DEWS2003 5-B-03

ページングB木の設計と開発

小倉　匠吾† 　三浦　孝夫† 　塩谷　勇††

† 法政大学 工学研究科 電気工学専攻　〒184–8584 東京都小金井市梶野町 3-7-2
†† 産能大学 経営情報学部　〒259–1197 神奈川県伊勢原市上粕屋 1573
E-mail: †{i01r3211,miurat}@k.hosei.ac.jp, ††shioya@mi.sanno.ac.jp

あらまし　我々は，分散データを効率よくバランスすることのできる並列処理指向ページングB木を提案している．ページングB木を用いた分散化では，B木を部分木から成るいくつかのページに分割する．論理的なB木と物理的なB木の対応とることで，物理的なB木を論理的なB木から独立して管理する．これにより，分散データのバランスをとるためにページを任意のプロセッサーに移動させることができる．本研究では，並列処理の際に問題となる施錠について論じ，並列処理の性能を実験により評価した．
キーワード　並列, B木, ページング

A design and development of a paging B-tree

Shogo OGURA†, Takao MIURA†, and Isamu SHIOYA††

† Dept.of Elect.& Elect. Engr., HOSEI University　3-7-2, KajinoCho, Koganei, Tokyo, 184–8584 Japan
†† Department of Management and Information Science, SANNO University　1573, Kamikasuya, Isehara city, Kanagawa 259–1197 Japan
E-mail: †{i01r3211,miurat}@k.hosei.ac.jp, ††shioya@mi.sanno.ac.jp

Abstract　We propose Paging B-tree for the purpose of parallel database processing and well-balancing of data distribution. By the technique we can divide a B-tree into several pages which contain the sub-trees. We give the relationship between the logical and the physical trees. Physical B-trees are managed independent of logical ones so that we can move them into any processors to obtain well-balance of data distribution. In this work, we consider the lock which poses a problem in the case of parallel processing. And we estimated the performance of parallel processing by experiment.
Key words　Parallel, B-tree, Paging

1. 前　書　き

並列環境上では，データを均等に分散することが全体の処理性能の向上につながるため，データの偏りを解決する問題は，広く処理の最適化ととらえることができる．

従来の研究ではデータによって分割する方法として，値域を指定して分割する方法とハッシュ関数を用いて分割する方法[1]が知られている．しかし，値域を指定して分割する方法はデータの分布によって各計算機間でデータ数の偏りが生じる場合がある．ハッシュ関数を用いて分割する方法では，データ数の偏りは少なくなるが，領域質問ができなくなってしまう．これらの欠点を解決する方法として，B木のノード単位として分割する方法が提案されている．これは分割によって新たなノードが生まれたときに配置する計算機を決定する．決定方法としては，Random, Round Robin, Local Balancing[1]などの方法がある．しかし，いずれの方法もデータが削除された場合には，分散データのバランスは保証されない．

また，各計算機にB木の根から葉までの部分木を配置するFatBtree[3][4]という方法も提案されている．これは根に近いほど多くの計算機がコピーを持ち，葉に近いほどコピーを持たない．データの更新は大抵の場合，葉の近くが書き換えられることから更新処理が少なくなることが期待されるが，根ノードで更新が起こった場合には全ての計算機においてディスクアクセスを必要とする同期オーバーヘッドが生じてしまう．

我々は新たな分散B木構造としてページングB木を提案している．既に我々は部分木自体をB木構造を用いたページの対応付けに関する性能評価[7]及び，キャッシュに関する性能評価[8]において，その有用性を示している．

本研究では，並列処理の際に問題となる施錠について論じ，ソケット通信を用いたLAN環境でのページングB木を実装し，

実験により並列処理の性能を評価する．

2 章では部分木ページング B 木の説明とその処理の流れを述べる．3 章でページング B 木におけるキャッシュの活用について論じ，4 章では並列環境でのページング B 木の設計について論じる．5 章では実験結果を示し，6 章で結びとする．

