．もし子ページが複数ある場合，それらの論理ページ番号に基づくポインタを別ページ (ポインタページ) に保存する方式[6] を採った (図2)．

論理ページ番号は，データページかポインタページかを識別するフラグ (1 bit)，データページの場合は PE 識別子 (11 bit) と物理ページ番号 (20 bit) からなり，ポインタページは物理ページ番号 (20 bit) とそのページ内のオフセット (11 bit) から構成される (図2，図3参照)．

3.2 初期 Fat-Btree の構築

ある程度のタプル数を持っているリレーションから Fat-Btree を構築する際に，どのようにして各 PE で

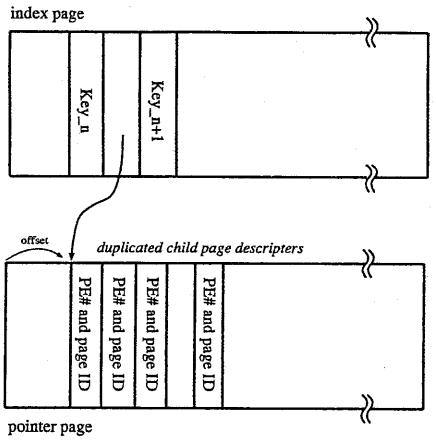

図2　ポインタページ

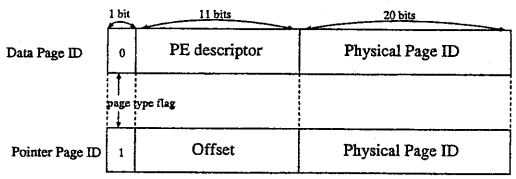

図3　論理ページ番号

異なるディレクトリ構造を構築するかという点が問題となる．

今回の実装では，SIB，CWB と Fat-Btree を比較することを考慮して，以下の手順で Fat-Btree の構築する方法を採用した．

(1) リレーションをキーでソートし，各 PE に均等に分散する
(2) ある PE でボトムアップに単一インデックスの Btree(SIB) を構築する
(3) SIB のインデックスページを，各 PE を同期させながら，ルートから左再帰で各 PE に送ることにより CWB を構築する
(4) 各 PE を同期させながら，ルートから左再帰で不要なページを削除しながら，ポインタページの情報を交換し Fat-Btree を構築する

3.3 サーバ

データを扱うために，ページの入出力を行うページサーバと，ロックをサービスするロックサーバを実装した．nCUBE3 の OS である Plan9 は，多数の共有メモリプロセス (スレッド) を I/O カーネルコールでブロックさせるプログラミングスタイルのため，ある PE 内では各サーバはマルチスレッド化されており，スレッドプール中の各スレッドがメッセージを介してサービスを処理する．各 PE ではこれらのサーバが独立して動作している．

このブロック方式のサーバに関する様々な問題点については別に節を設けて議論する．

3.3.1 ページサーバ

ページサーバは，ディスク上に存在する必要なページと，メモリ上のページキャッシュの間の読み書きを制御する．ページキャッシュのポリシーは，明示しない限りディスクへの書き込みを強制しない no-force ポリシーと，明示しない限りキャッシュからページを追い出さない no-steal ポリシーの組み合わせである．なお，キャッシュのリプレース戦略は LRU を採用している．

ページサイズはフォーマット時に設定可能であるが，実験では 4KB に固定している．

3.3.2 ロックサーバ

ロックサーバは，ページの更新を一貫的に行うために，5 種類のロックモード (IS, IX, S, SIX, X) を用意する．メッセージを介してリクエストを受けたロックサーバ中のあるスレッドは，そのロックモードを取得しにいくが，現在のロックモードと適合しない場合，指定したタイムアウト時間が来ない限り，ブロックして待つ．

