データベースシステム　124－20
情　報　学　基　礎　62－20
（２００１．５．２２）

B木ファイルのブロック整列

渡辺　聡　三浦　孝夫
法政大学　工学研究科　電気工学専攻

本研究では，二次記憶領域において，B木ファイルへの順次探索を改善するブロック整列技術を提案する．順次探索において伝統的なB木技術，特に再構成技術が我々の助けとならないことを明らかにし，ブロック整列技術の有効性を示す．基本的なアイデアはブロックの並び換えである．この整列技術ついて幾つかの実験を行い，その結果を基にブロック整列技術について考察する．

Reordering B-tree Files

Satoshi WATANABE　Takao MIURA
Dept.of Elect. and Elect. Engr.,HOSEI University

In this investigation, we address a *reordering* technique that improves sequential processing to B-tree files dramatically on the secondary storage. We show why conventional B-tree technique doesn't help us to process sequential queries. Especially conventional reorganization is not really helpful. Our basic idea comes from reordering of blocks. In the first stage we obtain all the data in a logical order, and in the second stage we put them into pre-order. We discuss some experimental results that show how this technique works well.

1　目的

我々は，探索に関する基本的な問題に直面することがある．ここで取り扱う探索とは，**乱順探索**と**順次探索**であり，$S = \{v_1, v_2, ..., v_n\}(S \subseteq U, U$ は重複を認めないデータ集合である．$)$ とすると，乱順探索は，$x \in U$ である x が S の要素であるかを走査することであり，順次探索は $y \in S$ に対して，$x > y$ を満たす $x \in S$ の最小値を探索することである．順次探索の場合，**論理順に整列された**ファイルでは，x は単純に y の次であるので簡単に思われる．従来のどの教科書も，B木技術はどちらの探索に対しても適していると言っている (参考文献 [6])．

探索に関する問題は，一般的なデータ処理に対してとても重要である．例えば，巨大な二次記憶を搭載したコンピュータは，高速なCPUを使用することにより高性能を実現しているが，低速な入出力装置の使用などにより，性能の低下に直面することがある．そこで，データ量から影響を受けにくい探索方法が必要となる．1つの解決策として**ハッシュ技術**があるが，順次探索ができないという問題がある．ここではB木処理を考える．

一般に，情報が多ければ多いほど高い信頼性が得られるが，より大きなファイルはより多くの処理時間を要する．そこで我々は，動的な特性を保ちつつ，高い探索効率が必要となる ([4])．探索効率は木の高さ，ブロックのばらつき，バッファリングにより影響を受けると考えられる．一般に，再構成技術はその問題を改善することができる．

本研究では，二次記憶領域における順次探索を改善する**ブロック整列技術**を説明する．我々の基本的なアイデアはブロックの並び換えである．初めにすべてのデータを**論理順**に走査し，次にブロックを**読み込み順**に置き換える．ブロック整列技術は再構成とは違い，木構造をまったく変えないので，乱順探索に影響を与えない点に注意してほしい．幾つかの実験結果から，この技術が問題改善に有効であることが理解してもらえるだろう．

以下，2節ではB木システムの概要と，順次探索に関する再構成の疑問点について述べ，3節では，ブロック整列技術の概要を例を用いて説明する．また，4節では幾つかの実験結果を示し，ブロック整列技術について考察する．

2 B木ファイル

2.1 B木ファイルの動的特性

初めにB木というものを定義し，その特性を簡単に説明する．まず，重複要素を含まない集合Sを考え，それに対し**探索**，**挿入**，**削除**，**最小探索**を定義する．B木:Tは，ノードとそれらの間のポインタで構成された木構造として定義される．それぞれのノード内部は，m個の値$v_1 < ... < v_m$と$m+1$個のポインタ$p_0, .., p_m$で構成される．ここで$d \leq m \leq 2d$であり，dは位数と呼ばれる．どのポインタp_iも高さ$h-1$の部分木T_iを示す．ここでT_iのノードのどの値vも，$v_{i-1} < v < v_i$を満たす．さらに重要なことは，Tはバランスを保つということである．もし$m = 2d$ならば，Tは**完全である**と呼ばれ，木が完全であるならば，明らかに$h = \log_{2d+1} n$である．

