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に関する研究

金政 泰彦

北陸先端科学技術大学院大学 情報科学研究科

1998 年 2 月 13 日

キーワード： 並列データベース, ディレクトリ更新, 並列インデックス, 並列 B-tree, Fat-Btree.

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。そのため、各 PE 間で負荷を均等にすることが処理性能向上に継り、負荷を均等化する為のデータの分配方式が重要になる [1, 3]。

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などがある [2]。しかし、ハッシュ分配方式には、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、値域分配方式には、分割の基準値を静的に決定するので、データ更新によってデータ配置の偏りが生じた時に平坦化するコストが非常に大きくなるという欠点がある。

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 を利用する研究がなされている [4]。並列 B-tree を利用することによって、両方式の欠点をある程度解消できる上に、高速にアクセスすることが可能になる。しかし、従来の並列 B-tree の研究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 にディレクトリ全体をコピーして配置すると、ディレクトリ更新時に全 PE への同時アクセスが必要となり、それがシステムのスループットを大きく低下させてしまう。一方、更新を簡略化するため 1 つの PE のみに全ディレクトリを配置すると、その PE にアクセスが集中し、そこがボトルネックになる。

そこで本研究では、Fat-Btree[6] という新しい並列ディレクトリ構成方式を利用する。本研究では、この Fat-Btree の特性の検討を行なうとともに、確率を基にした解析によっ

て、B-tree のディレクトリ全体を全ての PE にコピーするという従来の方式との性能比較を行ない、Fat-Btree の方がスループット、レスポンスタイム共に優れることを示す。また本研究では、Fat-Btree を実際に実装する方法に関する検討も行ない、通常の B-tree との違いに起因して実装の際に生じる選択肢について考察する。

2 Fat-Btree

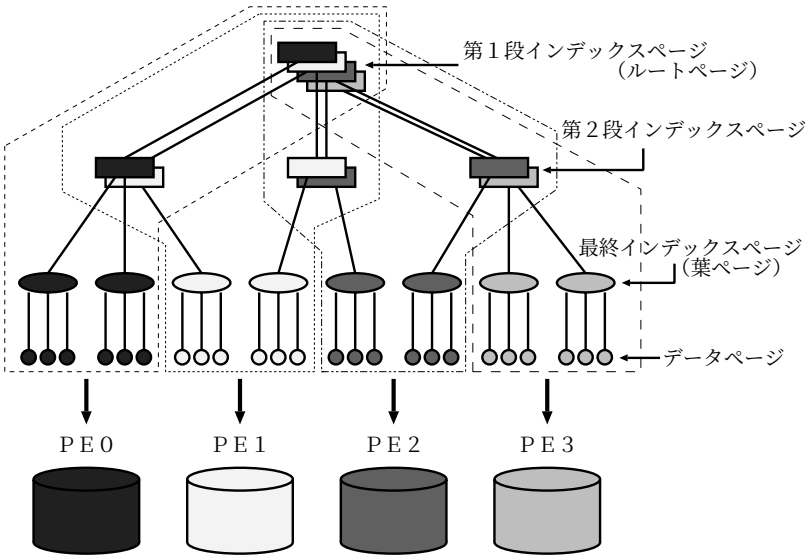

図 1: Fat-Btree 構造

Fat-Btree では、まず、B-tree の葉ページ (最終インデックスページ) を各 PE に均等に分散配置する。そして、ディレクトリ部分である B-tree の葉ページ以外のインデックスページは、各 PE に配置されている葉ページの再帰的に上位にあたるページのみをその PE に配置する。これによって、Fat-Btree で各 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となる (図 1)。

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、コピーされて多数の PE に配置され、ルートページは全ての PE にコピーされることになる。しかし、ディレクトリの更新によって書き換えられるのは、大抵の場合は比較的下位のインデックスページで、それらのインデックスページは少数の PE にしか配置されない。よって、全 PE に B-tree のディレクトリ全体を置く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 毎にデータの検索が可能で、特定の PE への処理の集中を避けることが出来る。

3　確率に基づく性能解析

Fat-Btree の有効性を検証するために確率に基づく性能解析を行ない、従来の手法である全 PE に B-tree のディレクトリ全体を置く方式 [4] (以後、B-tree 全コピー方式と呼ぶ) と比較した。解析においては、インデックスページのコピーの存在する確率やページのスプリットする確率を考慮して、READ や WRITE の 1 回の処理に必要なページアクセス数と通信メッセージセットアップ数から、1 回のオペレーションの総処理時間とレスポンスタイムを求め、それからシステム全体のスループットとレスポンスタイムを計算した。

