# メニーコア環境における索引性能の比較調査

名古屋大学 大学院情報学研究科 石川佳治

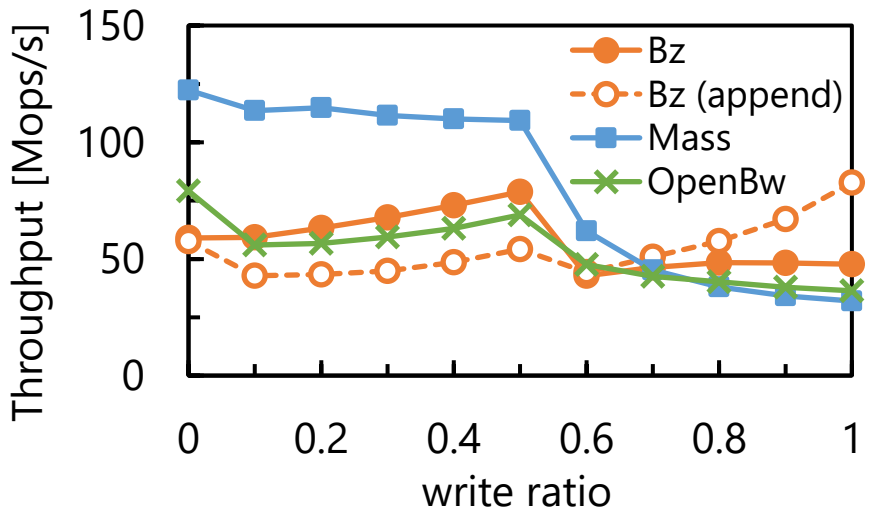
# 背景｜並列処理における索引構造

CPUコア数の増加に伴う並列処理の需要増加

- **1ソケットで64コア128スレッド**のCPUの登場
  - RDMAによる低遅延分散並列処理で更に加速する可能性あり

**DB内の索引も並列処理に適したもの**が必要

- 並列で書き込んでも構造が破綻せず，
  かつ読み取りを止めない
- 書込み時の一貫性の保証方法によって
  書込みの割合に対する性能傾向も変化




**read/writeの割合に対するスループットの変化**

# 目的｜並列処理における索引性能の評価

最新の**索引構造の並列処理における基礎性能**は不明瞭

- 特に書込みの競合が多い場合の評価は少ない
- そもそも実装を公開しておらず評価できないものも

基礎となるB+木ベースの索引について再現実装を行い，
**索引に対する同時実行制御が性能に与える影響**の調査

- Bw木[1]， Bz木[2]， B+木（悲観的[3]・楽観的ロック[4]）
- 今後様々なDBでこれらの手法やそれと同等なものが実装された際に，
  索引選択を行う一つの指針となれば

# アウトプット（予定）

**論文**

- 各索引構造の比較結果をまとめたもの
- 再現実装に使用する機能（multi-word CASなど）も論文化予定

**参考実装**

- 各種索引構造および評価用ベンチマーク
  - Bw木，Bz木，B+木（悲観的・楽観的ロック）
  - 可能な範囲で追試や索引の追加が容易な形で実装中
- 再現実装に使用する内部ライブラリ

# 現在の評価状況

# 性能比較結果の概要

現状で得られている比較結果について一部紹介

- 実験環境やベンチマークの詳細は付録を参照
- また，実装途中のものもあるため最終的な結果と異なる可能性あり

**現状の所感**

- 書込みの競合がそこまでない環境ならば**Masstreeが速い**
  - Tsurugi内ではMasstreeを使用する予定
- **ロックフリー索引（Bw木・Bz木）は期待されているほどの性能が出ていない**
  - **ポテンシャルはある**が，ロックフリーな構造を活かしきれていない

# 比較対象

**再現実装の対象**（現時点ではBz木のみ実装完了）

- Bw木[1]：SQL Serverで使用されている追記型のロックフリーB+木
- **Bz木**[2]：直接更新・追記の2つのモードを持つロックフリーB+木
- B+木（悲観的ロック[3]）：PostgreSQLが使用しているものと同じ手法を使用
- B+木（楽観的ロック[4]）：近年提案された楽観的ロックに基づくB+木

**参考比較**

- **Masstree**[5]：Silo向けに開発された索引（Tsurugiとは別実装のもの[7]を使用）
- **OpenBw木**[6]：Bw木のオープンソース実装[8]