2. ページング B 木

2.1 データ表現

ページング B 木では，B 木を部分木単位に分割し並列処理を行う．この方法はデータの重複が起きないのでデータの同期の必要がなく，領域質問にも対応している．また，論理的な B 木を物理的な B 木によって仮想化することで，任意に物理位置を変換可能となり，データの偏りの問題に対応している．

ページング B 木では特定した深さの部分木に分割する．分割した部分木は各々独立して管理する．あるノードの次にアクセスされるのはその下層のノードとなるので，それらを一つのページとして処理する．例えば深さ 6 の B 木を深さ 2 の部分木に分割すると図 1 のように 3 レベルに分割される．

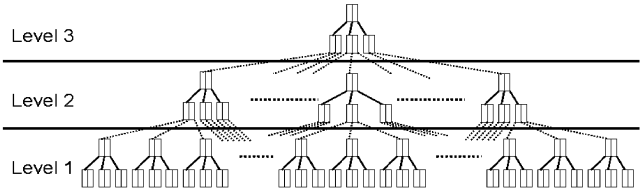
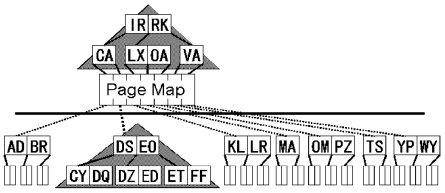


図 1　部分木分散

ページング B 木は部分木の物理キーと論理キーの対応をとることで，B 木の仮想化をしている．これによりページの物理位置を変更しても物理キーが変更されるだけで論理的な B 木構造を維持することができる．ここでは論理キーを部分木の根ノードの先頭の値（ルートキー）と部分木の配置されている高さ (レベル) の組み合わせとし，物理キーをホスト名とホスト内位置の組合せとする．各部分木には図 2 のようなページマップが用意されている．ページマップは下層部分木の物理キーのリストである．部分木の葉からのポインタは対応するページマップ項目を指している．図 2 において”DS”を論理キーとする下層部分木の物理キーは，上層部分木において論理キー”DS”を検索することで上層部分木のページマップ内から得ることができる．このようにしてキーの対応を取ることができる．



Page Map

	ホスト名	ホスト内位置
0	Host A	3
1	Host C	1
2	Host A	2
3	Host B	3
:	:	:

図 2　ページマップ

2.2 データ検索

ページング B 木における検索の手順を説明する．例えば図 2 で値”DZ”を検索するには，まずヘッダ情報として保持している根部分木の物理キーより根部分木を読み込む．そして根部分木で”DZ”を検索すると，ノード [CA] の 2 つ目のポインタに辿り着きページマップ内の物理キーを得る．最後に得られた物理キーの下層部分木を読み込み，”DZ”を検索すると値”DZ”が発見できる．

2.3 データ挿入

図 2 の状態に”EA”を挿入した場合を例として挿入方式を述べる．このページング B 木の位数は 1 とする．まず”EA”の挿入場所を検索する．挿入場所は上述の例で検索した”DZ”と隣の”ED”の間である．挿入を行うとこのノードはオーバーフローして分割が起こり，”EA”が部分木の根ノードに挿入される．根ノードでもオーバフローが起こり分割が起こる．部分木の根ノードが分割する場合には，部分木が分割を起こす．根ノードの分割で新しく生まれたノード [EO] が根ノードとなる部分木が生まれる．そして”EA”が上層の部分木に送られる．上層では新しい部分木の物理キーがページマップに挿入され，部分木では”EA”を挿入する．その際に新しいページマップ項目もポインタとして挿入される．挿入後のページング B 木の状態を図 3 に示す．

