1N-03

Bz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

中山 宗　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

近年，Bz 木 [1] などのロックフリーアルゴリズムに基づく索引が注目されている．ムーアの法則の終焉によるメニーコア環境への転換に伴い，マルチスレッド向けの索引に対する需要が増加しているためである．近年提案された Bz 木は B^+ 木のロックフリー拡張の 1 つであり，ロックフリーな動作により高い同時実行性を達成する．

Bz 木の元論文では，マルチスレッド環境における構造変更操作に関する競合の対処法として，構造変更操作を複数スレッドで追従する手法が提案されている．しかしこの手法では，最終的に構造変更操作を反映するのは 1 スレッドのみであり，操作が反映されないスレッドの処理が無駄となり非効率的であると考えられる．

そこで本稿では，構造変更操作を複数スレッドで追従する手法と対象ノードの再探索に戻る手法の性能を実験により比較する．

2　Bz 木の概要

Bz 木は multi-word compare-and-swap（MwCAS）命令 [2] を用いて B^+ 木を拡張したロックフリーの索引構造である．MwCAS 命令は compare-and-swap（CAS）命令の拡張であり，メモリ上の複数ワードを対象に値の比較と交換をアトミックに行う．Bz 木は MwCAS 命令を用いてレコードの挿入および木の構造変更におけるマルチスレッド間の順序付けを行い，ロックフリーな動作を実現している．

2.1　構造

Bz 木は，Bz 木の根を指すルートポインタと，複数の中間ノードと葉ノードからなる．全体像を図 1 に示す．ルートポインタが根となる中間ノードへのポインタ，中間ノードが子となる中間ノードまたは葉ノードへのポインタを持つ．これにより，全体として木構造を構築する．

図 2 に示すように，各ノードは 1 つのヘッダと，複数のメタデータからなるメタデータ列，複数のレコードからなるレコード列を持つ．ヘッダには，ノードの大きさやレコード数，ノード更新時に必要な情報など，ノード自身の情報が格納される．メタデータは 1 つのレコードに対して 1 つ存在し，対応するレコードのメモリ上での位置など，対応するレコードの情報を格納する．レコードはキーとペイロードの組からなる．ペイロードは，中間ノードの場合は葉ノードへのポインタ，葉ノードの場合はデータそのもの，またはデータへのポインタである．

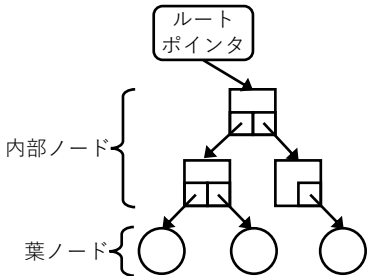

図 1　Bz 木の構造

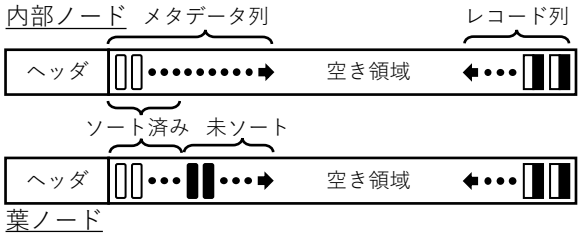

図 2　内部ノード（上）と葉ノード（下）の構造

中間ノードと異なり，葉ノードは差分レコードを持つ．差分レコードとは，対応するメタデータがソートされていないレコードである．中間ノードのメタデータ列は，中間ノードへ新たなメタデータとレコードの組を挿入する際に，対応するレコードのキーの順にソートし常に順序を保つ．葉ノードに新たなメタデータとレコードの組を挿入する際は，メタデータ列のソートは行わない．ゆえに，葉ノードはソート済みなメタデータ列だけでなく，未ソートなメタデータ列とそれらに対応する差分レコードを持つこととなる．葉ノードへ新たなレコードを挿入する際にメタデータ列をソートしないことで，レコードの書き込み操作の性能向上を図っている．

2.2　構造変更操作

Bz 木はノード再編成，ノード分割，ノードマージの 3 つの構造変更操作を持つ．ノード分割とノードマージは B^+ 木も持つ構造変更操作であるが，ノード再編成は差分レコードを葉ノードに持つ Bz 木特有の構造変更操作である．

■ノード再編成　葉ノードが持つ全てのメタデータをソートする，葉ノード特有の操作である．葉ノードが持つ差分レコードのサイズが閾値を超えた場合に行う．葉ノードが持つ未ソートなメタデータ列とソート済みなメタデータ列をマージソートすることにより実現する．ノード再編成により葉ノードが持つ差分レコードの数を制限し，レコードを読み込む操作の性能低下

Performance Evaluation of Multi-threaded Structure Modification Operations on BzTree
Shu Nakayama, Kento Sugiura, Yoshiharu Ishikawa, and Kejing Lu
Graduate School of Informatics, Nagoya University

