Research Session 13: Similarity Search and Join

Efficient Metric Indexing for Similarity Search

Lu Chen[1], Yunjun Gao[1], Xinhan Li[1], Christian S. Jensen[2], Gang Chen[1]

1: College of Computer Science, Zhejiang University, Hangzhou, China

2: Department of Computer Science, Aalborg University, Denmark

概要

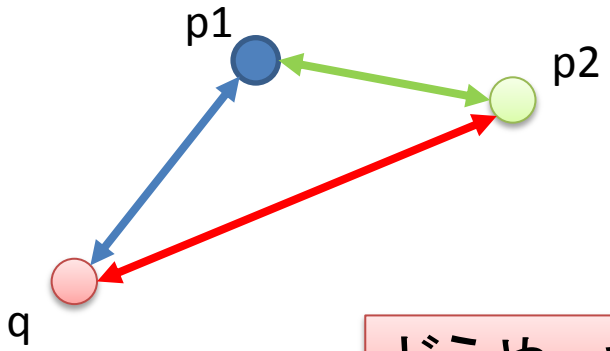
- 高次元なデータに対するSimilarity Searchを高速にする索引技術を提案
- ピボットベースと空間充填曲線の２段階マッピングを用いた索引技術
  - 距離計算の削減とB+-Treeの活用
- データの制約はメトリック空間であること
  - 三角不等式が成り立つこと
- 従来手法に比べ、構築と空間コストの削減と、Similarity Searchの効率化を達成

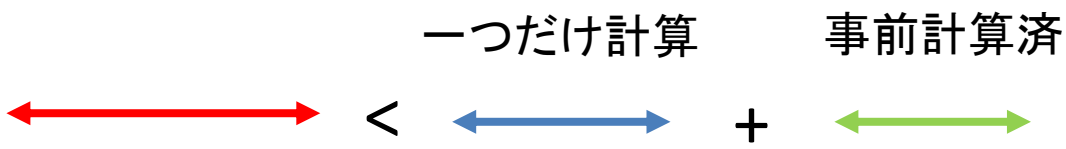
モチベーション

- 距離計算回数を減らしたい
  - 計算コストは、次元数に比例
    - ピボットベースのマッピング手法で削減
- ページアクセス数を減らしたい
  - なるべく同一ページに類似するデータをいれたい
    - 空間充填曲線を用いて、近傍データを同一ページにまとめる
    - 一次元のB+-Indexを利用することができ、効率的なページアクセスを実現

ピボットベース手法

- Pre-computationによる距離計算削減手法
  - ピボット間の距離を事前に計算
  - 三角不等式が成り立つことが仮定できると、あるピボットとの距離を計算するだけで、他のピボットとの距離の上限が計算可能





