【ICDE2014勉強会】

Session 17: Main-Memory Databases

担当：友田，礒田（日立製作所）

目次

1. Exploiting Hardware Transactional Memory in Main-Memory Databases (DE140012)
2. Locality-Sensitive Operators for Parallel Main-Memory Database Clusters (DE140048)
3. Rethinking Main Memory OLTP Recovery (DE140247)
4. Optimal Hierarchical Layouts for Cache-Oblivious Search Trees (DE140646)

目次

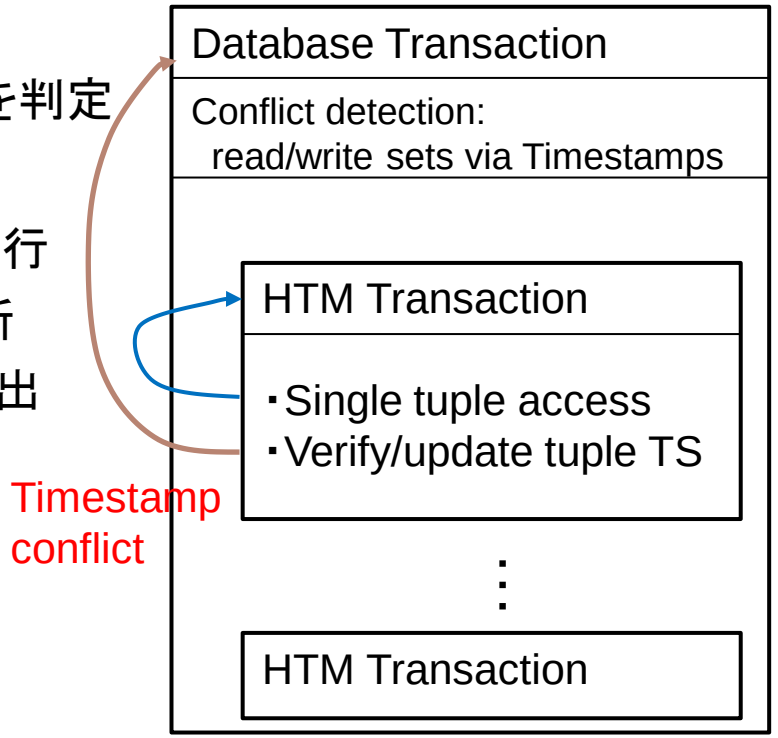
1. Exploiting Hardware Transactional Memory in Main-Memory Databases (DE140012)
2. Locality-Sensitive Operators for Parallel Main-Memory Database Clusters (DE140048)
3. Rethinking Main Memory OLTP Recovery (DE140247)
4. Optimal Hierarchical Layouts for Cache-Oblivious Search Trees (DE140646)

背景と課題

- 背景
  - Disk-based DBからIn-memory DBへの変化
    - 従来はI/O latencyに比べてlock/latch時間は小さかったが、これが全体に占める割合が増加
  - Intel HaswellでのHardware Transactional Memory(HTM)の登場
    - HTMを活用したlock/latch処理の軽減の必要性
- 課題
  - DBのTXをHTMのTXにそのままmapは出来ない
    - Intel HTMではread/write setをL1$にバッファしているので、多数のデータにアクセスするとL1$から溢れてしまい、HTM TXがabortしてしまう。

提案手法

- DB TXを複数のHTM transactionに分割
  - Database Transaction
    - TXのread/write set全体のTimestampを判定
  - HTM transaction
    - 個別のデータにアクセスはHTM TXで実行
    - Read/writeするごとにデータのTSを更新
    - HTM TX間のread/write setの衝突を検出



- データ配置：HTMに適応したデータ配置方法
  - HTMの判定がcache line単位のため、同一cache lineに複数のデータが載らないようにする。

評価

- 環境:
  - CPU: Intel i5 4670T Haswell Processor(4core), 6MB L3$, 2.9GHz, DRAM: 16GB. OS: Linux 3.10. Workload: TPC-C