ロックの適合性は，表 1 の通りである[9]．

表1　ロックマトリックス

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

3.4 ロックプロトコル

データの検索処理では，ルートページを IS モードでロックした後，次の処理を繰り返してインデックスページを順に辿っていく．

(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 ロックを取り，親ページのロックを外す

葉ページまで辿り着けば，IS ロックを S ロックにコンバートし必要なタップルを読む．

データの更新処理は，B-OPT 方式を採用し，ルートページを IX ロックして次の処理を繰り返す．

(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 IX ロックを取り，親ページのロックを外す

葉ページまで辿りつけば，IX ロックを X ロックにコンバートし更新処理を行う．もし葉ページがスプリットを起すならば，葉ページの IX ロックを外し，ルートから SIX ロックを葉ページまで取得する．さらにスプリットするページのみを X ロックにコンバートし更新処理を行う．IS，IX，S モードはローカル PE でのみ取得するだけで良いが，SIX および X ロックはそのページにコピーが存在する場合，関連するすべての PE 上のコピー対して取得する必要がある．インデックスページのコピーをロックする際には，デッドロックをできる限り回避するために，PE 番号の昇順にロックを取得する方式を採っている．しかし，この方式ではロック取得時間が非常に長くなる．

データ検索処理において，IS モードでインデックスページを辿るのは，表 1 に基づき，データ更新処理のために SIX モードを取得されているインデックスページでもキーを読めるようにするためである．

なお，今回の実装では，データ検索のロック時間を短縮する B-link[10] は用いていない．Fat-Btree では，そのページ以下に葉ページを持たなくなった時点で不要なインデックスページは消去されるため，link 先のページの存在を簡単には保証できない．これを保証するためには複雑かつオーバヘッドの大きいプロトコルが必要となる．

3.4.1 スレッドサーバの問題点

各スレッドが I/O カーネルコールでブロックするモデルを用いて実装する場合，以下の問題が生じる．

- 多くのスレッドがブロックしている PE ではスループットが低下する
- データの更新時に，リモートのインデックスページのコピーの SIX や X ロックを取得する際に，多くのリモート PE のスレッドを消費する．更新操作が多ければスレッドの枯渇が発生する
- 多くのスレッドを用意すれば，Btree の処理に対するコンテクストスイッチオーバヘッドの占める割合が増加し，全体のスループットが低下する

これらの諸問題を解決するには，

- 動的なスレッドの生成を行う．しかしスレッドの生成オーバヘッドは大きい．
- カーネルコールの直前に I/O のステータスを調査し☆，ブロックを回避する．

などが考えられる．

4. Fat-Btree の性能評価

Fat-Btree の初期評価を無共有並列計算機 nCUBE3 上で行った．今回は，データの読み出し性能のみ，SIB，CWB，Fat-Btree とを比較することにより評価した．

用いた nCUBE3 は 321PE からなり，そのうち 64PE 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．nCUBE3 の諸元は，表 2 の通りである．なお，今回の実験ではディスクを持つ PE のうち 4~16PE を使用した．

4.1 基本パラメタ

実験に用いた基本パラメタは，表 3 に示す通りである．1 ページ内のデータの占有率は，Btree の評価で通常用いられる $\log 2 = 0.693$ を用いて Btree を構築した．

実験は，キーをランダムに選び，1 万回の検索を行った際のスループットを，ページキャッシュサイズを変化させながら計測した．検索のリクエストを送るのは，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．

4.2 実験結果

図 4 は，4PE のときのスループットの比較を示したグラフである，キャッシュサイズが小さいときは Fat-Btree，CWB の順でスループットが大きい．しかしながら，キャッシュサイズが 100 ページ以上となったとき，三者のスループットの差はほとんど無くなった．これは CPU バウンドとなったからと考えられる．キャッシュが 100 ページ以下では，SIB は明らかにインデックスノードを持つ PE がボトルネックとなっている．