どうやっていいピボットを選ぶかが鍵

ピボットマッピング

- オブジェクトを、オブジェクトとピボットの距離を要素に持つベクトルにマッピング

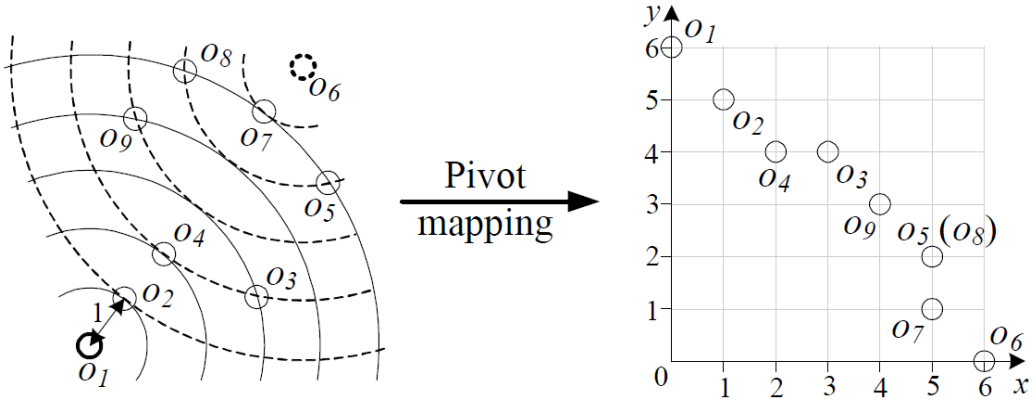


Fig. 2. Pivot mapping

図は論文より引用

- 理論的な解析によると、なるべく遠く離れた点をピボットを選んだほうがよい（本文参照）

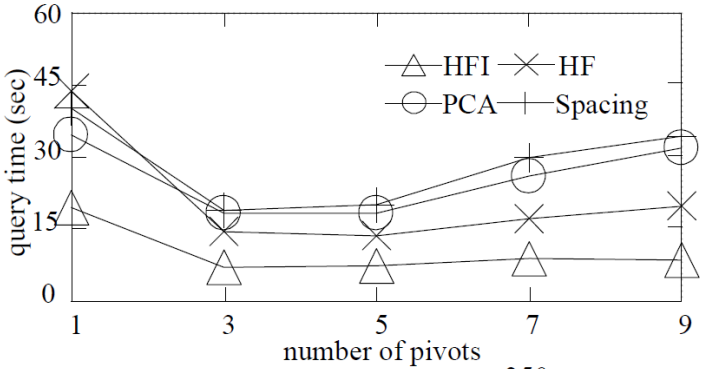
ピボット選択：HFI

- Outlier発見アルゴリズムHF[20]を利用
  - HFの詳細については割愛

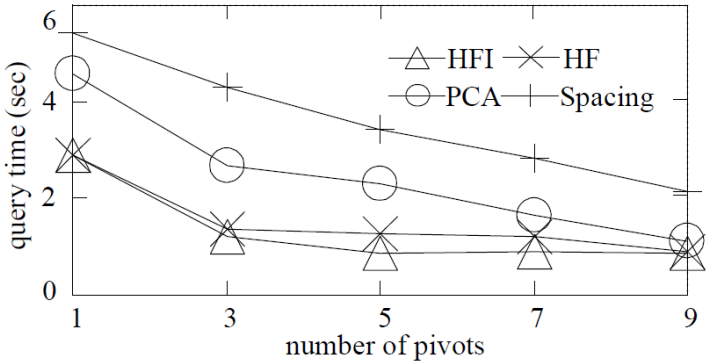
- アルゴリズム
  - HFでピボット候補集合を求める
  - 候補集合からから、評価指標を最大化するピボットを貪欲法で抽出

評価結果（ピボット選択）

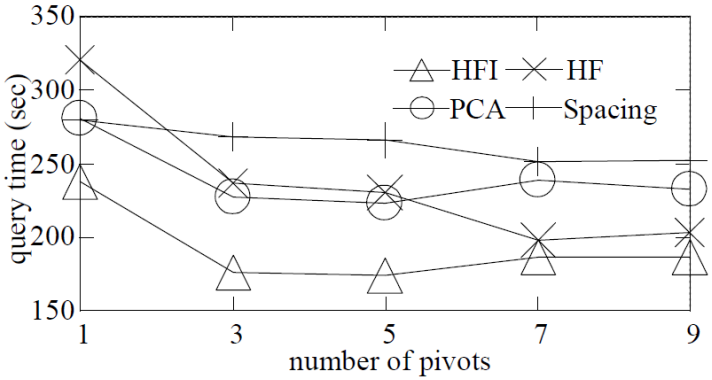
- 様々なデータセット・距離関数で評価
  - Word（edit dist.）, Color（L5-norm）, DNA（cosine dist.）