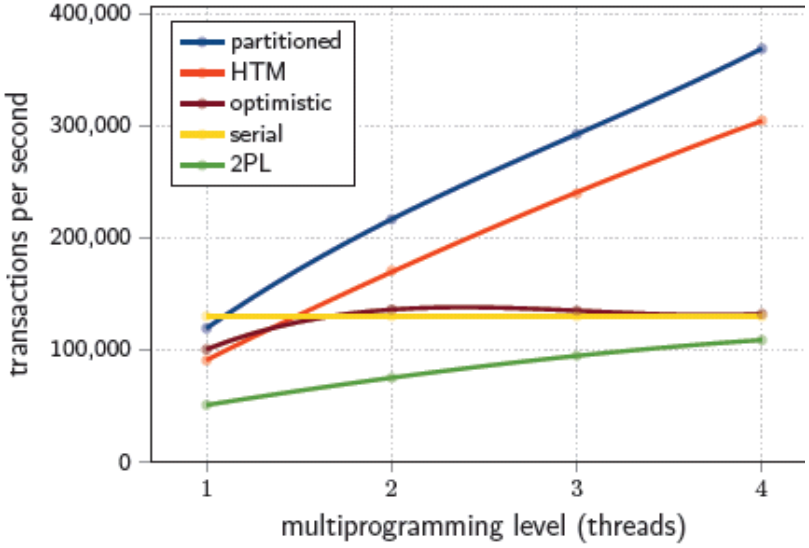


Figure 12. Scalability with TPC-C on desktop system

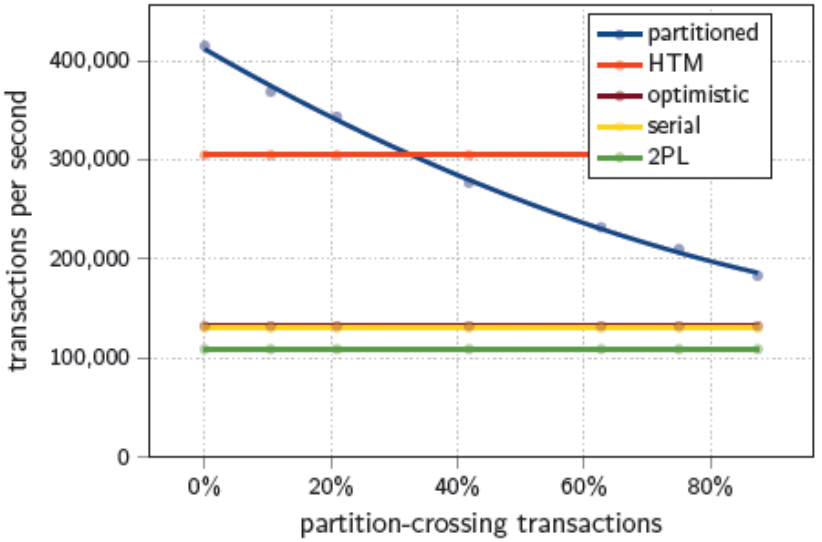


Figure 13. TPC-C with modified partition-crossing behavior

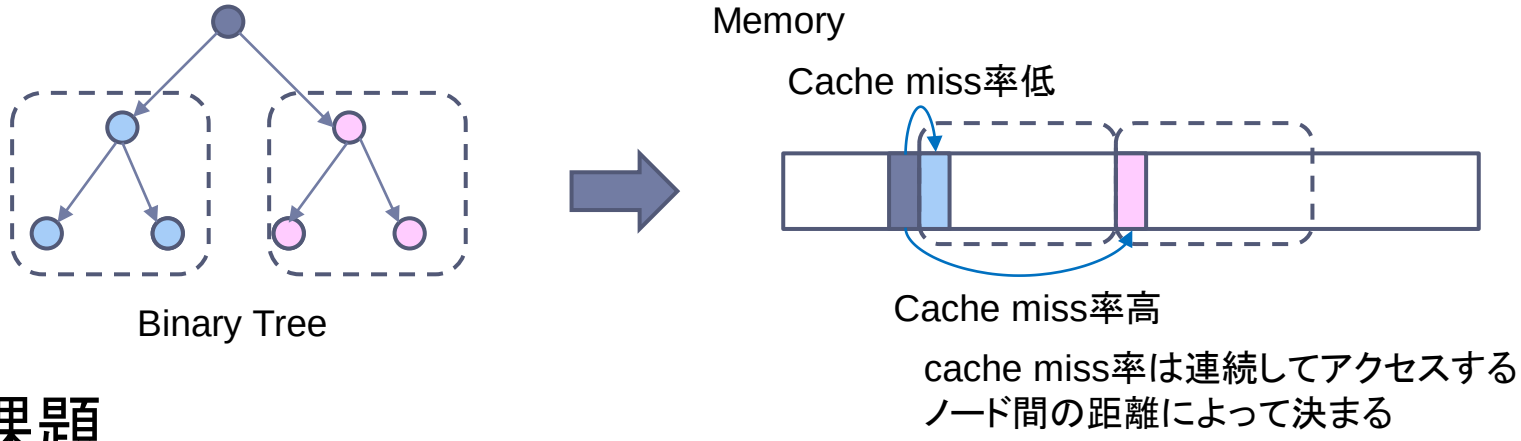
- 結果
  - Partitioned, HTMだけがスケールする。
    - Partitionedは人力でpartitionしている。
  - TPC-Cのpartitionを跨ぐ比率を増加させるとHTMは一定

