ロックフリー索引のトライ木化による高速化に関する研究

井戸 佑[1,a)]　杉浦 健人[1,b)]　石川 佳治[1,c)]　陸 可鏡[1,d)]

概要：代表的な索引構造である B+木は様々なデータを格納可能な汎用性の高い索引である．一方で，扱うキーに制限を加え，索引構造を最適化させることで性能を向上させる研究も行われてきた．本研究では，著者らの研究室で開発しているロックフリー B+木（B^c 木）に対し同様の拡張および性能改善を行う．具体的には，扱うキーをバイナリ比較可能なものに制限することでトライ木の構造を適用し，空間利用効率の向上およびそれに伴う検索性能の向上を図る．

1. はじめに

2. 関連研究

2.1 B^c 木

2.2 Mass 木

3. B^c-forest の構造

B^c-forest は B^c 木を基本単位とする階層を持つ索引構造である．B^c-forest の概形を　　に示す．3 章では B^c 木との差異である，トライ木構造の適応と posting list の導入について説明する．

3.1 トライ木構造の適応

Mass 木は B^+ 木にトライ木構造を適応させることで，キャッシュ効率を改善させた．B^c-forest では B^c 木に対し，同様の改善を図る．

Mass 木の観点では B^+ 木から B^c 木になることにより，ロックフリーによる書き込み性能の改善が図る．また B^c 木の観点では，整数型や文字列型などのバイナリ比較可能なキーに対し，キャッシュ効率の改善が図る．

3.2 posting list の導入

Mass 木においては，空間利用効率が問題となる．・　は末尾 3 byte が異なる 19 byte の 2 キーを格納した Mass 木の索引構造である．先頭 8 文字が共通するため，layer1 を作成する．同様に 8〜16 byte 目が共通するため，layer2 を作成する．Mass 木では本来，1 つのノードに最大 16 個のレコードを格納することが出来る．しかし layer1 では，1 つのレコードしか格納されていない状態で layer2 を作成している．layer1 にのみ注目すると，空間利用率は約 6.25 %しかない．

B^c-forest では空間利用率の改善として，posting list を導入する．posting list では，1 つのキーに対して複数のペイロードを対応付けることが出来る．・　は posting list を導入した際の索引構造を示したものである．layer1 において posting list 作成することで，layer2 の無駄な階層化と空間利用率の悪化を防ぐ．

4. B^c-forest のノード操作

4.1 書き込み

4.2 読み取り

5. B^c-forest の構造変更操作

5.1 統合

5.2 分割

5.3 新層作成

6. おわりに

謝辞　本研究は JSPS 科研費 JP20K19804，JP21H03555，JP22H03594，JP22H03903 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] 名古屋大学大学院情報学研究科
Graduate School of Informatics, Nagoya University
[a)] ido@db.is.i.nagoya-u.ac.jp
[b)] sugiura@i.nagoya-u.ac.jp
[c)] ishikawa@i.nagoya-u.ac.jp
[d)] lu@db.is.i.nagoya-u.ac.jp