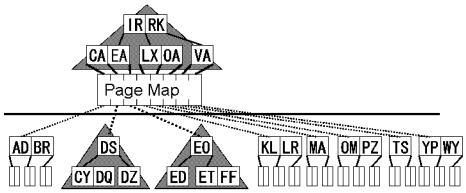


図 3　部分木分割後

2.4 部分木配置

ここでは部分木を配置する際の通信コストと分散データのバランスについて論じる．まずデータの更新により新しく生成される部分木の配置について述べる．分割により新しい部分木が生まれた時には，B 木の処理が行われているホストに配置する．これは任意に部分木配置を変更できるので，データの転送の必要ないホストに配置するのが最も効率が良いからである．またホスト数 n のページング B 木にデータを一括挿入する際に，データを 1/n づつ各ホストから挿入すると，各ホストでの分割発生率が均一になるので，部分木がバランスされて配置される．その後も任意のタイミングで部分木を移動させることが可能である．ページング B 木で全ての部分木が自由に任意のホストに移動することが可能なのは，キーの対応を取ることで論理的な B 木構造を保持しているからである．データを移動させることで，分散データのバランスさせることが可能である．

またデータを任意のホストに移動可能ということは，データ順序 (辞書式順序) 以外の物理配置も可能になる．例えばキーが住所になっていて，”トウキョウ*”の多い部分木と”カナガワ*”の多い部分木を近くに配置したり，”トウキョウ*”の多い部分木と”トクシマ*”の多い部分木を遠くに配置するなどの地理的順序の配置ができる．

3. ページングＢ木におけるキャッシュ管理

ページング B 木はでページの処理の結果によって次にアクセスするページが決定するので，並列処理で効果的な先読み（Read Ahead）やパイプライン化には不向きである．そこで読み込んだページを保存して再利用するキャッシュにより効率向上を狙う．キャッシュの単位は分割された部分木にページマップを加えたものとする．

次にページ置き換えアルゴリズムについて考える．B 木は根に近いほどアクセス頻度が高く葉に近いほどアクセス頻度が低いが，領域質問により連続データが参照される．また，一時的にアクセスが集中することも予測される．これらのことから，参照される頻度が最も低いページを書き出す LFU (Least Frequently Used) 方式や一番古くからあるページを書き出す FIFO（First In First Out）方式ではキャッシュヒット率が悪化する．最後に参照されてからの経過時間が最も長いものを書き出す LRU（Least Recently Used）方式が有効であると考えられる．

根部分木には問い合わせに対し必ず最初にアクセスが必要であり，極端にアクセスが集中してしまう．この問題も LRU 方式のキャッシュにより解決される．LRU 方式でキャッシュすると，頻繁にアクセスされる根部分木は非常に高い確率でキャッシュに保持され，実データへのアクセスは無くなる．

4. ページングＢ木処理の設計

ここでは，LAN 結合の計算機におけるソケット通信を用いた並列環境でのページング B 木を設計する．各ホストは CPU を 1 台，ディスクを 1 台持つ計算機とする．

4.1 タスク分散

まず，各ホストに B 木の処理を担当する処理タスク（B-tree）とディスク I/O を担当する I/O タスク（File）を一つずつ割り当て，根ページの格納位置情報を保持したり全体の管理をするマスタータスク（Master）をシステム全体で一つ，任意のホストに割り当てる．そして，処理タスクに対して各命令を要求する質問タスク（Query）がユーザーインターフェイスを司る．また，マスタータスクのユーザーインターフェイスとして管理タスク（Admin）がある．例えばホスト数が 3 の時には図 4 のようにタスクを割り当てる．

管理者は管理タスクを用いて，ページング B 木で扱う論理的な B 木の名前を指定して開いたり閉じたりすることができる．それらの操作はマスタータスクを通じて，全ての処理タスクと I/O タスクに伝えられる．検索や挿入などの命令は質問タスクから入力される．要求された命令は質問タスクと同じホストの処理タスクに伝えられ処理される．処理タスクは，まずマスタータスクに根ページの格納位置を問い合わせる．そして根ページの格納されているホストの I/O タスクに要求する．I/O タスクは要求されたページをその処理タスクに送り返す．処理タスクはそのページを処理し，ページマップから子ページの格納位置を取得し I/O タスクに要求する．処理が完了するまで繰り返され，最後に処理タスクから質問タスクに答えが返される．

