1N-04

Bw木におけるマルチスレッドでの構造変更操作に関する性能評価

平野 匠真　杉浦 健人　石川 佳治　陸 可鏡
名古屋大学大学院情報学研究科

1　はじめに

ムーアの法則が限界を迎えたことにより，単一コアの性能向上は頭打ちとなっている．そこで，CPU に搭載された複数のコアを活用するマルチスレッド処理が主流となってきている．索引技術においてもその傾向は同様であり，CPU のメニーコア化に伴って，マルチスレッド処理を意識した索引構造が多く提案されている．

マルチスレッド処理においてロックによる同時実行制御はスケーラビリティが悪いため，B^+ 木を基にした Bw 木 [1] や Bz 木 [2] といったロックフリー索引が提案されている．これらの索引構造では，ノードの分割やマージなどの構造変更操作において，構造変更中のノードを観測したスレッドがそのノードの構造変更操作を後追いするという処理がある．

後追い処理はキューやスタックといった簡潔なデータ構造において採用された手続きを踏襲したものであるが，Bw 木や Bz 木といったより複雑な索引構造における後追い処理の有効性は検証されていない．また，構造変更操作は唯一のスレッドのみによって完了されるため，他のスレッドによる構造変更操作が無駄となり，結果として性能の低下に繋がると考えられる．そこで，本稿では Bw 木において後追い処理の有無による性能を比較し，後追い処理の必要性を評価する．

本稿では，まず Bw 木についての構造および操作について述べる．次に，後追い処理の概要と Bw 木における後追い処理の動作について述べる．最後に Bw 木の後追い処理の有無による性能の比較および評価について述べる．

2　Bw 木の概要

Bw 木は B^+ 木を拡張したロックフリー索引構造である．挿入・削除を始めとするレコード操作や分割などの木の構造変更操作において compare-and-swap（CAS）命令を用いることにより，ロックフリーでの更新を実現している．

Bw 木は，直感的にはロックフリーの単方向連結リストと B^+ 木を組み合わせた索引構造であり，図 1 のような構造を持つ．更新内容が記述された差分レコード（delta record）をノードの前に連結リストとして挿入し，索引に対する全ての更新を表す．差分レコードのリストが一定以上の長さを持つときノードの統合操作が行われ，統合後は B^+ 木のノードと同様のものが生成される．また，単一の CAS 命令を用いてノード間のポインタをインストールできるようメモリ内のデータ構造を設計することにより，木のロックフリー化を実現している [3].

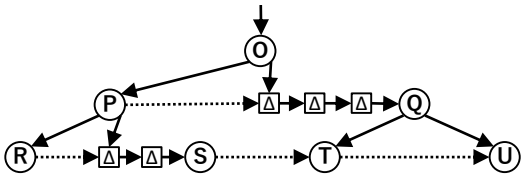

図 1　Bw 木の簡単な木構造

Bw 木は，B^+ 木と同様に，索引部およびデータ部によって構成される．索引部のノード（中間ノード）にはキーでソートされた分割キーと子ノードへのポインタの組を格納し，木の下方向に対しての検索を補助する．最下層のデータ部のノード（葉ノード）はキーと対応する値の組を格納している．なお，前述のとおり各ノードがそれぞれ差分レコードのリストを持つ点に注意する．また，全てのノードはそのノード内に格納可能な最小キーおよび最大キーを情報として持つ．各ノードは右の兄弟ノードへの単方向リンクを持つため，自身の最大キーよりも大きなキーが与えられた際は右の兄弟ノードを続けて検索できる．

以下では，Bw 木についての葉ノード操作および構造変更操作について述べる．

2.1　葉ノード操作

Bw 木は以下の読み取りおよび書き込みおよび削除をサポートする．

■読み取り　読み取り操作は与えられた対象キーを探索し，その値を返す操作である．読み取り操作には，点読み取りである read と範囲読み取りである scan がある．

■書き込み　書き込み操作（write）は葉ノードにキーと値のペアの情報を持つ差分レコードを作成し挿入する操作である．書き込み操作には，キーの不在を条件とする insert，キーの存在を前提とする update，対象キーを削除する delete がある．

2.2　構造変更操作

Bw 木はノードの統合および分割といった構造変更操作を行う．構造変更操作は差分レコードの挿入によって実現される．以下にそれぞれの操作について述べる．

■統合　統合はあるベースノードに対して，チェーン上に連なった全ての差分レコードとベースノードを統合し新たなベースノードを作成する操作である．差分レコードのチェーン数が大きくなると，検索の際にたどるリンクの数が大きくなるため検索効率の悪化を招く．そのため，差分レコードの数が閾値よりも大きくなった場合や読み取り操作時のノード内の全レコー

Performance Evaluation of Multi-Thread Structure Modification Operation in a Bw-Tree
Shoma Hirano, Kento Sugiura, Yoshiharu Ishikawa, and Kejing Lu
Graduate School of Informatics, Nagoya University