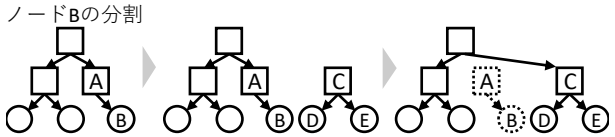

図 3　ノード分割の操作例

を防ぐ．

■**ノード分割**　保持するメタデータ列とレコード列の合計サイズが閾値を超えたノードを，2 つのノードに分割する操作である．ノード内のレコード列と対応するメタデータ列をキーの順に前半と後半で 2 等分した後，それぞれのメタデータ列とレコード列を持つ 2 つのノードを作成する．その後，分割対象ノードを削除し，新たに作成した 2 つのノードを挿入する．ノード分割を行うことで，ノードの保持するデータがノードサイズを超えることを防ぐ．

■**ノードマージ**　2 つのノードを 1 つのノードへと統合する操作である．レコードと対応するメタデータを削除する操作により，ノードが保持するデータのサイズが閾値を下回った場合に行う．データサイズが閾値を下回ったノードと，そのノードの左右に存在する兄弟ノードの 1 つを統合する．2 つのノードが保持するレコード列と対応するメタデータ列を持つノードを新しく作成した後，統合対象である 2 つのノードを削除し，新たに作成したノードを挿入する．ノードマージを行うことで，保持するレコード数が少ないノードを減らし，索引構造全体におけるレコードの探索性能の低下を防ぐ．

全ての構造変更操作による Bz 木への変更は，1 つの特定ノードへのポインタの変更により実現される．1 つのポインタとそのポインタを持つノードのヘッダの変更を MwCAS を用いて行うことで，構造変更操作をアトミックに行う．ノード再編成の場合は，差分レコードを持つ葉ノードへのポインタを，ソート済みなメタデータのみを持つ新たに作成した葉ノードへのポインタに交換する．ノード分割やノードマージのようにノード数が変化する操作の場合は，変更対象ノードではなく，変更対象ノードへのポインタを持つ親ノードへのポインタを変更する．ノード分割の操作例を図 3 に示す．ノード分割の場合は分割対象ノード（B）の親ノード（A）を複製し，複製した親ノード（C）が持つ変更対象ノードへのポインタを削除し，分割後のノードとして新たに作成した 2 つのノード（D と E）へのポインタを挿入する．その後，元の親ノードへのポインタを，複製した親ノードへのポインタに交換する．ノードマージも同様である．

3　構造変更操作における追従処理の削除

Bz 木では構造変更操作によってノードに変更を加える際，変更を加えるノードを凍結状態にすることで，マルチスレッド環境下での同時実行制御を行う．凍結状態のノードが持つレコードは，書き込み操作や削除操作によって変更されないことが保証される．

元論文の手法では，書き込み操作や削除操作の対象ノードが凍結状態であったスレッドは一旦操作を中断し，凍結状態の原因となった構造変更操作を追従する．操作対象ノードが凍結状態であった場合，まず，操作対象ノードの親ノードが他スレッドによる構造変更操作の対象となっていないか確認する．操作対象ノードの構造変更操作を追従した後に，削除する予定のノードにポインタを挿入することを防ぐためである．同様の理由から，操作対象ノードがその左右のノードに対して行われているノードマージの対象でないかも確認する．親ノードが他スレッドによる構造変更操作の対象になっていたり，操作対象ノードが左右のノードに対して行われているノードマージの対象になっていた場合は，それらの操作を追従する．操作の対象になっていない場合は，操作対象ノードが持つレコード数やソート済みレコード数から，そのノードに行われている構造変更操作を特定する．その後，操作対象ノードに対して特定した構造変更操作を開始する．ゆえに，元論文に準ずる場合，複数のスレッドが同一の構造変更操作を行うこともあり得る．複数のスレッドが同一の構造変更操作を行った場合，最も速く操作を終了したスレッドの操作のみを反映し，その他のスレッドの操作は反映されない．

前述した複数のスレッドにより同一の構造変更操作を追従する手法は非効率的であると考えられる．複数のスレッドが同一の構造変更操作を行ったとしても，1 つのスレッドによる操作のみが反映されその他のスレッドによる操作は無駄となるためである．

そこで本研究では，非効率的であると考えられる構造変更操作を複数スレッドで追従する手法と，構造変更操作の競合を検知したスレッドが対象ノードの再探索に戻る手法の性能を比較する．対象ノードの再探索に戻る手法では，操作対象であるノードが凍結状態であったスレッドは構造変更操作を追従せずに，元の書き込み操作や削除操作の対象となるノードを根から再び探索する．ゆえに，対象ノードの再探索に戻る手法では反映されない操作を行うスレッドが生じず，構造変更操作を複数スレッドで追従する手法より効率的であると予想される．

4　おわりに

本稿では，BzTree の元論文で提案された構造変更操作を追従する手法が非効率的あることを述べ，その非効率的な手法に代わる対象ノードの再探索に戻る手法を提案した．今後は，3 章で述べたように，構造変更操作を追従する手法と対象ノードの再探索に戻る手法の性能を比較する実験を行っていく．

謝辞

本研究は JSPS 科研費（JP20K19804, JP21H03555, JP22H03594）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. Arulraj, J. Levandoski, U. 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.

[2] T. L. Harris, K. Fraser, and I. A. Pratt, “A practical multi-word compare-and-swap operation,” in *Proc. DISC*, pp. 265–279, 2002.