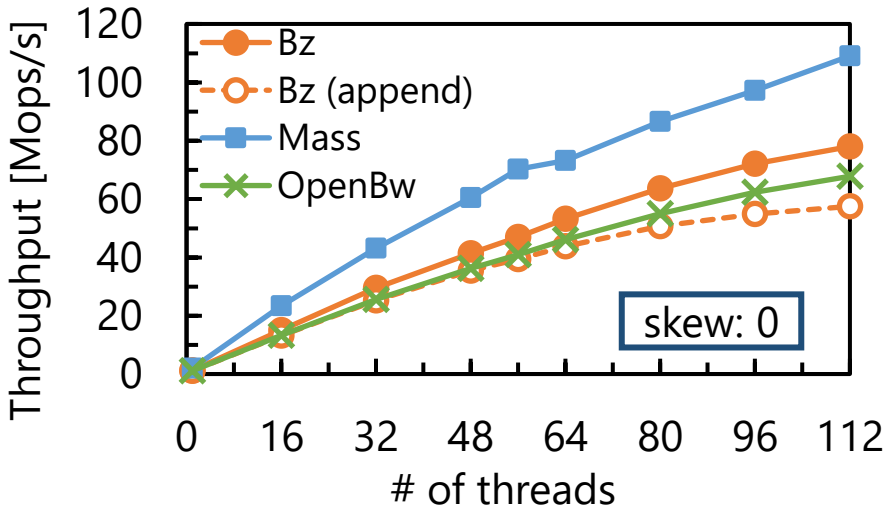
# 比較結果｜Balancedワークロード（Read:Write = 1:1）

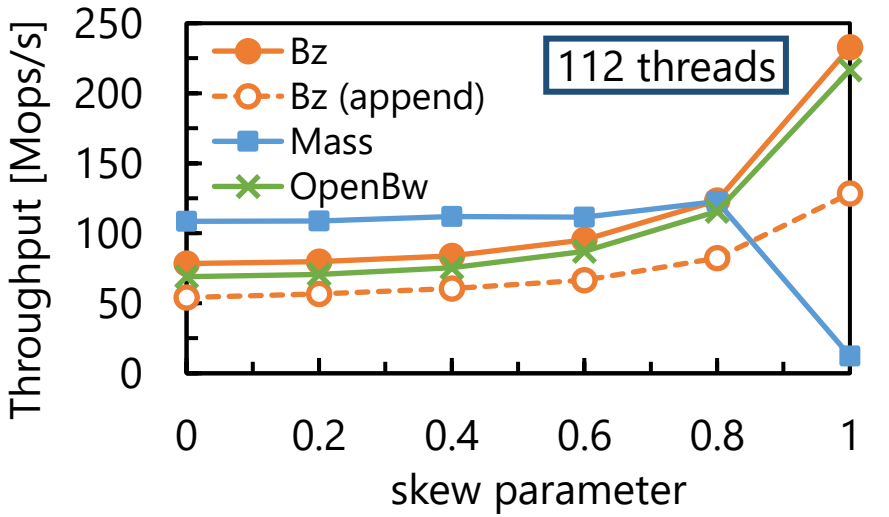
基本的にMasstreeが最も高速

- 一方で競合発生時の性能悪化は大きい
  - ただし，使用している実装が悪い可能性あり

その他ロックフリー索引はおよそ同傾向

- 偏りを与えるとキャッシュヒットが増え高速化
  - ただし，**必ずしも競合に強いわけではない**
  - 詳細はwrite onlyワークロードで



# 比較結果｜Write Onlyワークロード

追記型のBz木が最も高速

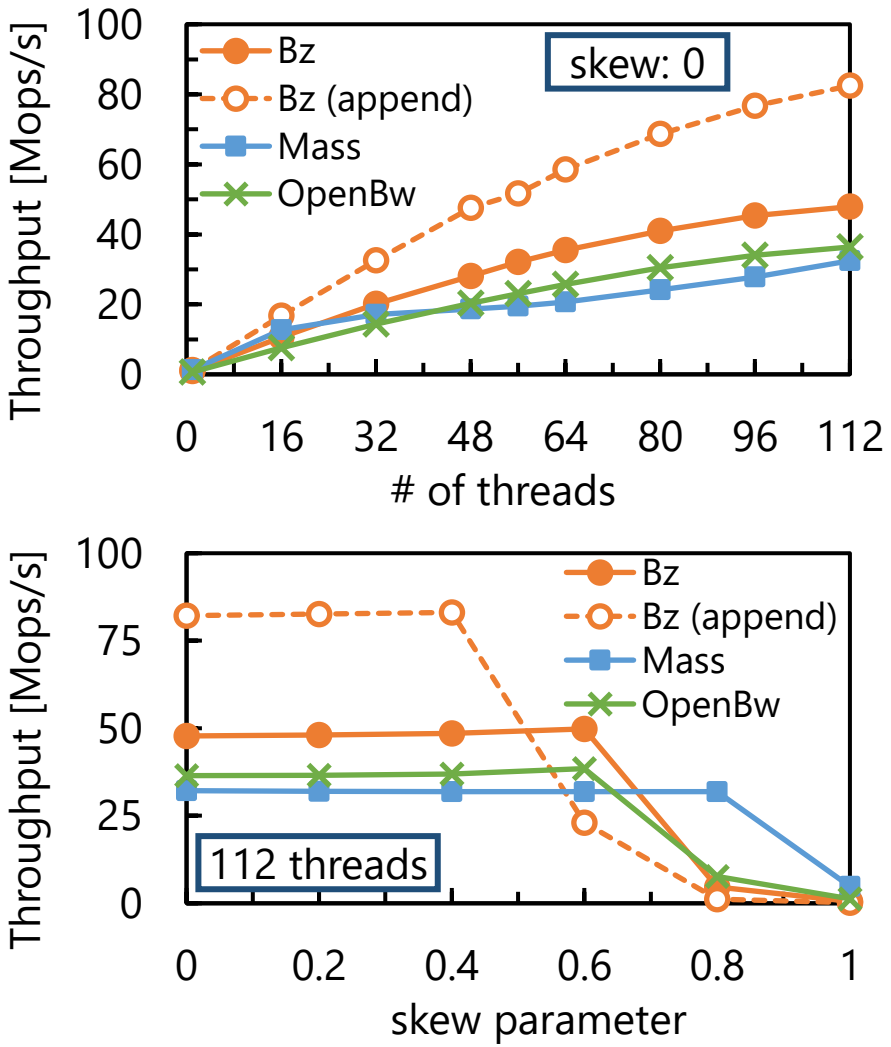
- **CPUキャッシュを汚さずに更新が可能**

一方で**同じ追記型のOpenBw木は伸びが悪い**

- ロックフリー化のための一貫性保証が悪影響？

キーに偏りを与えるといずれも性能が悪化

- MasstreeよりもBw木・Bz木の方が悪化がはやい
- Bw木・Bz木の構造を見る限り，
  **ロックフリー索引の性能はまだ改善できる**はず



# まとめ

メニーコア環境における索引性能の比較評価を実施中

- **Masstreeは競合が少なければ安定して高速**
- 現状のロックフリー索引の性能はやや疑問
  - ただし，**ロックフリー化による書込み性能向上**はポテンシャルあり

比較評価の結果は論文として，
再現実装した索引およびベンチマークはOSSとして公開予定

- 可能な限り追試や対象索引の追加が行いやすい形で実装中

# 参考文献


1. J. J. Levandoski et al., “The Bw-Tree: A B-Tree for New Hardware Platforms,” In Proc. ICDE, pp. 302-313, 2013.
2. J. Arulraj et al., “BzTree: A High-Performance Latch-Free Range Index for Non-Volatile Memory,” PVLDB, Vol. 11, No. 5, pp. 553–565, 2018.
3. P. L. Lehman et al., “Efficient Locking for Concurrent Operations on B-Trees,” ACM TODS, Vol. 6, No. 4, pp. 650-670, 1981.
4. V. Leis et al., “The ART of Practical Synchronization,” In Proc. DaMoN, No. 3, pp. 1-8, 2016.
5. Y. Mao et al., “Cache Craftiness for Fast Multicore Key-Value Storage,” In Proc. EuroSys, pp. 183-196, 2012.
6. Z. Wang et al., “Building a Bw-Tree Takes More Than Just Buzz Words,” In Proc. SIGMOD, pp. 473-488, 2018.
7. https://github.com/kohler/masstree-beta
8. https://github.com/wangziqi2016/index-microbench