並行実行木 Masstree の調査

渡辺　敬之† 　川島　英之†† 　建部　修見††

† 筑波大学情報学群情報科学類　〒 305–8571 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター　〒 305–8571 茨城県つくば市天王台 1–1–1
E-mail: †watanabe@hpcs.cs.tsukuba.ac.jp, ††{kawasima,tatebe}@cs.tsukuba.ac.jp

あらまし　本論文では並行実行木 Masstree の一括挿入法を提案する．提案技法は 16 スレッドを用いた場合，既存手法に比べて 2.48 倍の高性能化を達成した．

キーワード　索引，並行実行制御，Masstree

1. 序　　論

複数のスレッドが同一のデータ構造にアクセスして READ/WRITE 検索を行うとき，並行データ構造が必要になる．並行データ構造において最も直観的な方式は全体を停止させ，高々1 つのスレッドだけにデータ構造へのアクセスを許す手法である．この手法はマルチコアやメニーコアが常識である現代において不適である．

効率的な並行データ構造として，2012 年に EuroSys 会議で発表された Masstree [4] がある．この論文では，Masstree が前述の全体停止を行わず，複数のスレッドが並行的にデータ構造へアクセスしても，性能が 16 スレッドまでスケールアップすることが示されている．Masstree は効率的な楽観的並行実行制御手法 Silo [6] で導入され，不揮発メモリを前提とした高性能 OLTP システム FOEDUS [2] にアイディアが用いられており，現代の高性能トランザクション処理システムの基礎の一つだと考えられる．

前述の通り，Masstree は範囲検索を含む処理が並行的に実行される時に高い性能を発揮することが分かっている．一方，巨大な入力データが存在する時，そのデータのために Masstree を高速に構築するには一括構築を行うことが自然な選択肢である．Masstree はトライ木のノード部分に B+木を格納する構造である．B+木，トライ木ならば，一括挿入法による高性能化手段がそれぞれ知られている [3] [1]．一方，Masstree の一括挿入法は我々の知る限り存在しない．

我々は Masstree の一括挿入法を新たに提案し，その性能を実験的に評価した．これを本論文は報告する．提案技法は Masstree がトライ木と B+木から構成されている点に着目する．データを整列した後，トライ木のレベルでのデータ分散を行い，各トライ木において B+木の構築を行う．

本論文の構成は次の通りである．2 章では Masstree とその再実装について述べる．3 章では提案手法である Masstree の一括挿入法を述べる．4 章では提案手法の評価結果を述べる．5 章では本論文の結論を述べる．

2. Masstree

Masstree は B+木とトライ木の性能を合わせた高性能なデータ構造である．2^{64} のファンアウトを持つトライ木であり，トライ木のそれぞれのノードが B+木で構成されている．トライ木は prefix を共有させることによって長いキーを効率良く探索することができる．一方，B+木は短いキーに対してはとても効率が良い．よってどちらの側面も持っている Masstree はより効率良く探索を行える．

言い換えると，Masstree は一つ以上のレイヤーから成る B+木で構成されている．それぞれのレイヤーは異なった 8 バイトに区切られたキーによってインデックスされている．図 1 にこの例を示す．トライ木のルートノードであるレイヤー 0 には各キーの 0-7 バイトの部分がインデックスされている．さらに 1 つ下のレイヤー 1 には各キーの 8-15 バイトの部分がインデックスされている．次に深いレイヤー 2 では各キーの 16-23 バイトの部分がインデックスされるというように以下同様に続いていく．つまり同じ h レイヤーにインデックスされているキー同士は同じ 8h バイトの prefix を持っているということである．

それぞれのレイヤーには 1 個以上の border ノードと 0 個以上の interior ノードが含まれている．border ノードとは一般的な B+木の leaf ノードとほぼ同じである．しかし，leaf ノードがキーとその value しか持たないのに対して，Masstree の border ノードは次のレイヤーへのポインターを持つ場合がある点が異なる．

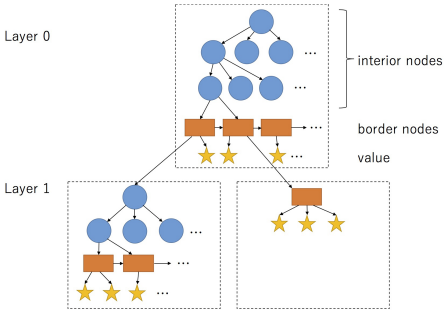

図 1　Masstree の構造

3. 提　　案

Masstree は B+木同様に挿入時に split が生じるため，多数の入力データから構造を構築するには一括挿入が望ましい．

我々が調査した限りでは Masstree の一括挿入法は存在しなかった為，その技法を提案する．提案技法は Masstree がトライ木と B+木から構成されている点に着目する．データを整列した後，トライ木のレベルでのデータ分散を行い，各トライ木において B+木の構築を行う．これをアルゴリズム 1 に示す．ここで，data は挿入をするデータの集まりであり，len はそのデータの数を表している．また，node は B+木におけるノードを表しており，nkeys はそのノードに現在入っているキーの数を示している．2 行目からのループでこれから挿入したいデータを全て入れ終わるまでループを行っている．同じ prefix を持つキーを挿入をする場合の処理は 11 行目から 21 行目に示している．

