1N-01

B+ 木のマルチバージョン化による範囲走査性能への影響評価

桑村 真生 † 杉浦 健人 †† 野原 健汰 †† 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

近年では SAP HANA や Hekaton など，multi-version concurrency control（MVCC）を用いたインメモリデータベース管理システムが多数提案されている．既存の MVCC 研究においてタプルのバージョン情報は主にヒープテーブル側で管理されており，索引とは独立していることが多い．よって，索引層のマルチバージョン化に伴う性能への影響はいまだ検証されていない．

また，MVCC では記憶領域の回収・再利用などのために不要なバージョンのガベージコレクション（garbage collection, GC）が必要になる．MVCC においてはこの GC がボトルネックになりうることが先行研究 [1] で指摘されている．

そこで本研究では，従来の索引構造である B+ 木に対する操作をマルチバージョンに対応したものへ変更し，クラスタリング索引のマルチバージョン化に伴う性能への影響を明らかにする．特に，クラスタリング索引の強みの 1 つである範囲操作性能の劣化，および GC による書き込み操作性能の劣化の程度を調査する．

本稿では，まず B+ 木の概要について述べる．次に，マルチバージョン化された B+ 木のバージョニング，各種操作および GC の設計について述べる．最後に，B+ 木のマルチバージョン化に伴う影響の評価方法について述べる．

2 B+ 木の概要

B+ 木はデータベース管理システムの索引に用いられることが多い平衡木の一種である．中間ノードは子ノードへのポインタとキー値を保持し，葉ノードはキー値の順に並べられたレコードと兄弟ノードへのポインタを保持する．葉ノードどうしを繋げることにより，範囲検索におけるルートからリーフへの探索の回数が最小限で済むため，範囲走査を高速に実行できることが知られている．

上述した通常の B+ 木では，読み取りや書き込みの際の一貫性の保障のために，操作の対象となるノードに加えてその親ノードもロックしなければならない．一方，B+ 木の中間ノードに最大キーと兄弟ノードへのポインタを付与した B^{link} 木を用いることで，操作対象ノードのロックのみで読み書きする手法が提案されている [2,3]．図 1 に B^{link} 木の概観を示す．

本研究の実験においては，B^{link} 木を用いて読み書き対象ノードだけを楽観的にロックする手法 [3] を使用している．このため，以降本稿では B^{link} 木を B+ 木と呼ぶ．

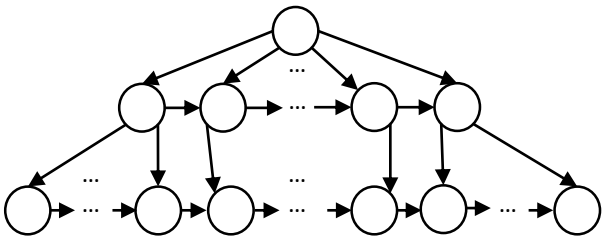

図 1 B^{link} 木の概観

3 B+ 木のマルチバージョン化

3.1 バージョンリストの設計

マルチバージョン化された B+ 木では各レコードの更新を新しいバージョンの生成で表し，レコード毎にバージョンの連なりを単方向リストとして管理することで複数バージョンの値を保持する．ここで用いる単方向リストをバージョンリストと呼ぶ．バージョンリストにはレコードのバージョンを新しい順に保存していく．すなわち，最も新しいバージョンはリストの先頭に，最も古いバージョンはリストの末尾に保持される．

B+ 木の葉ノードには各レコードの最新のバージョンを保持する．最新ではないバージョンは B+ 木とは独立した領域に保持されるが，最新バージョンはバージョンリストの先頭であるため，それを起点にすることで B+ 木から全てのバージョンにアクセスできる．

本研究では，バージョンの表現にバージョンレコードという構造体を用いる．バージョンレコードはタイムスタンプ，削除済みフラグ，レコードの値および次のバージョンレコードへのポインタを持つ．タイムスタンプはレコードが書き込まれた時刻を表し，この値が大きければ大きいほど新しいバージョンであることを意味する．削除済みフラグは，レコードが削除されていることを表すフラグである．

バージョンレコードの連なりであらわされたバージョンリストの例を図 2 に示す．図 2 では，あるレコードを値 A で挿入した後 B に更新し，その後削除している．

3.2 B+ 木に対する操作

マルチバージョン化された B+ 木に対する操作は以下のような内容になる．

■更新 まず，更新したいレコードを含むノードの排他ロックを取得する．次に，更新したいレコードの最新のバージョンを見つけ，それをコピーする．最後に，コピーしたバージョンレ