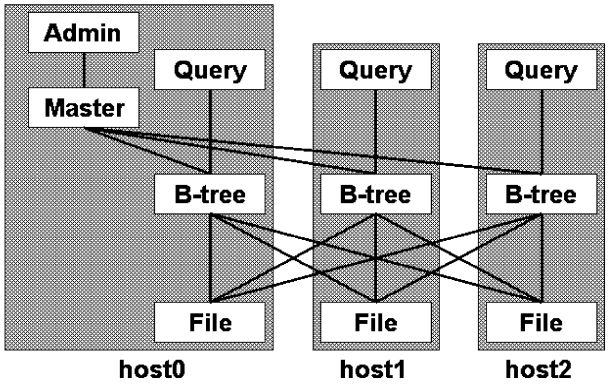


図 4　ページング B 木のタスク

また処理タスクはキャッシュバッファを持ち，読み込んだページは LRU 方式で保持され，必要なときには I/O タスクに要求せずに再利用される．

各タスクの処理自体は要求を一つずつ実行する逐次処理となるが，処理タスクと I/O タスクを分けることにより，ローカルホストでの処理に待たされることなく，各ホストからの I/O 要求に答えることができる．

4.2 施 錠 管 理

ページに対し書き込みを行う時に，複数の更新処理が同一ページに行われると不整合が生じてしまう．このような不整合が生じないために同時実行制御を行う必要がある．ここでは同時実行制御の手法として最も一般的な施錠を用いる．

ページング B 木ではコピーを持たないため，B 木に用いられる施錠手法が利用できる．そこで施錠オーバーヘッドを縮小するとして知られている B 木索引の階層構造を利用した施錠手法 [2] を応用する．

検索では根から目的の葉までのノードを順に読み込み，この時に共有施錠する．検索では一度読み込めば 2 度とアクセスする必要が無いので，これらの施錠は直ちに解放することができる．

挿入では根から目的の葉までのノードを順に読み込み，この時に排他施錠する．検索とは違い、分割の発生により葉から根に向かって順に挿入が伝播して行く．しかしノードに空き領域が存在すれば分割が発生しないので，親ノードに挿入が伝播することはない．従ってノードに空き領域が存在する時には，根から親ノードまでの施錠を解放することができる．

これら施錠の管理を行う施錠テーブルは，I/O タスクが保持するものとする．この場合，処理タスクがページをキャッシュしていたとしても，施錠状態を問い合わせる通信が必要になる．しかし，もし処理タスクが保持した時には，施錠テーブルの同期等に必要となる通信コストが膨大なものとなるので，I/O タスクが保持する方が効率的であると考えられる．

4.3 キャッシュ同期

施錠により実データの不整合は回避されるが，各処理タスクにキャッシュされているデータには適応されない．キャッシュの同期を取り，キャッシュと実データの整合性を保つ必要がある．

キャッシュ同期には，実データ更新時に同期をとる方法と読み込み時にマッチングする方法が考えられる．更新時に同期を取る方法は，多くの通信を必要とし明からに大きなオーバヘッドが必要である．一方で，読み込み時には施錠状態を確認するために通信が必要なので，この際にページのバージョン番号によるマッチングを同時に行えば，増加する通信コストは最小限に抑えられる．このことからバージョン番号を用いたマッチングを採用することにする．

host0 と host1 が同じページにアクセスする例により，キャッシュ同期の手順を説明する．まず host0 があるページを読み込み，キャッシュする．その時のページバージョンは 1 である．直後にもう一度 host0 がそのページを読み込む時には，実データのページバージョンは 1 のままなので，読み込まずにキャッシュを利用する．次に host1 がそのページを更新すると，実データのページバージョンが 2 になる．この時の host1 のキャッシュでのページバージョンは 2 だが，host0 のキャッシュでのページバージョンは 1 のままである．その後 host0 がそのページを読み込む時には，実データのバージョンが 2 に更新されているので，新たにそのページを読み込む必要がある．