Algorithm 1 一括挿入法

```
Data: char** data, int len
 1: node = create new border node;
 2: for i = 0 to len - 1 do
 3:     while node→next != NIL do
 4:         node = node→next;
 5:     end while
 6:     n = node;
 7:     key = Key(data[i]);
 8:     value = data[i].value;
 9:     next_layer:
10:     nkeys = n→nkeys;
11:     if n→keyslice[nkeys-1] == key then
12:         if n→lv[nkeys-1] does not have next layer then
13:             create new layer;
14:         end if
15:         shift key to next 8 bytes;
16:         n = n→lv[nkeys-1].next_layer;
17:         while n→next != NIL do
18:             n = n→next;
19:         end while
20:         goto next_layer;
21:     end if
22:     if nkeys == NODE_MAX then
23:         nn = create new border node;
24:         nn→version = n→version;
25:         n→next = nn;
26:         nn→prev = n;
27:         insert n, nn, and key to interior node;
28:         n = nn;
29:     end if
30:     insert key and value to n;
31:     n→nkeys = n→nkeys + 1;
32: end for
```

4. 評価結果

本章では前章で紹介した提案手法の評価を行うために実施した実験の概要と結果を示す。

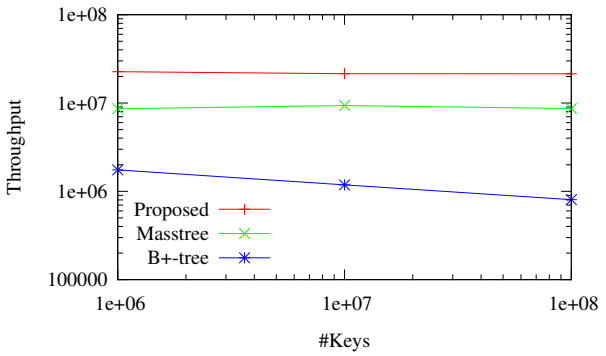

図 2　データサイズの影響

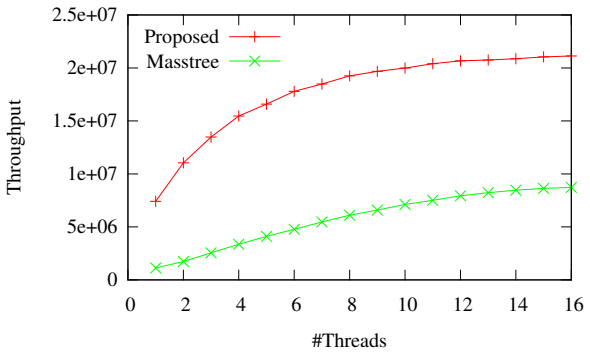

図 3　スレッド数の影響

5. 実験環境

表 1 に示す環境で実験を行った。実験に用いたノードの仕様は次の通りである：プロセッサが Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz，コア数は 24，メモリは 64GB，OS は CentOS release 6.6 (Final)．キー長は 10 とした．整列には並列基数ソート [5] を用いた．

表 1　実験環境

OS	CentOS release 6.6 (Final)
Kernel	Linux 2.6.32-642.1.1.el6.x86_64
CPU	Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz × 2
# of CPU cores	12 × 2
Memory	64 GB

提案手法，Masstree，B+木に関してデータサイズを変えながら挿入スループットを測定した結果を図 2 に示す．いずれもスレッド数は 16 とした．Masstree が B+木よりも高い性能を示し，提案手法が Masstree よりも優れることが観察された．キー数が 1 億の時，提案手法は Masstree，B+木に対してそれぞれ 2.48 倍，26.6 倍の性能向上を示した．

提案手法と Masstree についてスレッド数を変えながら挿入スループットを測定した結果を図 3 に示す．提案手法が全てのスレッド数で Masstree を上回ることが観察された．

6. 結　　論

本論文では並行実行木 Masstree の一括挿入法を提案した．

提案技法は 16 スレッドを用いた場合，既存手法に比べて 2.48 倍の高性能化を達成した．

謝　　辞

本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」，JST CREST「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」，JST CREST「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」，科研費「#16K00150」による．

文　　献

[1] Daniar Achakeev and Bernhard Seeger. Efficient bulk updates on multiversion b-trees. *PVLDB*, Vol. 6, No. 14, pp. 1834–1845, 2013.

[2] Hideaki Kimura. Foedus: Oltp engine for a thousand cores and nvram. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 691–706, 2015.

[3] Dongzhe Ma and Jianhua Feng. A generic approach for bulk loading trie-based index structures on external storage. In *Web-Age Information Management - 15th International Conference*, pp. 55–66, 2014.

[4] Yandong Mao, Eddie Kohler, and Robert Tappan Morris. Cache craftiness for fast multicore key-value storage. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pp. 183–196, 2012.

[5] Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey. Fast sort on cpus and gpus: A case for bandwidth oblivious simd sort. In *SIGMOD Conference*, pp. 351–362, 2010.

[6] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 18–32, 2013.