性能解析の結果、メモリをキャッシュとして用いず、PE 数が少なく、ディレクトリの更新が殆んど無い場合を除いて、大抵の場合、Fat-Btree 方式の方がスループットが優れていることが確かめられた。特に、PE の数、タプル数が多くなり、WRITE の割合が増えるにつれて、Fat-Btree 方式は B-tree 全コピー方式よりもスループットが大幅に向上する。これは、PE の数が増加すると、B-tree 全コピー方式ではディレクトリ更新時の PE 間の同期処理に時間が非常にかかり、そのためにシステム全体のスループットが低下してしまうのだが、Fat-Btree は一部の限られた PE のみで更新処理を行なえるので、多数の PE による同期処理が少なくて済み、PE の数の増加にしたがってスループットがほぼ線形に向上するからである。

また Fat-Btree では、1 つの PE に置くディレクトリのページ数が少ないので、キャッシュメモリのヒット率が高くなり、それによって B-tree 全コピー方式よりもレスポンスタイムも向上することが分かった。

4　Fat-Btree の実装

Fat-Btree のインデックスページでは、通常の B-tree と違って子ページが他の PE、しかもコピーされて多数の PE に存在する場合があり、その場合に子ページへどのようにしてリンクを張るのかという問題がある。その解決策としては、同一 PE 内にある子ページの物理ページ番号はインデックスページ内に記載し、他の PE に置かれているコピーの物理ページ番号は別ページに保管して、インデックスページからはその物理ページ番号保管場所へのポインタだけを張っておくという方法が適している。

また、初期状態におけるリレーションからの Fat-Btree 構築の際の、各 PE 異なるディレクトリ構造の構築法については、各 PE でそれぞれ同時に同じリレーションから同じ Fat-Btree を作り、その後に各 PE で不要なページを削除するという方法が適している。

さらに、アクセスパスが PE 間に跨っている Fat-Btree におけるページロックプロトコルに関しては、READ では、Fat-Btree を B-link tree 化 [5] することによって、PE 間での問い合わせの引き継ぎの際に、子ページのロック獲得前の親ページのロック解放を行なうという方法が良く、また WRITE では、IX ロックを用いてアクセスパスをたどり、もし葉ページでの WRITE 処理に伴ってディレクトリが更新される場合には、ルートページから SIX ロックを用いてもう一度 WRITE をやり直すという楽観的な手法が適している。

5 おわりに

Fat-Btree を利用した並列ディレクトリ構成方式では、PE 数、タプル数が多くなり WRITE の割合が増えると、従来の手法である B-tree 全コピー方式よりスループットが大幅に向上することが、確率に基づく性能解析によって確かめられた。また、スループットが向上するだけでなく、レスポンスタイムも向上することが確かめられた。この性能解析の結果に関しては、電子情報通信学会データ工学研究会にて発表を行なった [7]。

本研究では更に、Fat-Btree を実際に実装する方法についての検討を行ない、インデックスページの構造、初期状態における構築法、ページロックプロトコルに関する考察を述べた。この Fat-Btree 実装に関する考察は、データ工学ワークショップ (DEWS'98) で発表する予定である。

今後の研究課題としては、Fat-Btree を実際に実装し、ロックの待ち時間や実際のページコピー数を考慮した、より詳細な性能解析を行なう必要があると考えられる。

参考文献

[1] G. Copeland and W. Alexander and E. Boughter and T Keller, "Data Placement In Bubba", Proc. of ACM SIGMOD Conf., pp.99–108 (1988).

[2] D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems", CACM, Vol. 35, No. 6, pp.85–98 (1992).

[3] S. Ghandeharizadeh and D. J. DeWitt, "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines", Proc. of 16th VLDB Conf., pp.481–492 (1990).

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

[5] V. Srinivasan and M. J. Carey, "Performance of B-Tree Concurrency Control Algorithms", Proc. of ACM SIGMOD Conf., pp.416–425 (1991).

[6] 横田 治夫, 宮崎 純, 土屋 由美子, "並列アクティブデータベースエンジン Parade の並列処理", 平成 8 年度科学研究費重点領域研究「高度データベース」松江ワークショップ講演論文集, pp426–433 (1996).

[7] 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), 電子情報通信学会 (1997).