5. 性 能 評 価

5.1 実験の狙い

ここでは並列処理での性能を評価する．また各ホストで同数のデータ挿入した際にページ数がバランスして配置されていることを確認する．

5.2 実 験 手 順

今回用いる計算機とページング B 木の仕様を表 1 に示す．今回の実験では同一スペックの 4 台の計算機を 1 台のスイッチングハブに結合して実験を行う．

OS	FreeBSD4.3
CPU	Pentium Ⅲ 700MHz
メモリ	64MByte
HDD	IDE Ultra ATA-33
LAN	100Mbps

Data (Key+Value)	8Byte(int 型+int 型)
位数	5
部分木の深さ	2
ページサイズ	3408Byte
キャッシュ	100 ページ

表 1　実 験 環 境

まず 100,000 個のダブりのない乱数配列を用意し，それをホスト数 n に対して 100,000/n 個ずつ各ホストから挿入する．そして，測定ホスト以外のホストでは更新比率 0.1 の処理を繰り返し行わせている状態で，更新比率 0.1 の 100,000 件の処理にかかった時間を測定ホストで測定する．これを全てのホストで測定し，スループットを算出する．スループットとは単位時間あたりの完了要求数であり，本研究では評価項目としてスループット値を用いる．

5.3 実 験 結 果

挿入後の各ホストのページ数を表 2 に示す．いずれのホスト数でも各ホストでほぼ均一に配置されていることが確認できる．

ホスト数	host0	host1	host2	host3	合計	平均
1	1674	-	-	-	1674	1674
2	824	804	-	-	1628	814
3	528	530	549	-	1607	536
4	379	427	437	407	1650	413

表 2　ページ配置

測定結果を表 3 に示し，そこから算出したスループットを表 4 に示す．また，ホスト数 4 での各ホストのスループットのグラフを図 5 に示し，ホスト数ごとのスループットのグラフを図 6 に示す．ホスト毎のスループットをホスト数 4 で見てみると，ほぼ均一だが host0 と host1 が host2 と host3 に比べて若干高い値を示している．ホスト数毎のスループットを host0 で見るとホスト数 1 に比べてホスト数 2 で大きく下がり，ホスト数 3, ホスト数 4 と徐々に下がっている．合計のスループットでは，ホスト数 2 で若干下がっているが，ホスト数 3, ホスト数 4 と上がっている．

ホスト数	host0	host1	host2	host3
1	52.45	-	-	-
2	115.63	116.45	-	-
3	130.04	129.73	148.78	-
4	139.39	137.04	151.69	152.09

表 3　測定結果［秒］

ホスト数	host0	host1	host2	host3	合計	比率
1	1907	-	-	-	1907	100 %
2	865	859	-	-	1724	90 %
3	769	771	672	-	2212	116 %
4	717	730	659	658	2764	145 %

