5L-06

不揮発性メモリのための索引手法の分析

西村 学 † 杉浦 健人 †† 石川 佳治 ††

† 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

揮発性メモリに迫る高速性と耐久性を満たす不揮発性メモリの研究・開発が進められており，将来主流になると考えられている．不揮発性メモリは揮発性メモリに比べ読み書きの速度や耐久性で劣る一方，電源を切っても記憶内容を保持できることからこれまで HDD や SSD といった補助記憶装置として使用されてきた．しかし，近年においては 3D XPoint 技術を使用した高耐久かつ低レイテンシの DCPMM（Intel Optane Persistent Memory）など不揮発でありながら揮発性メモリに迫る性能を持つメモリの研究・開発が進められており，そのようなメモリを活用する手法やアーキテクチャの研究も同じく進められている．

索引構造もその一つであり，不揮発性メモリに索引を構築することで高い性能と高速な障害回復との両方を達成できるとして注目されている．しかし，B^+ 木のような既存の索引構造をそのまま不揮発性メモリに構築しても，その恩恵は受けられない．そのため，これまでの研究では不揮発性メモリの特性を最大限に活かすような索引構造の提案が多くなされてきた [1,2].

しかし，主記憶として使用できるような不揮発性メモリはまだ普及し始めたばかりであり，提案された時点では実物が不足していた．ゆえに，これらの不揮発性メモリ向けの索引構造の性能評価は，揮発性メモリ上のエミューレーションに基づくものがほとんどであり，実際の性能は未知数な部分が多い．そこで本研究では，不揮発性メモリ向けに提案された索引構造の一つである Bz 木 [3] を用いて，いくつかの実験を通してその動作や特性について議論する．

2 Bz 木の概要

Bz 木は，B^+ 木を拡張して実装される索引構造であり，挿入・削除・更新といった標準的な操作を不揮発性メモリ上でアトミックに実行する．PMwCAS（persistent multi-word compare-and-swap）[4] を用いることで既存の不揮発性メモリ向けの索引と比べて簡潔に実装を記述でき，かつラッチフリー（ロックフリー）な動作により高い同時実行性を達成する．以下ではその構造などについて概略を説明する．

2.1 構造・設計

Bz 木は，キーおよび子ノードへのポインタを格納する内部ノードと，キーおよびレコードへのポインタもしくは実際のペイロードを格納するリーフノードから構成される．内部ノードはイミュータブルであり，検索に最適化するため常にソートされたキーをもつ．一方で，リーフノードはソートされたキーとソートされていないキーをもつ．これはレコードをノードの空いている領域にそのまま格納するためであり，内部ノードと比べて書き込みに最適化されている．ただし，リーフノードに格納されているキーは性能維持のために定期的にソートされる．

ノードを構成する要素には，レコード（キーとペイロードのペア）だけではなくヘッダ，メタデータエントリ，空きスペースがある．これらはノードの先頭からこの順番で配置され，レコードを格納する部分はノードの末尾に存在する．ヘッダやメタデータエントリには，挿入や更新などの操作に必要な情報が格納されている．空きスペースはリーフノードにのみ存在し，挿入や更新をバッファするために使用される．内部ノードではバッファの必要がないため存在しない．

2.2 PMwCAS とその利用

索引構造において，ノードのマージや分割といった操作は複数ワード（ポインタなどメモリ上の 4 ないし 8 バイト領域）の更新を必要とする．通常の CAS（compare-and-swap）命令では操作対象が単一ワードに限られるため，それらの操作は複数ステップに分割され，ノードの中間状態（処理途中の状態）は他のスレッドに公開されてしまう．したがって，索引構造がラッチフリーであるためには，競合状態を回避し協調動作するためのロジックが別に必要となる．

一方で，PMwCAS は不揮発性メモリのために volatile MwCAS を拡張したものであり，一度に複数のワードをアトミックに更新できる．これにより，索引構造の更新も一つの命令として処理でき，複雑なロジックを記述することなく競合を回避したラッチフリーな動作が実現できる．

2.3 動作環境

不揮発性メモリ向けに提案された Bz 木であるが，揮発性メモリであっても構造の変更なくそのまま使用できる．揮発性メモリ上で動作させる場合，Bz 木のノードは全て揮発性 DRAM に格納され，障害時に内容は失われる．

一方で，不揮発性メモリで動作させる場合，ノードは全て不揮発性メモリに格納される．つまり，データの書き込みと共にその内容は永続化され，障害が発生したとしても Bz 木の索引構造は健常なまま保持される．

3 Bz 木の詳細

本章では，Bz 木におけるノード操作の詳細について説明する．

Analysis of Index Structure for Non-Volatile Memory

Manabu Nishimura†, Kento Sugiura††, and Yoshiharu Ishikawa††

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

††Graduate School of Informatics, Nagoya University

3.1 リーフノードに対する操作

■読み込み 読み込み操作は，リーフノードが格納しているものによって異なる動作をする．レコードへのポインタを格納している場合，スレッドは先にソートされたキー空間を二分探索し，見つからなければソートされていないキー空間を線形探索する．実際のペイロードを格納している場合，スレッドはソートされていないキー空間を最新のエントリから順に探索し，見つからなければソートされたキー空間を探索する．これは，最新のエントリほど最新のペイロードを示すためである．なお，読み込み操作ではノード上で同時に行われる更新は全て無視される．