(c) *Words*



(f) *Color*



(i) *DNA*

図は論文より引用

High Performance Temporal Indexing on Modern Hardware

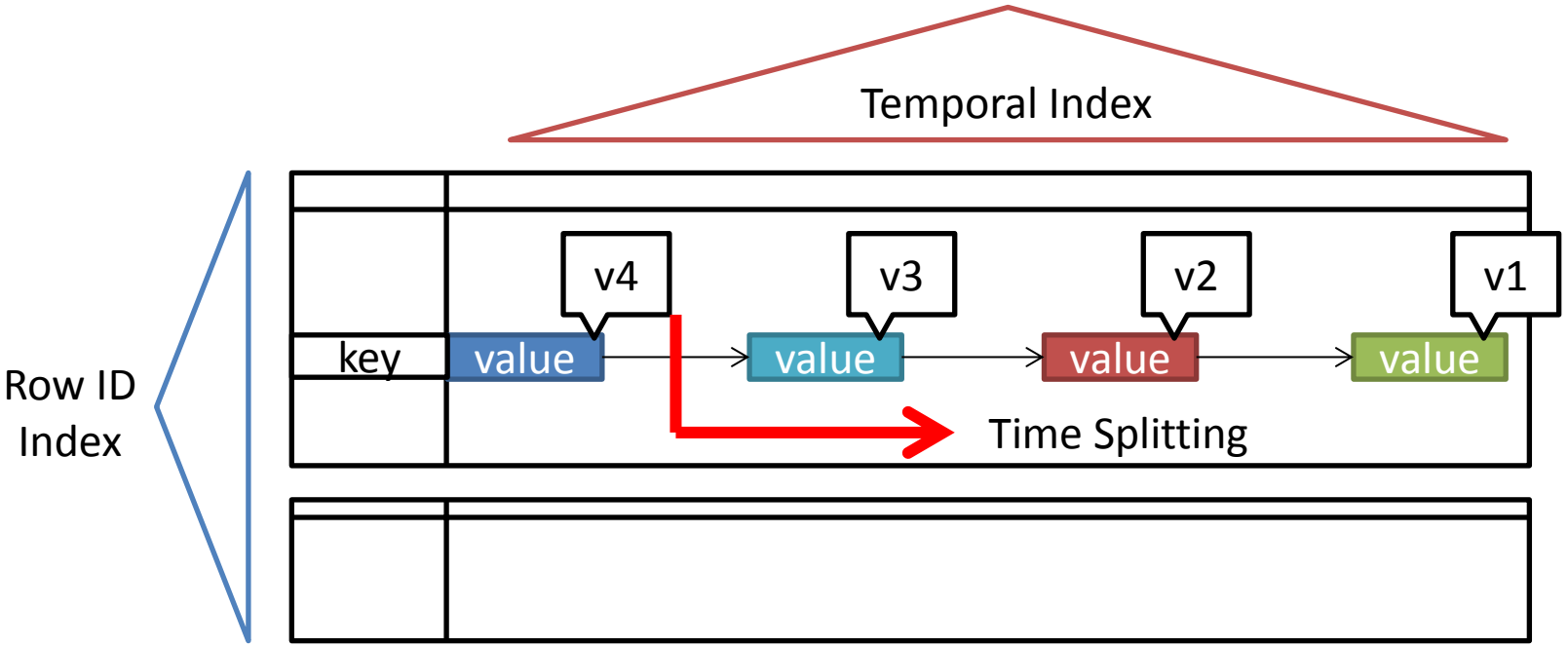
David B. Lomet[1], Faisal Nawab[2]