表 4　スループット［要求/秒］

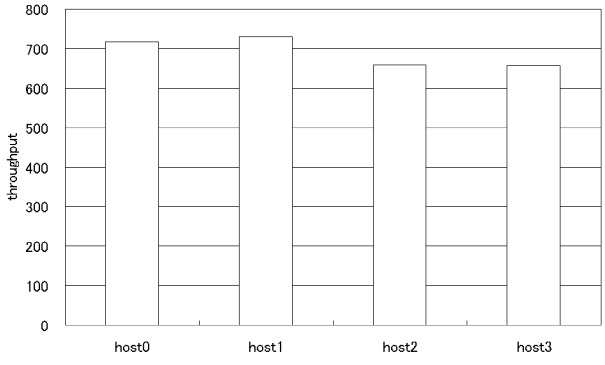


図 5　ホスト毎のスループット

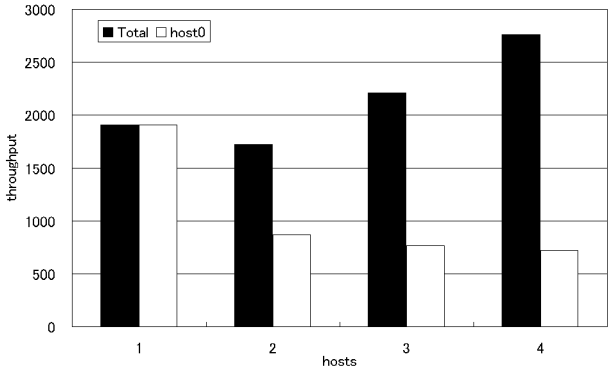


図 6　ホスト数毎のスループット

5.4 考　　察

まず各ホストのスループットだが，host0,host1 は host2,host3 に比べて通信コストが少ないために高い値となったと考えられる．その要因は host0 にはマスタータスクがあり，host1 には根ページが配置されていたことである．必ずアクセスが必要となるこれらが同一ホスト内にあるために，host2,host3 に比べて通信コストが少なかった言えるだろう．なお全てのホスト数においてマスタータスクは host0 に設定してあり，根ページはホスト数 1 では host0 に，それ以外では host1 に配置されていた．

次にホスト数毎のスループットに関する考察を行なう．host0 においてホスト数が上がるごとに値が減っているのは，通信コストが大きな原因であると考えられる．ホスト数が上がればその分ローカルホストにページがある確率が減るので，その分通信が増えている．特にホスト数 1 は単一ホストによる逐次処理であり，ホスト間通信は行われないために大きな差となったと考えられる．また，他ホストでも処理が行われているため，施錠の影響も少なくないと考えられる．合計のスループットを見てみるとホスト数 2 では若干落ちるが，ホスト数 3, ホスト数 4 と伸びている．このことから，ある程度まではホスト数ホスト数を増やせばスループットは伸びていくと考えられ，ページング B 木の有用性を確認できた．

6. 結　　び

本研究ではページング B 木の並列処理の設計し，実装を行った．そして並列処理の性能を実験により評価し，その有用性を示した．

また，本研究で設計したページング B 木は我々の研究室のホームページによりフリーウェアとして公開予定である．(http://www.dbl.k.hosei.ac.jp)

謝　　辞

本研究の一部は文部科学省科学研究費補助金 (課題番号 14580392) の支援による．

文　　献

[1] B.Seeger and P.Larson: ”Multi-Disk B-tree”, proc. ACM SIGMOD Conference 1991, pp.436-445
[2] Ramakrishnan, R. and Gehrke, J.: Database Management Systems (2nd ed) 2000, McGrawHill
[3] H.Yokota, Y.Kanemasa, and J.Miyazaki: ”Fat-Btree: An Update-Conscious Parallel Directory Structure” , proc. IEEE ICDE 1999, pp.448-457
[4] 宮崎 純, 横田 治夫: ”並列ディレクトリ構造 Fat-Btree の高信頼性構成とリカバリ”, 電子情報通信学会論文誌 D-1, 2002 年 9 月
[5] T.Miura, W.Matsumoto, I.Shioya, and Y.Wada: ”Extensible Perfect Hashing”, proc.ACM CIKM Conference 2000
[6] S.Watanabe, and T.Miura: ”Reordering B-tree”, proc.ACM SAC Conference 2002
[7] S.Ogura, and T.Miura: ”Paging B-Trees for Distributed Environment”, Workshop on Distributed Data and Structures WDAS 2002
[8] 小倉 匠吾, 三浦 孝夫: ”並列環境のためのページング B 木におけるキャッシュの有効活用”, DBWS 2002