目次

1. Exploiting Hardware Transactional Memory in Main-Memory Databases (DE140012)
2. Locality-Sensitive Operators for Parallel Main-Memory Database Clusters (DE140048)
3. Rethinking Main Memory OLTP Recovery (DE140247)
4. Optimal Hierarchical Layouts for Cache-Oblivious Search Trees (DE140646)

背景と課題

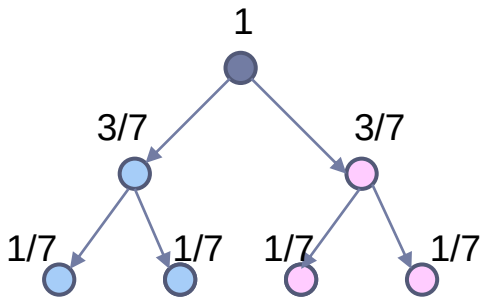
- 背景
  - メモリ階層やキャッシュ構成の多様化，複雑化
  - これらの違いに拠らない最適なツリーのレイアウトの必要性
  - Binary Treeのレイアウトによりキャッシュミス率が大きく変化



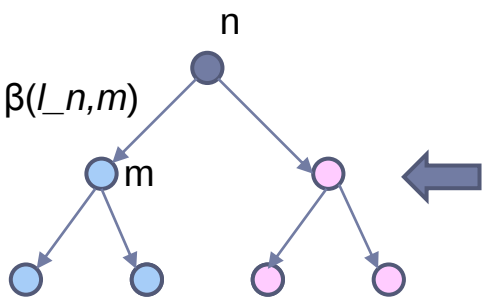
- 課題
  - Hierachical Layout, Recursive Layoutの中で，
    もっとも最適なTreeのレイアウトはどのようなものか？

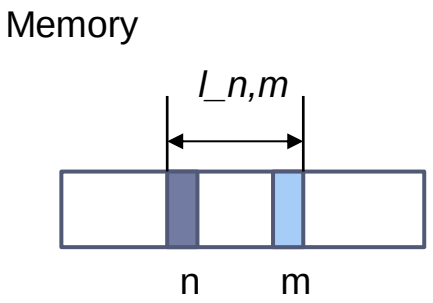
提案手法1

- Cache miss率と相関の強い評価関数を構成
  - Weighted Edge Product *v*_0
    - [概要] 以下の2つの要素を全ノードとエッジについて足し合わせ:
      - Weight: Treeの探索時に各ノードに到達する確率
      - β: ノードから子ノードにたどる時のcache miss率のlog
        (ノード間の距離 *l_n,m* の関数)