1: Microsoft Research, Redmond

2: Department of Computer Science, University of California, Santa Barbara

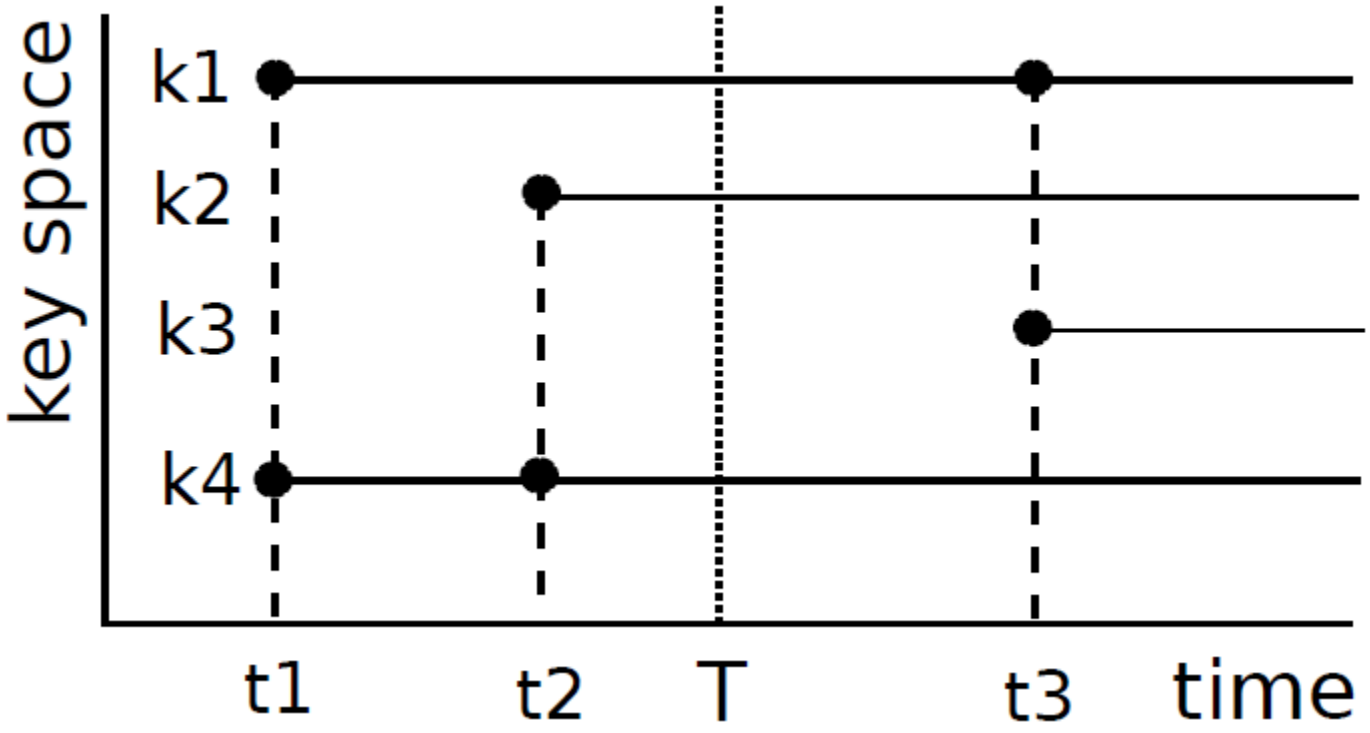
概要

- 値の履歴をもつTransaction Time Databaseのデータ管理をLatch-freeで実現
- 時刻分割をする際、データの重複持ちや分割する時刻の制約を解決