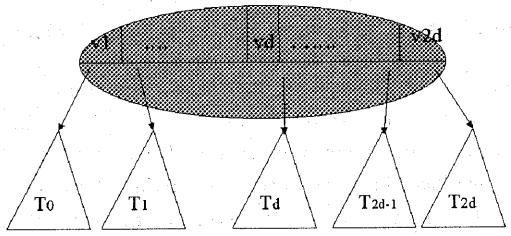

図 1: B木内のノード内部

与えられたSで生成されるB木は1つではない．実際mを変えることにより，幾つかの高さhのB木:Tを考えることができる．ここで$\log_{2d+1} n \leq h \leq \log_{d+1} n$であり，$n$は$T$のデータ数である．例として，$S = \{2, 5, 7, 12, 16\}$は，以下の異なる2つのB木を考えることができる．

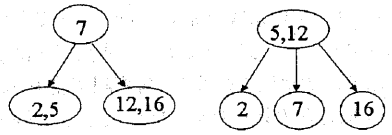

図 2: 同一要素での異なるB木

B木の更新をする際に，スプリットが起きる可能性がある．これにより，ブロック内のデータの半数が新たなブロックに移動する．これらの考えにより，ある一定のブロック利用率を保ちながら，挿入を効率良く行うことができる．

しかし，スプリット技術は，B木に対して幾つかの問題を引き起こす．その1つが，隣接ブロックのばらつきである．これによりI/O効率が低下する可能性がある．また，ブロックの空き領域が50 %になるので，最悪の場合，ブロックの半分が未使用になる．

2.2 B木ファイルの改善と再構成

より深くB木の特性を考える．乱順探索において木の高さは，探索効率に深く影響する．そこでバッファ技術を導入する．探索の際に，上位レベルのブロックは頻繁に読み込まれるので，バッファを適用することにより探索効率の向上が期待できる．一般に，バッファ数は木の高さより多く設定されているはずである．しかし，オペレーティングシステム (OS) とB木システムとの間に不釣合いが生じる．例えば，UNIXはシステム固有のバッファを持っている．この理由で利用者は，直接I/Oを制御することができず，スプリットによりさらに状況が悪化する．新たにブロックが生成される場合，OSはディスク上の空いている領域を探し，挿入する．結局，ブロックはディスク上に分散することになり，我々が順次探索を行う際に，効率的なI/O制御，及びバッファの利用が困難となる．

これに対して幾つかの解決策が提案されている．バッファ数の増加，ファイル分割などが考えられるが，これらはデータ量との問題がある．事実，莫大なデータ量に対して，それらを適用するのは難しい．そこで，より的確な解決策として，ブロック連結 (*Block Sequential*) ポインタがある．ポインタにより，上位レベルのブロックへの移動を減らすため，すべての同一レベルのブロックを連結する．しかし，これではブロックの分散には対応できない．論理順序が物理順序と一致しないためである．

これらの欠点を補うために，B木ファイルの再構成が考えられる．これにより木のレベルを低くし，効率的なブロック利用を行うことができる．例えば再構成を行う際に，すべてのノード ($d \leq m \leq 2d$) に対する枝の数を増やすことが考えられる．しかし，木の高さを低くするために，位数 d を大きくすることは，ブロックの利用効率の悪化を招く可能性があり，それがいつも適当であるとは言えない．さらに重要なことは，再構成をおこなってもスプリットが起きることである．我々は，直接スプリットを制御することはできず，ディスクの空き領域はOSにより制御されるため，新たに作られたブロックはいつも連続して割り当てられるとは限らない．このような理由から順次探索において，再構成がB木ファイルの欠点を補うために有効であるとは思えない．

次の節では，順次探索に特化した**再整列**という新しい技術を提案する．この技術は，乱順探索においては状態を維持しながら，順次探索の効率を向上することができる．

3 B木ファイルの再整列技術

3.1 B木ファイルの物理順序

この節では，B木ファイルの改善のための我々の考えを概説し，そのアルゴリズムについて考察する．よく知られているように，B木ファイルの処理は順次探索に基づいている．ここで順序 (我々の場合，B木の順次処理の観点から論理順序) について考えなければならない．すべての探索は索引を通して実行されている．我々がB木ファイルを順次あるいは乱順探索する際は，論理順序ではなく読み込み順序で木を探索していく．順次探索も乱順探索も索引を利用するので，上位レベルの索引であるほど長くバッファ内に滞在することになる．OS側のI/O制御は，隣接ブロックを一度に読み込むのが一般的であり，前方への読み込みは簡単に行うことができる．

我々の考えは，物理順序を論理順序に並び換えるところからきているので，もしブロックが読み込み順序を保ち，索引のレベルに対して十分なバッファが用意されていれば，順次探索はバッファを通して効率的に行えるはずである．