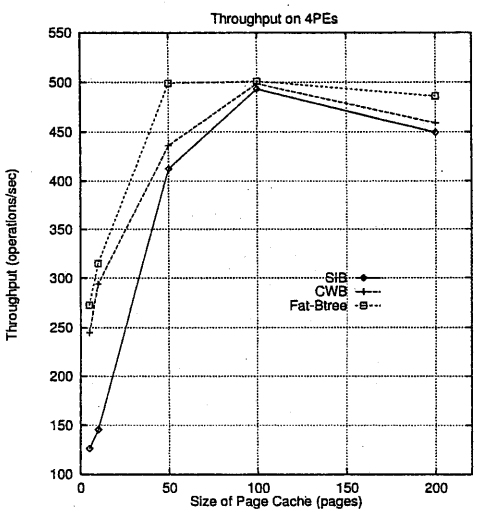

図 4　4PE 時のスループットの比較

☆ オリジナルの Plan9 には select() が存在しないが，nCUBE3 では select() が実装されている

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	スレッド当り 18.46(MB/s)
通信オーバヘッド	1 通信当り 127.2(μs)
ディスク容量	1 GB
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　基本パラメタ

項目	パラメタ
データベース	ウィスコンシンベンチマーク用 128K タップル (使用 PE 数で均等分割)
ページサイズ	4KB
インデックスページのキー数	351(最大 507)
葉ページのタップル数	13(最大 19)
木の高さ	3
葉ページ数	10084
インデックスページ数 (レベル 1)	29
インデックスページ数 (レベル 2)	1
ローカル PE でのロックオーバヘッド	11(μs)
リモート PE でのロックオーバヘッド	310(μs)
ページキャッシュサイズ	PE 当り 5〜200 ページ
キャッシュヒット時のページ読み出し時間	10(μs)
キャッシュミス時のページ読み出し時間	11.7(ms)
ページサーバスレッド数	5
ロックサーバスレッド数	5

図5 は，8PE のときのスループットの比較を示したグラフである．Fat-Btree と CWB のスループットの差は，キャッシュのヒット率の差である．例えば，キャッシュサイズが 50 ページの時，Fat-Btree のヒット率は 81.5% であるのに対して，CWB は 78.3%であった．SIB がキャッシュサイズが 50 ページのとき，最大スループットとなるのは，インデックスページを持つ PE でのキャッシュのヒット率が 93.8%と非常に高いためである．しかしキャッシュサイズが 50 ページ以上ではスループットが低下する．これはページと LRU を管理するデータ構造が，スレッド間のデッドロックを避けるために単なるリストとなっており，リストを手繰るオーバヘッドによりスループットが低下している．SIB に限らず，キャッシュサイズが大きくなったときのスループットの低下を避けるために，ページと LRU を管理するデータ構造にインデックスを張るなどの工夫が必要である．

図6 は，16PE のときのスループットの比較を示したグラフである．Fat-Btree は，グラフ中のすべてのポイントで他の方式のスループットを上回っている．16PE でデータベースを分散させた時，Fat-Btree では，インデックスを辿る際に PE を跨る確率が増加する．事実，ページキャッシュが 50 ページの時，CWB のリモートロック数が 9400 回に対して，Fat-Btree は 12000 回である．しかし，リモートロックのオーバヘッドよりもディスクのアクセスオーバヘッドの方が 2 桁大きいため，結局キャッシュのヒット率の高い Fat-Btree の方がスループットが高くなる．