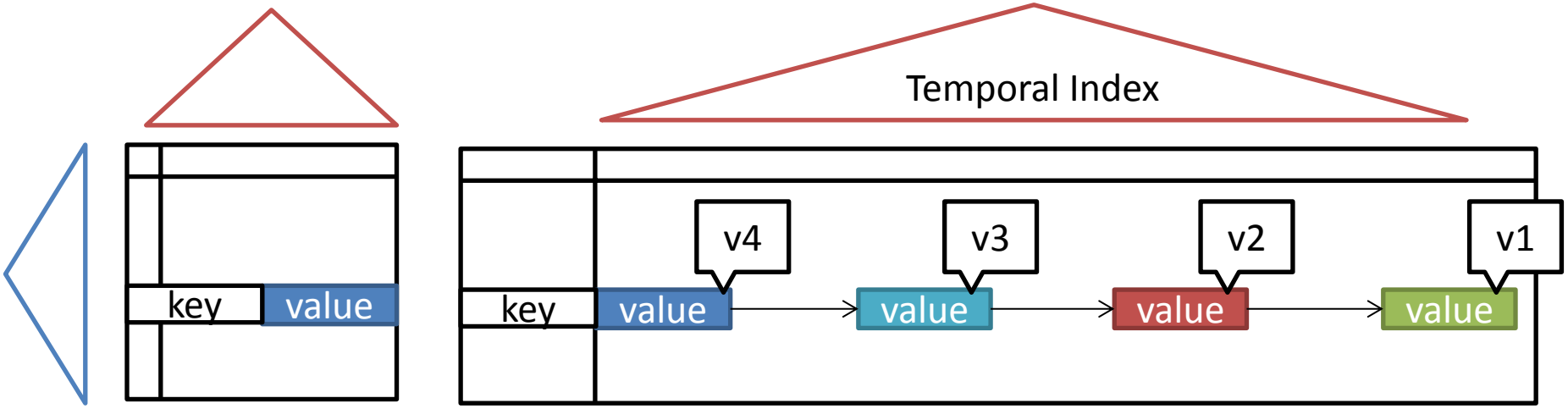
なぜ時間分割が難しいのか？

- ある時刻で分割した時、それをまたぐデータをどのように扱うか？



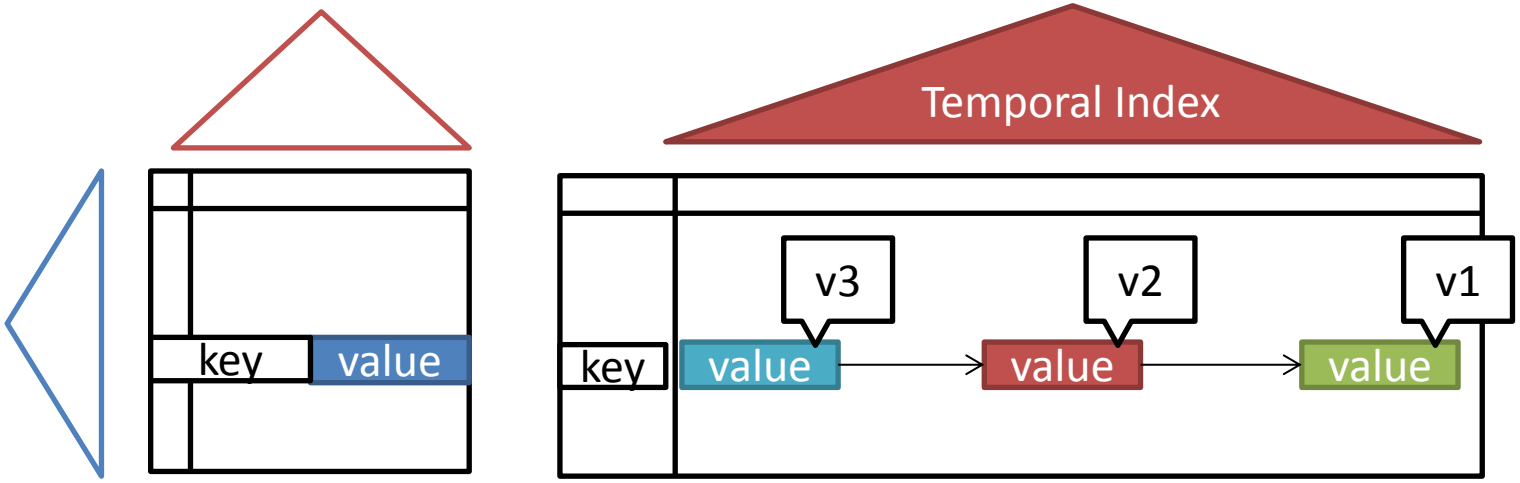
従来方法1：WOB-Tree(重複持ち)

- ある時刻以前でページを塩漬け
- 必要な最新データだけ新しいページにコピー
- 利点：履歴データやTemporal Indexはそのまま
- 欠点：データの重複持ちが発生