例 1 $S = \{A, B, C, D, E, F, G\}$ を仮定し，ディスク上で読み込み順 D,B,A,C,F,E,G で探索することを考える．このとき，物理順序と読み込み順序が一致した状態とは次図のような状態のことである．□

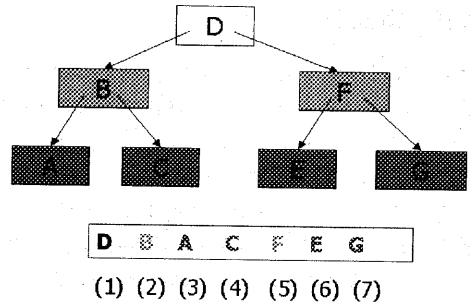

図 3: 読み込み順における物理位置の一致

読み込み順序と物理順序が異なる理由は2つある．1つは，スプリットによりブロックが分散するためである．もう1つはI/OがOSに依存するためである．UNIXのスーパーブロックやiノードのように，記憶装置へのアクセスはOSが固有に割り当てる．そのとき，読み込み順での順次処理は，ブロック分散によりかなり影響を受ける．

例 2 データが E,D,F,C,A,G,B の順に保存されていると仮定した場合，順次探索を行う際に 2,7,5,4,3,1,6 の順に探索するはずである．□

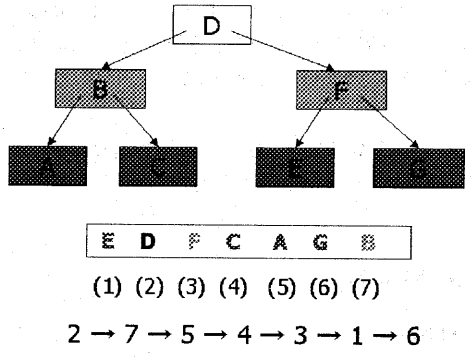

図 4: 読み込み順における物理位置の不一致

3.2 ブロック再整列

我々の目的は，どのようにして読み込み順にブロックを並び換えるかである．より具体的に説明する．どのブロックも以下のような構造を持つので，すべての木構造を再構成することができる．

Ptr_0	V_1	Ptr_1	...	Ptr_{m-1}	V_m	Ptr_m

ここで注目してもらいたいのは，どのブロックもただ1つのブロックからポイントされており，B_1, B_2 2つのブロックを仮定した場合，このブロックを置換するために，我々は2つのポインタのみ変更し，ブロックの内容は変えないということである．例えば，32番目のブロック B_2 の物理位置が，10番目のブロック B_1 の読み込み位置だった場合を考える．10番目のブロックには B_1，32番目のブロックには B_2 が入っているので，お互いの内容を移し変える．次に，32番目のブロックを指すポインタを10番目に，10番目のブロックを指すポインタを32番目に書き換える．2つのポインタはそれぞれの親ブロックからのポインタである．

このブロック置換の考えを基に，我々のアルゴリズムを例を用いて説明する．

例 3 ここでの目的は，物理位置 2 のブロック (D) を物理位置 1 のブロック (E) と置換することである．まず，ブロックを物理位置 1 に移動し，E を物理位置 2 に移動する．その際，E の親ブロック F のポインタを書き換える．

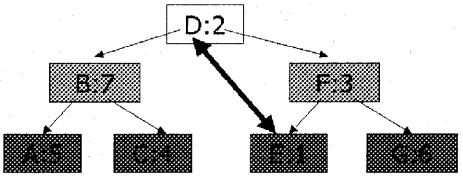

図 5: ルートブロックの置換

次に，物理位置 7 の B を物理位置 2 に移動させるために E と B と置換し，それぞれの親のポインタを書き換える．

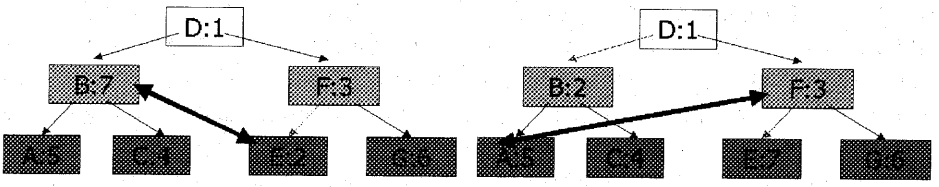

図 6: B,E と A,F の置換

次に A は物理位置 3 の F と，同様の手順で置き換える．幸運にも C と F は既に適切な位置に配置されている．最後に E と G を移動することにより並び換えが完成する．□