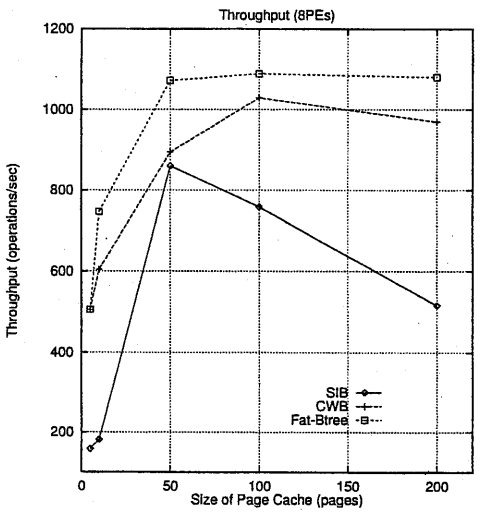

図5　8PE 時のスループットの比較

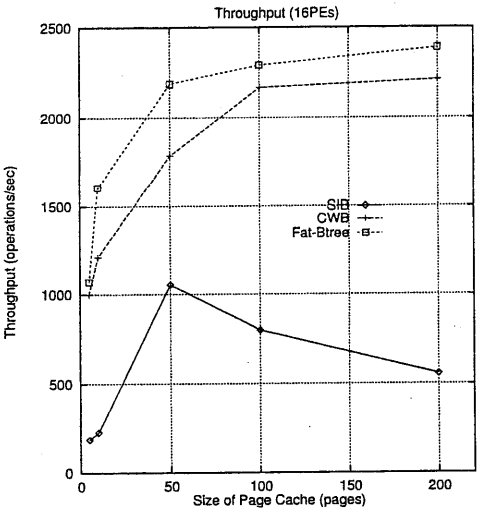

図6　16PE 時のスループットの比較

5. おわりに

我々の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 の実装とその評価について述べた．Fat-Btree は，タップルを格納する葉ページを各 PE に分散し，葉ページの直系の親となるインデックスページだけからなる Btree の部分木を各 PE は持つ．これにより，従来の SIB や CWB のような並列 Btree よりも，アクセス性能とディスクの利用率の両方の面で有利である．

本論文では，Fat-Btree を nCUBE3 上に実装することにより，ロックのオーバヘッドを含めた実際の性能の評価を行った．その結果，Fat-Btree はデータ検索性能において SIB や CWB の性能を上回ることが実証された．これは，Fat-Btree が SIB とは異なりすべての PE がインデックスを持つことによりアクセスパスのネックがない，また CWB とは異なりすべてのインデックス持たないことによりディスクキャッシュのヒット率が高いことに起因する．CWB において，オーバヘッドの大きいリモートロックと検索が PE 間を跨るケースは，葉ページを辿るときのみであるため，その回数は少ないが，Fat-Btree は検索の途中でも起こる．しかし，リモートロックおよび検索が PE 間を跨るオーバヘッドはディスクアクセスよりも十分にオーバヘッドが小さい．従って Fat-Btree は CWB よりも高い性能を持つ．

今回はデータ検索の評価のみを行ったが，今後，データの更新時の性能についても評価を行う予定である．データ更新では，データ検索よりもさらに Fat-Btree が有効であることが予想されている．さらに，今回の実装でページとキャッシュの LRU の管理を行うためのデータ構造が大きなオーバヘッドの原因と判明したので，このデータ構造の改良も同時に行う予定である．

謝辞　本研究の一部は，文部省科学研究費補助金重点領域研究「高度データベース」ならびに日本学術振興会未来開拓学術推進事業「未来映像音響創作と双方向臨場感通信を目的とした高品位 Audio-Visual System の研究」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, *Proc. of ACM SIGMOD Conf. '88*,, pp. 99–108 (1988).
2) Ghandeharizadeh, S. and DeWitt, D. J.: Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines, *Proc. of 16th Int'l. Conf. on VLDB* (1990).
3) DeWitt, D. 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) Seeger, B. and Larson, P.: Multi-Disk B-trees, *Proc. of ACM SIGMOD Conf. '91*, pp. 436–445 (1991).
5) 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 DE97–77 (AI97–44) (1997).
6) 金政泰彦, 宮崎純, 横田治夫: 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) Yokota, H., Kanemasa, Y. and Miyazaki, J.: Fat-Btree: An Update-Conscious Parallel Directory Structure, *Proc. of ICDE '99*, pp. 448–457 (1999).
8) Leiserson, C. E.: Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing, *IEEE Transactions on Computer*, Vol. c-34, No. 10, pp. 892–901 (1985).
9) Gray, J. and Reuter, A.: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers (1993).
10) Lehman, P. and Yao, S.: Efficient Locking for Concurrent Operations on B-trees, *ACM TODS*, Vol. 6, No. 4 (1981).