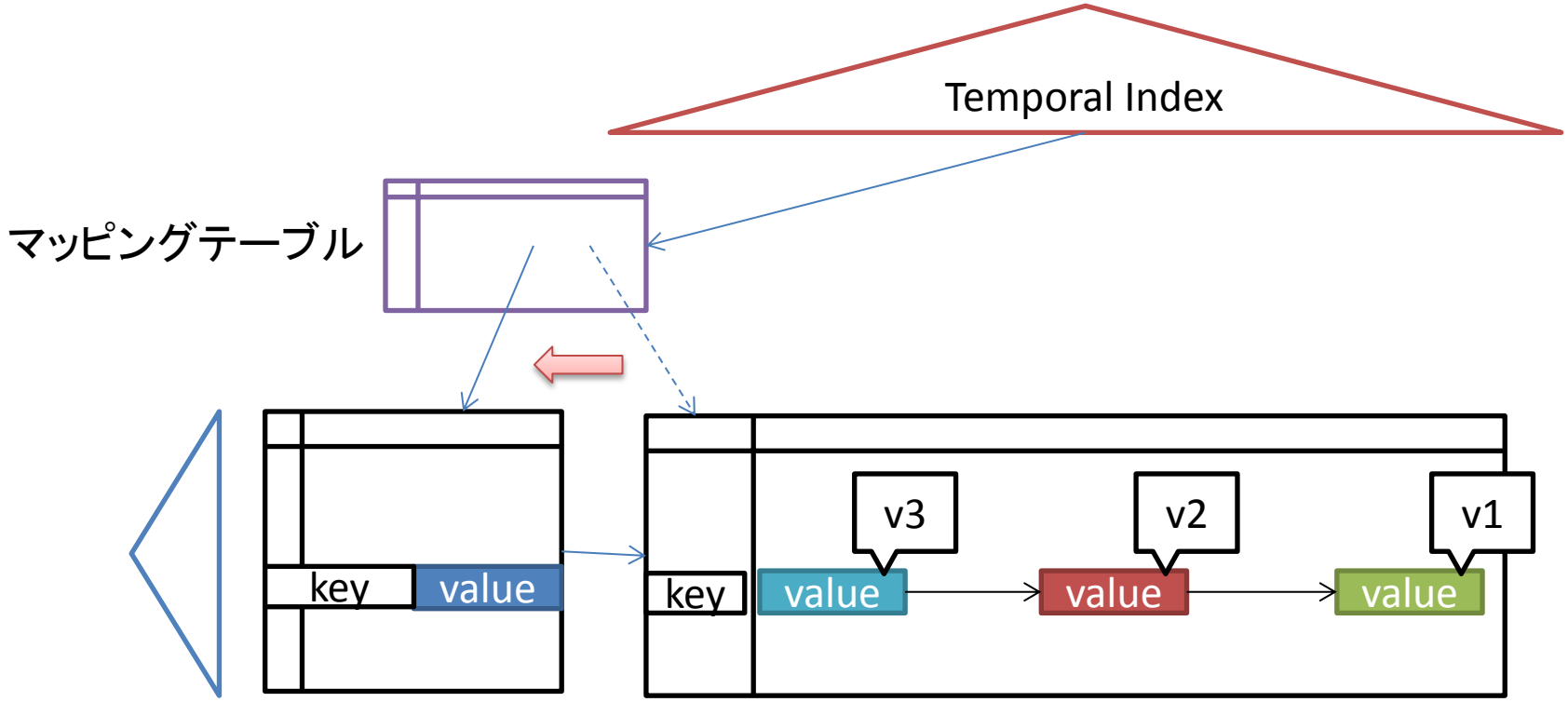
従来方法2: TSB-Tree

- ある時刻以前のデータを新しいページに追い出す
- 利点：データの重複がない
- 欠点：Temporal Indexを修正せずにすまそうとすると分割できる時刻に制約

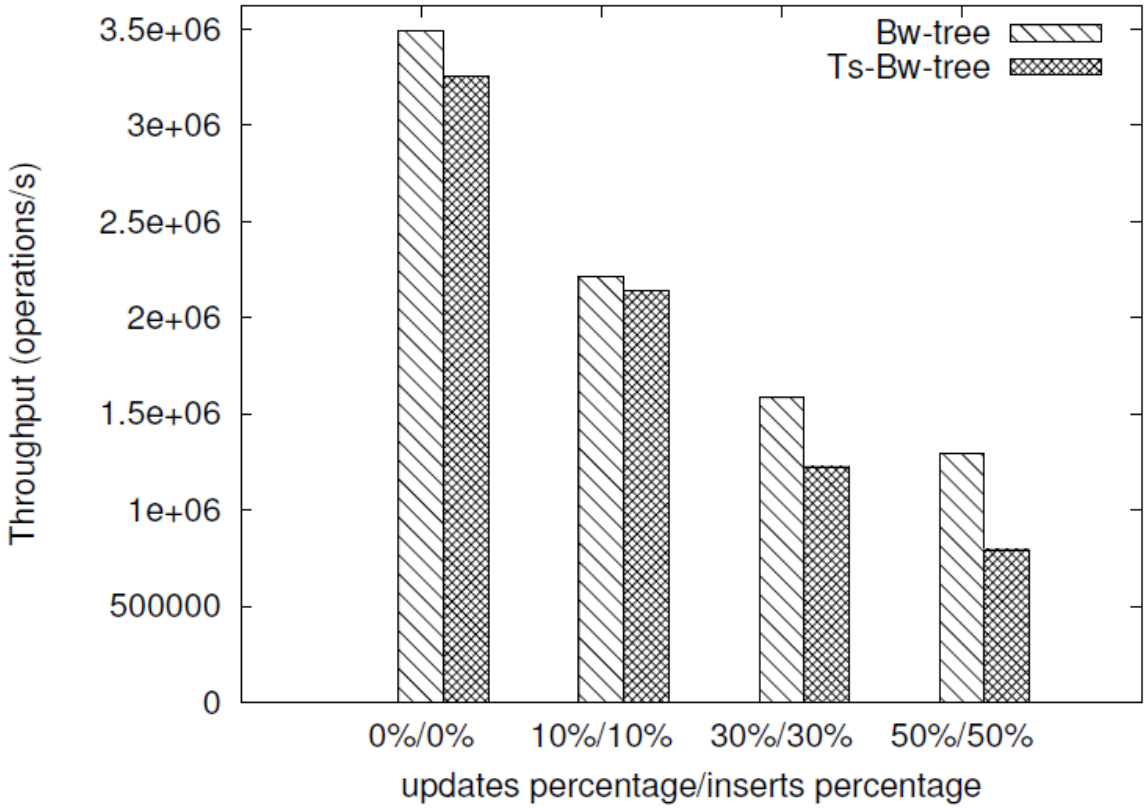


提案手法

- マッピングテーブルを介してページにアクセス
- 物理的なページの場所は変わるが、論理的なページの場所は変わらない



評価


Bw-tree
Ts-Bw-tree
Throughput (operations/s)
3.5e+06
3e+06
2.5e+06
2e+06
1.5e+06
1e+06
500000
0
0%/0%
10%/10%
30%/30%
50%/50%
updates percentage/inserts percentage