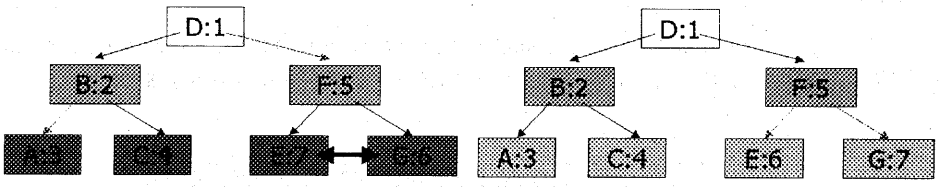

図 7: E,G の置換

3.3 再整列と再構成

再整列が再構成とどのような点で異なるのかを考える．前節で指摘したように，再構成は何を改善したのか，実際におこなってみなければわからない．再構成後，現状よりも木の高さが低くなるのか疑問であり，ブロック分散も減らない可能性がある．一方，再整列の場合は，高さやブロックの利用効率などの状態を維持しつつ，隣接ブロックを改善するので，順次探索について確実に性能が改善される．

4 実験とその結果

4.1 実験

この節では，幾つかの実験に対する結果を示す．我々は，B木システムを PentiumII (300MHz)，メモリ 64MB，SCSI ハードディスク[1] (4GB)，ADAPTEC 2940 Ultra SCSI interface を使用して FreeBSD 4.0 上で実験する．さらに，目的に合うようにキャッシュバッファの数を 10 に変更し，*B-Plus* パッケージ[2]を使用する．これはB+の特徴として，可変長キーを提供しているが，我々は再整列の基本的動作を解析したいので，このような特性を回避する[3]．このパッケージでは，索引処理はデータ部分と分離されており，索引部分とデータ部分で別々にファイルを持たなければならない．

B木ファイルへの再整列技術の適用を評価するため，次の手順で実験をおこなう．初めに，I/O 処理の OS 依存を避けるため，実験を始める際にハードディスクを初期化する．次に，ファイルの作成後，すべてのデータを順次探索する．その後，B木ファイルを再整列し，すべてのデータを順次探索する．

実験に用いるファイルは，再整列とデータ数との関連を調べるために 10000，100000，300000，500000，1000000 の 5 つを乱数を用いて作成する．さらに，キーとして使用するために，すべての整数を 100 バイトの文字列に変換する．どのブロックも 1024 バイトなので，位数は 5 となる．予想される索引レベルは，多くても $\log_5 10,000 = 6$，$\log_5 100,000 = 8$，$\log_5 300,000 = \log_5 500,000 = 9$，$\log_5 1,000,000 = 10$ である．

ハードディスクの初期化について，もう少し言及しておく．実験が OS 環境に深く依存することが予想される．特に，空きディスクの管理は完全に OS の役割なので，ブロックは任意に割り当てられ，保持される．そのようなディスク割り当てを避けるために，実験毎にディスクを初期化する．さらに実験をできるだけ同じ条件下で行うために，ファイルの削除などは行わず，ディスク上には必要なファイルのみ存在する．これにより我々は，ブロックが増加的に隣接して割り当てられることを期待している．

4.2 結果

以下の表で，10000/100000/300000/500000/1000000 はそれぞれのファイルを意味する．始めの 2 行は (B木/再整列) の順次探索の実行時間を．次の 2 行はそれぞれ実行時の転送回数を表す．

ElapsedTime

	10000	100000	300000	500000	1000000
B-tree	2.11	23.69	71	409.6	1407.48
reordered	1.54	17.29	20.46	26.75	68.73

Transfers

	10000	100000	300000	500000	1000000
B-tree	210	2050	6156	35893	116632
reordered	141	1502	1829	2562	6222

図 8: 実行時間と転送回数

次のグラフは，上の表をグラフ化したものである．

[1] セクタあたり 512 バイト，トラックあたり 63 セクタ (32 KB)，255 ヘッド．
[2] このパッケージは Hunter and Associates によるシェアウェアである．
[3] どのブロックも 1 キロバイト で 100 バイト のキーを持つ．

