1N-02

# Bz 木における範囲走査性能の改善

井戸佑 † 杉浦健人 †† 中山宗 †† 石川佳治 †† 陸可鏡 ††

† 名古屋大学情報学部コンピュータ科学科 †† 名古屋大学大学院情報学研究科

## 1 はじめに

近年，メニーコアなどを前提としたインメモリデータベースの研究が進んでいる．データベースの構成要素の 1 つである索引技術も同様に，メニーコア・大容量メモリに適合させる必要がある．索引操作の同時実行制御の手法としてロックフリー索引がある．

$B^+$ 木 [1] をロックフリー化させた索引として Bz 木 [2] および Bw 木 [3] が提案されている．Bz 木は葉ノード内に，ソートされたレコード領域とソートされていないレコード領域を持っている．追加レコードはソートされていないレコード領域に格納することで，追加の度にソートする手間を省いている．そのため，範囲走査の際には，1 度ノード内のレコード全てをソートする必要がある．既存の手法では，1 度ノード内のレコードのソートを行った後，範囲内のレコードを 1 つずつ返している．しかし，この手法では 1 度の範囲走査に 2 回のレコードアクセスが必要であり，効率が悪い．そこで，本稿ではソートを行いながら走査を同時に行う手法を提案する．ソートをしてから走査する既存手法とソートと同時に走査する提案手法を検証し，範囲走査に与える影響を評価する．

本稿では，まず Bz 木の構造を説明する．次に既存の範囲走査の手法および提案する範囲走査の方法を述べる．最後に，両手法の範囲走査の性能の比較および評価方法について述べる．

## 2 Bz 木の概要

Bz 木は $B^+$ 木を拡張したロックフリー索引である．木構造を図 1 に示す．Bz 木は，挿入や削除などの葉ノード操作や，統合や分割などの構造変更において multi-word compare-and-swap（MwCAS）命令 [4] を用いることでロックフリーを実現している．以下では，MwCAS 命令と Bz 木の概要を説明する．

### 2.1 MwCAS 命令

Compare-and-swap（CAS）命令はメモリの内容を更新する命令である．指定したメモリの現在の値を読み取り，読み取った値が期待する値であれば新しい値に変更する．しかし，期待する値でなければ CAS 命令は失敗する．CAS 命令は値の変更を確認でき，他スレッドとの競合を検証できる．

Bz 木は単一の CAS 命令ではなく，複数のメモリを対象とする MwCAS 命令を用いている．複数のメモリに対して現在の値を読み取り，読みとった全ての値が期待する値であれば新しい値に変更する．1 つでも期待と異なる値があれば，MwCAS 命令は失敗する．Bz 木では，葉ノード操作や構造変更を MwCAS 命令を用いて多段階に行うことによって，他スレッドとの競合を検知しロックフリーを実現している．

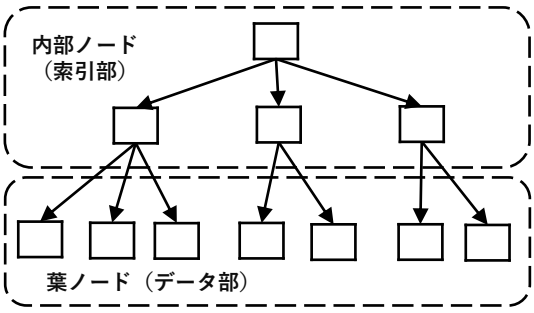


図 1 Bz 木の構造

### 2.2 Bz 木

Bz 木は $B^+$ 木の葉ノード内にロックフリーで追加できる領域を持たせた索引構造である．葉ノード間のポインタは持たず，MwCAS 命令を用いてロックフリーを実現している．なお，Bz 木は本来永続メモリ向けの索引として提案されたが，本稿では永続化については言及せずロックフリー索引としてのみ扱う．

Bz 木は $B^+$ 木同様に索引部，データ部から構成される．$B^+$ 木のノード内はヘッダ，メタデータ，空き領域，レコードの 4 領域で構成される．ヘッダはノードの先頭に位置し，ノード全体の情報を格納する．ノードサイズ，レコード数，ノードの操作可能性を示すフラグなどが格納されている．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情報を格納する．レコードはキーとレコードへのポインタまたは実際のペイロードを格納する．空き領域は差分レコード用の領域で，挿入や更新などの葉ノードへの操作を処理する．

## 3 Bz 木における範囲走査

Bz 木は範囲走査をサポートしている．以下では，範囲走査，既存手法および提案手法の概要を述べる．

### 3.1 範囲走査

範囲走査はユーザが始点キーと終点キーを指定することで，そのキーの範囲内のレコードを検索し，返す操作である．始点キー，終点キーは省略が可能であり，その場合には省略した一端がオープンエンドとなる．範囲走査は以下の 3 つの手順から成る．