Evaluating the effect of multi-versioning of B+-trees on range scans

Masaki Kuwamura†, Kento Sugiura††, Kenta Nohara††, Yoshiharu Ishikawa††, and Kejing Lu††

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

††Graduate School of Informatics, Nagoya University

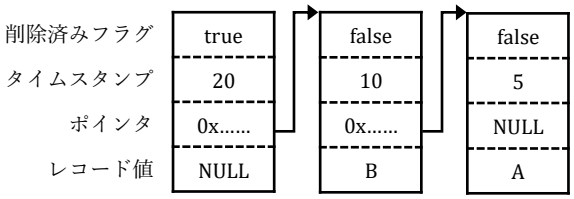

図 2　バージョンリストの例

コードを指すポインタと更新後の値を持つバージョンレコードを葉ノードに上書きし，ノードのロックを解放する．

■削除　葉ノードに書き込むバージョンの削除済みフラグがセットされている点を除いて更新と同一の処理となる．

■挿入　挿入操作は，挿入されたことがないキー値を持つレコードを初めて挿入する場合と，挿入された後に削除されたレコードと同じキー値を持つレコードをもう一度挿入する場合に分けられる．前者の場合は，キー値に応じて適切なノードを排他ロックし，そこにバージョンレコードを書き込む．このとき書き込むバージョンレコードは次のバージョンへのポインタを持たない．後者の場合は，更新操作と同一の処理となる．

■読み取り　シングルバージョンと同様に常に最新の値を取得する．

■範囲走査　まず，走査開始時点のタイムスタンプを取得する．次に，シングルバージョンと同様に次々とノードを走査し，値を読み取っていく．ただし，マルチバージョン化された B^+ 木においては葉ノードにある最新のバージョンを読むべきとは限らないため，必要に応じてバージョンリストを走査し適切なバージョンを読み取る．走査開始時に取得したタイムスタンプを t_s，あるレコード r が持つバージョンのタイムスタンプの集合を T_r とすると，読み取るべきバージョンは $t_r = \max\{t \mid t \in T_r \wedge t_s > t\}$ なるタイムスタンプ t_r を持つバージョンである．安全な読み取りのため，全てのレコードの，そのようなタイムスタンプ t_r を持つバージョンが（レコードが読み取られるかどうかに関係なく）後述する GC から保護される．

3.3　ガベージコレクション

マルチバージョン化された B^+ 木では，記憶領域の圧迫を防ぐため，およびバージョンリスト走査の高速化のために不要なバージョンの GC が必要になる．ただし，範囲走査によって保護されたバージョンは後から読み取られる可能性があるため削除してはいけない．

安全に削除可能なバージョンは，最新のバージョンと保護されたバージョンを除いた全てのバージョンである．タイムスタンプの単調増加性から，ある時点で保護されておらず最新バージョンでもないバージョンが後から保護されることはないためである．

本研究の GC には，レコードに書き込みを行うスレッドが削除可能なバージョンを協調的に削除する手法 [4] を用いる．この手法では GC 用のロックは取得せず，書き込み時に取得するロックを流用して排他的に GC を行う．

4　評価分析

まず，マルチバージョン化された B^+ 木への各操作，特にマルチバージョン化が大きく影響すると考えられる範囲走査の性能についてシングルバージョン B^+ 木との性能を比較する．次に，GC の有無を切り替えることで，書き込み時に行われる GC が更新操作に与える影響について評価する．

5　おわりに

本稿では，本研究で用いる B^+ 木の概要について述べ，更に B^+ 木をマルチバージョン化にした場合の設計について述べた．今後は，4 章で述べたように B^+ 木のマルチバージョン化に伴う影響評価のための実験を実施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] J. Böttcher, V. Leis, T. Neumann, and A. Kemper, “Scalable garbage collection for in-memory MVCC systems,” *Proc. VLDB Endow.*, vol. 13, no. 2, pp. 128–141, 2019.

[2] P. L. Lehman and S. B. Yao, “Efficient locking for concurrent operations on B-Trees,” *ACM Trans. Database Syst.*, vol. 6, no. 4, pp. 650–670, 1981.

[3] S. K. Cha, S. Hwang, K. Kim, and K. Kwon, “Cache-conscious concurrency control of main-memory indexes on shared-memory multiprocessor systems,” in *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB ’01, (San Francisco, CA, USA), pp. 181–190, 2001.

[4] Y. Wu, J. Arulraj, J. Lin, R. Xian, and A. Pavlo, “An empirical evaluation of in-memory multi-version concurrency control,” *Proc. VLDB Endow.*, vol. 10, no. 7, pp. 781–792, 2017.