■挿入 挿入される新しいレコードは，次の手順でノードの空きスペースに追加される．まず，2 ワードの PMwCAS を用いてヘッダとメタデータエントリに新しいレコードのサイズなど新たな情報を書き込み，そのレコード用のスペースを確保する．スペースの確保に成功したら，そこへ実際にレコードの内容をコピーし，再び 2 ワードの PMwCAS を用いてヘッダとメタデータエントリを更新することで挿入が完了する．

動作中，スレッドはそのノードが凍結状態でないかどうかを適宜ヘッダから読み取り，凍結状態であるなら動作をやめ再試行する．なお，ノードの凍結状態はそのノードが他のスレッドから操作されているかどうかを示す．つまり，ノードを凍結させたスレッドの操作は他スレッドによって妨害されることがなく，これにより並列処理時のノードの一貫性を保つ．

■削除 レコードの削除は 2 ワードの PMwCAS によって実行される．まず，レコードが削除されたことを示すようにメタデータエントリを更新する．そして，削除するレコードのサイズをヘッダに書き込む．以上により削除が完了する．

■更新 更新は，読み込みと同じくリーフノードが格納するものによって異なる動作をする．レコードへのポインタを格納している場合，スレッドはまずヘッダとメタデータエントリから更新対象のレコードが有効な状態（ノードが凍結状態でなく，かつ対象レコードが削除されていない）かを確認する．その後，新しいレコードへのポインタ，ヘッダ，及びメタデータエントリの更新を 3 ワードの PMwCAS により実行し更新が完了する．実際のペイロードを格納している場合，挿入と同じように実行される．レコードのためのスペースを確保した後，そこへ新しいペイロードとキーのペアを書き込むことで更新が完了する．なお，同じキーに対する同時更新は “last write wins” プロトコルにしたがって処理される．

■範囲スキャン Bz 木は範囲スキャンをサポートする．まず，ユーザから指定された範囲の始点となるキーを基に，最初のリーフノードを見つける．次に，そのノード上で有効なレコードを順番にリスト化して配列を構築する．そして，このとき構築された配列の中で最大のキーより更に大きいキーを用いて次のリーフノードを見つけ，再び配列を構築する．これらの処理を範囲の終点となるキーまで繰り返すことで範囲スキャンは実行される．

3.2 ノード分割

ノードのサイズがある一定値より大きくなった場合，そのノードは図 1 に示すような以下の手順で分割される．

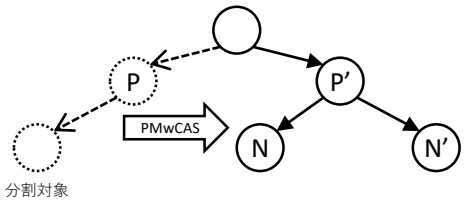

図 1 ノード分割

まず，分割するノードのヘッダを PMwCAS によって更新し凍結状態にする．次に，ノード上の有効な全レコードが均等に分割されるようなキーを 1 つ決め，新しく 3 つのノードを次のように初期化する．

- 決めたキー以下の全レコードをもつノード N
- 決めたキー以上の全レコードをもつノード N’
- 分割するノードの親 P を基とする N および N’ を子にもつノード P’

また，このときキーはソートされる．

新たなノードの割り当てに成功したら，3 ワードの PMwCAS によって競合を検出しつつ P の親がもつ P へのポインタを P’ に変更し，ノード分割が完了する．失敗した場合は再試行する．

4 評価・分析

オープンソースで公開されている Bz 木を使用する．まず通常の揮発性メモリ上で Bz 木を構築し，その基本的な構造や動作を確認する．また DCPMM の用意ができれば不揮発性メモリ上でも Bz 木を構築し，その動作を確認するとともに不揮発性メモリの特性や性能について検証する．特に，元論文では評価が不足していた範囲検索などの性能について調査する．

5 まとめ

本稿では，不揮発性メモリ向けの索引構造である Bz 木の概要及び標準的な操作の動作について説明した．今後は，Bz 木を揮発性メモリ及び不揮発性メモリ上で構築し，いくつかの実験によりその動作と性能特性を確認する予定である．

謝辞

本研究は JSPS 科研費（16H01722，19K21530，20K19804）の助成，及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）から得られた結果による．

参考文献

[1] S. Venkataraman, N. Tolia, P. Ranganathan, R. H. Campbell, *et al.*, “Consistent and durable data structures for non-volatile byte-addressable memory.,” in *FAST*, vol. 11, pp. 61–75, 2011.

[2] S. Chen and Q. Jin, “Persistent B$^+$-trees in non-volatile main memory,” *PVLDB*, vol. 8, no. 7, pp. 786–797, 2015.

[3] J. Arulraj, J. Levandoski, U. F. Minhas, and P.-A. Larson, “BzTree: A high-performance latch-free range index for non-volatile memory,” *PVLDB*, vol. 11, no. 5, pp. 553–565, 2018.

[4] T. Wang, J. Levandoski, and P.-A. Larson, “Easy lock-free indexing in non-volatile memory,” in *Proc. ICDE*, pp. 461–472, IEEE, 2018.