まず，葉ノードを探索する．始点キーが指定された場合，そのキーを用いて走査対象の葉ノードを検索する．指定が無けれ

---

Improvement of range scan performance in BzTree

Yu Ido†, Kento Sugiura††, Shu Nakayama††, Yoshiharu Ishikawa††, and Kejing Lu††

†Department of Computer Science, School of Informatics, Nagoya University

††Graduate School of Informatics, Nagoya University

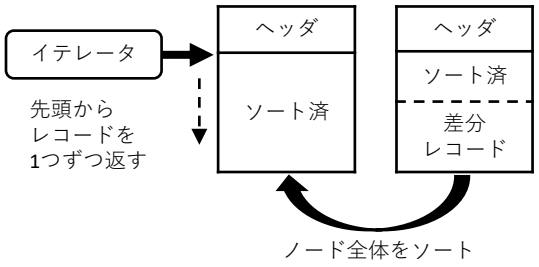


図 2　既存手法

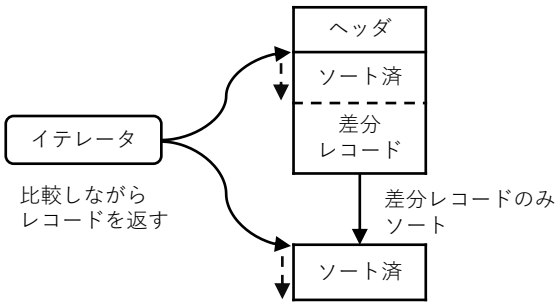


図 3　提案手法

ば索引内の左端のノードを選択する．

次に，走査用ノードを生成する．探索したノード内に差分レコードが存在すれば，差分レコードとソート済みレコードを統合し，ノード内の全てのレコードがソートされたノードの複製を作成する．同時に，削除済みのレコードやキーの重複を削除する．

最後にユーザは複製したノードをイテレータで走査する．イテレータでは，範囲内のレコードを先頭から 1 つずつ走査する．終点キーまで走査し終わる，またはユーザ側で終了すれば走査は終了する．ノード内のレコードを全て走査しても終点キーにたどり着かない場合には，1 つ右のノードへ移動し，走査用ノードの生成から再び行う．Bz 木は，葉ノード間のポインタは持たないため，1 つ右のノードは根から再検索する必要がある．

### 3.2　既存手法と提案手法

既存手法では，ノード内のソートを行ってから範囲走査する．既存手法を図 2 に示す．ノードのソート時に全レコードへアクセスした後，範囲走査時にも該当レコードにアクセスするため，1 度の走査で 2 回のレコードアクセスが必要となり，非効率である．

そこで，本稿で提案する手法は走査前のソート処理を行わず，ソートしながら走査する手法である．提案手法を図 3 に示す．ノードの選択後，ノード全体のソートは行わず，差分レコードのみソートを行う．ユーザはノードのソート済みレコード部分とソート済み差分レコードをイテレータで走査する．イテレータでは，ノードのソート済みレコードとソート済み差分レコードの先頭の値を比較し，小さい方を返す．同時に削除済みのレコードやキーの重複を削除する．以降の手順は同様である．提案手法では，データアクセスの回数が抑えられ，複製も排除できる．

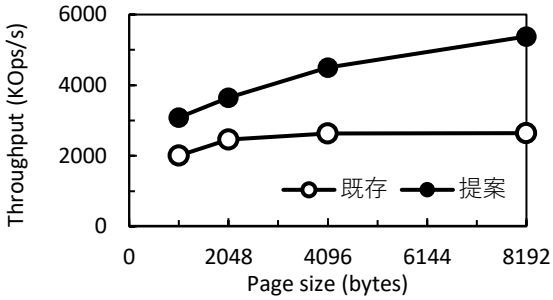


図 4　ページサイズ変化時の各手法の性能結果

## 4　評価分析

既存手法および提案手法における範囲走査の性能を調査する．各手法の範囲走査性能の比較をパラメータを変化させながら行い，提案手法でどの程度性能が改善するか定量的に測定や分析をする．具体的には，ページサイズ，Write 命令の割合，キーアクセスの偏りを変化させて，スループットを測定する．

ページサイズを変化させたときの各手法の性能結果を図 4 に示す．図 4 より，提案手法は既存手法の 1.5 倍程度の性能を出すこと，またページサイズが大きい程性能の上がり幅も大きくなることが分かった．既存手法は該当範囲のレコードに 2 回アクセスする必要があるため，キャッシュヒット率でスループットが大きく変わる．ページサイズを大きくしたことにより，キャッシュからあふれる可能性が大きくなったため，このような結果が得られた．

## 5　おわりに

本稿では，ロックフリー索引である Bz 木の概要，Bz 木における範囲走査を述べ，改善方法を提案した．今後は 4 章で述べたように既存手法と提案手法の範囲走査における性能比較のために実験していく予定である．

**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] 北川 博之, データベースシステム. 昭晃堂, 1996.
[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] J. Levandoski, D. Lomet, and S. Sengupta, “The Bw-tree: A B-tree for new hardware platforms,” in *Proc. ICDE*, pp. 302–313, 2013.
[4] T. L. Harris, K. Fraser, and I. A. Pratt, “A practical multi-word compare-and-swap operation,” in *International Symposium on Distributed Computing*, pp. 265–279, 2002.