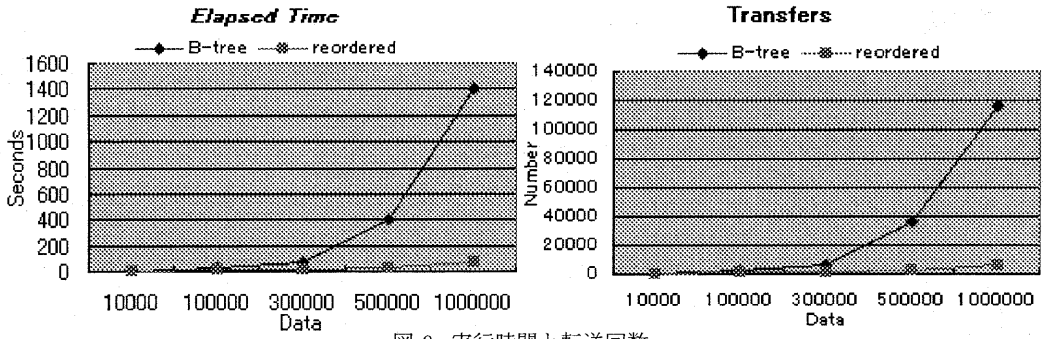

図 9: 実行時間と転送回数

より深く再整列について検証するため，利用者/システム の CPU の利用時間とデータ数との関係を次のグラフに示す．

SystemActivity

	10000	100000	300000	500000	1000000
(B-user)	0.103	0.786	2.204	3.889	8.342
(B-system)	0.031	0.651	2.423	5.649	14.494
(user)	0.062	0.724	2.175	3.492	7.86
(system)	0.066	0.478	1.396	2.27	4.126

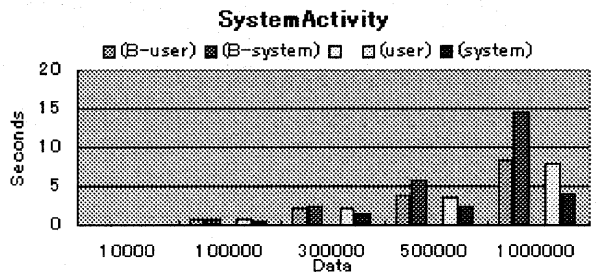

図 10: 使用者/システムの CPU 利用時間

4.3 考察

B 木と再整列のグラフを比較すると，再整列された B 木ファイルは，実行時間, 転送回数の両方で，順次探索に対して改善されているのがわかる．また，図 10 では，整列後のシステムの使用時間が減少しており，転送回数の減少などにより再整列技術が，システムへの負担を軽減したと考えられる．

さらに，次に示す改善率[4]のグラフから，実行時間と転送回数の間で類似した傾向がみられ，このことから探索時間は転送回数に依存していることがわかる．

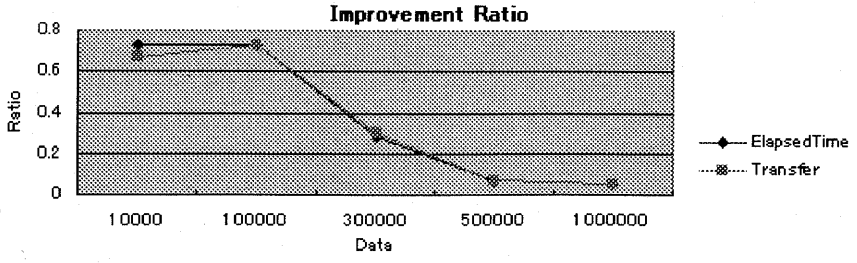

図 11: 実行時間と転送回数の改善率

[4]ここで改善率とは，整列後の値を整列前の値で割ったものと定義する．

5 結論

本論文で，ブロック整列技術とその有効性について述べてきた．この技術が通常のB木ファイルと比較して，順次探索に対してかなりの性能向上を得ることができる．したがって，我々はこの技術が順次探索における，性能向上手法の1つだと考えている．最後に，我々の取り組みは，木の高さもしくは枝の数などの状態を維持しつつ，順次探索のみ性能を向上させることを思い出してほしい．

参考文献

[1] Ing-Ray Chen: A Degradable Blink-Tree with Periodic Data Reorganization, The Computer Journal 38(3), pp.245-252 (1995)

[2] Ehud Gudes, Shalom Tsur: Experiments with B-Tree Reorganization, SIGMOD Conference 1980, pp.200-206

[3] Harbron, T.R.: File Systems - Structures and Algorithms, Prentice-Hall (1988)

[4] T.Miura, I.Shioya, W.Matsumoto and Y.Wada: Extensible Perfect Hashing, Conference on Information and Knowledge Management (CIKM) 2000

[5] June S. Park, V. Sridhar: Probabilistic Model and Optimal Reorganization of B+ Tree with Physical Clustering. TKDE 9(5), pp.826-832 (1997)

[6] Sedgewick, R.: Algorithms (2nd), Addison-Wesley (1988)