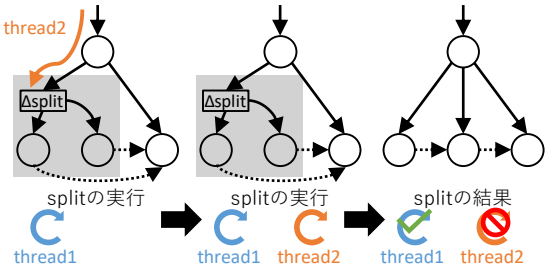

図 2　Bw 木における後追い処理

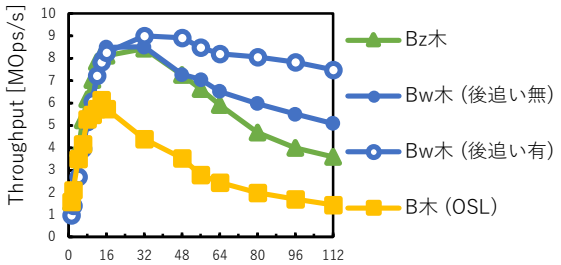

図 3　読み取りおよび書き込みの割合が等しいの場合のスレッド数ごとの各木の性能結果

ドをソートする必要がある場合に統合が行われる．スレッドが差分レコードの連結数が閾値を超えた状態や検索の際に差分レコードが挿入された状態のノードに到達した場合，CAS 命令を用いて差分レコードとベースノードを統合する．

■分割　分割はあるノードのサイズが閾値を超えた場合にノードの半分を右兄弟に割り当てる操作である．主に子ノードの分割と親ノードの更新の 2 つの段階の操作を経て分割が完了する．

■マージ　マージはあるノードサイズが閾値以下である場合に，そのノードのレコードを他のノードのレコードとつなぎ合わせる操作である．

3　後追い処理の概要

後追い処理とは，ノードの分割やマージなどの構造変更操作において，構造変更中のノードを観測したスレッドがそのノードの構造変更操作を後追いするという処理である．これは，キューやスタックといった簡潔なデータ構造において，性能向上を図るために採用された手続きである．次に，Bw 木における後追い処理について述べる．

3.1　Bw 木における後追い処理

Bw 木は 2 章で述べた通り，ノードの統合および分割といった構造変更操作を，差分レコードの挿入によって実現する．そのため，構造変更操作はその場で行われず，統合操作が行われる際に実行される．Bw 木における分割操作の後追い処理例を図 2 に示す．スレッドが処理途中の構造変更操作を観測すると，現在の処理を中断し，行われている構造変更操作を後追い実行する．最終的に唯一のスレッドが構造変更操作を完了させ，他のスレッドの構造変更操作は失敗する．

3.2　Bw 木における後追い処理の特性

前述した通り，Bw 木の後追い処理は簡潔なデータ構造において採用された手続きを踏襲したものであり，より複雑な索引構造における後追い処理の有効性は検証されていない．また，Bw 木の後追い処理では唯一のスレッドの構造変更操作のみが成功し，その他のスレッドの構造変更操作は失敗することから多数のスレッドの無駄が生じてしまう．そのため，キーの偏りの大きいワークロードにおける，多数のスレッドの同一ノードへの構造変更操作の後追い処理が行われた際には性能低下の可能性が考えられる．そこで，本研究では再現実装した Bw 木において後追い処理の有無による性能比較を行い，後追い処理の必要性を検証する．構造変更操作を観測したスレッドが後追い処理を行わない場合の挙動として，対象ノードを再探索する手法と再探索せずに待機する手法がある．再探索する場合，構造変更操作が失敗するスレッド数が少なくなる．再探索せずに待機する場合には，失敗するスレッド数が少なることに加えて，CPU のスリープによる CPU 利用率の大幅な削減が見込める．以上のように，後追い処理を行わない場合の方が後追い処理を行う場合よりも高い性能を示すと考えられる．

4　評価分析

後追い処理を行う場合と行わない場合それぞれの Bw 木を構築する．その後，それぞれの場合の Bw 木，Bz 木および楽観的ロック（OSL）B 木の性能を比較する．具体的には，読み取り操作と書き込み操作の割合の変化によるそれぞれの木の性能を評価する．読み取りと書き込みの操作割合が等しい場合のそれぞれの木の調査結果を図 3 に示す．図 3 より，後追い処理を行わない Bw 木が，それぞれの木の中で最も高い性能を示した．

5　おわりに

本稿では，ロックフリー索引である Bw 木および後追い処理の概要を述べ，後追い処理の有無による性能比較について提案した．調査の結果，後追い処理を行わない場合の Bw 木が，後追い処理を行う場合の Bw 木よりも高い性能を示した．今後は，葉ノード操作の割合やキーの偏りの割合，操作ごとの CPU 利用率など細かい指標について評価をしていく予定である．

謝辞

本研究は JSPS 科研費（16H01722，20K19804, 21H03555）の助成，および国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP16007）の結果得られたものである．

参考文献

[1] J. Levandoski, D. Lomet, and S. Sengupta, “The Bw-tree: A B-tree for new hardware platforms,” in *Proc. ICDE*, pp. 302–313, 2013.
[2] J. Arulraj, J. Levandoski, U. F. Minhas, and P. Larson, “BzTree: A high-performance latch-free range index for non-volatile memory,” *PVLDB*, vol. 11, no. 5, pp. 553–565, 2018.
[3] A. Petrov, 詳説データベース: ストレージエンジンと分散データシステムの仕組み. オライリー・ジャパン, 2021.