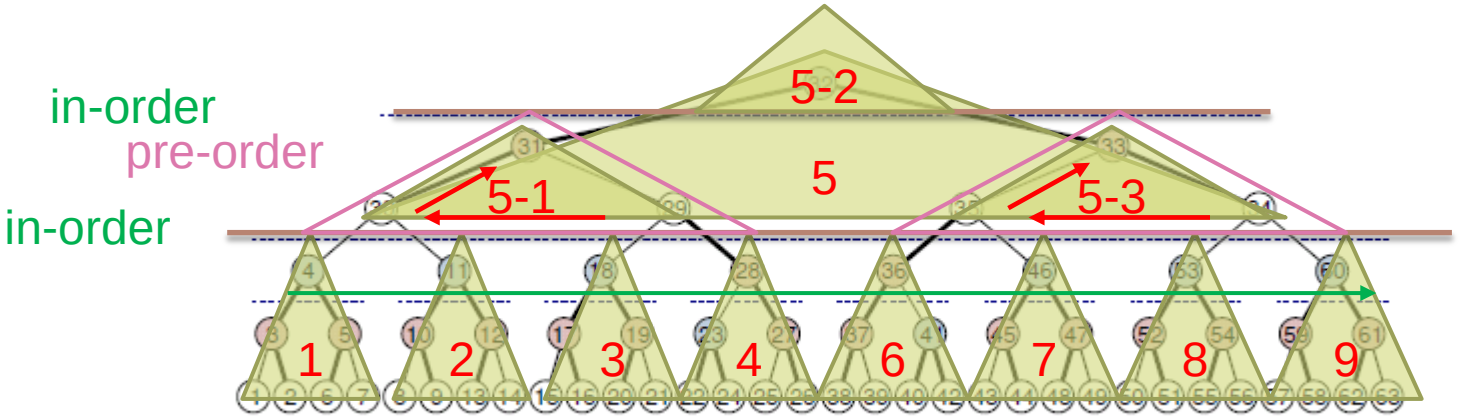
Weight:
全ノード(7)のうち，探索時に
各ノードを通る確率





提案手法2

- 評価関数を最小化するレイアウトを求める
  - Top階層はin-order, 下の方はpre-order
  - 下の階層のsubtreeの順序を逆順にする



(a) MinWEP = HalfWEP: $\nu_0 = 1.863, \nu_1 = 4.140, \mu_1 = 3.097, \mu_\infty = 26$

Fig. 7 (a)

評価

- 環境
  - CPU: 2.8GHz Intel Xeon X5660(Westmere-EP) 6core x 2socket
    12MB L3$, 256KB L2$, 32KB L1$ per socket
  - DRAM: 96GB
- 結果
  - 従来のlayoutから20%性能向上

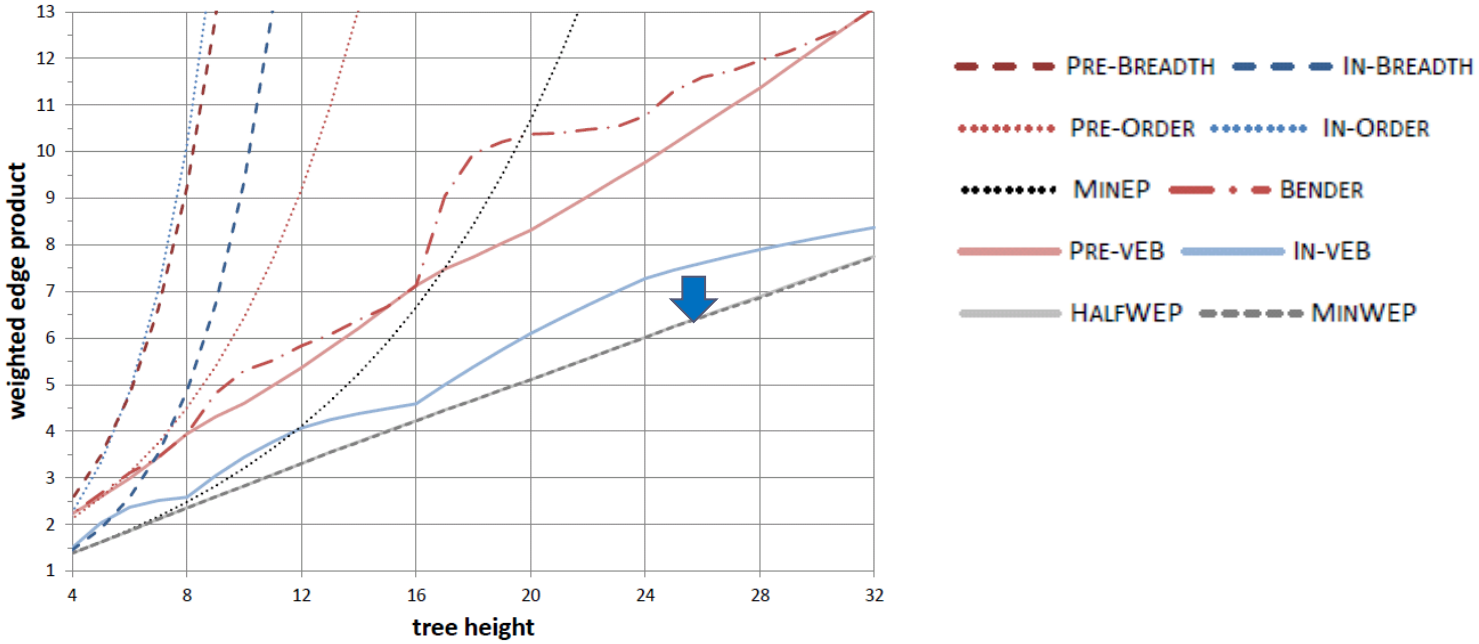


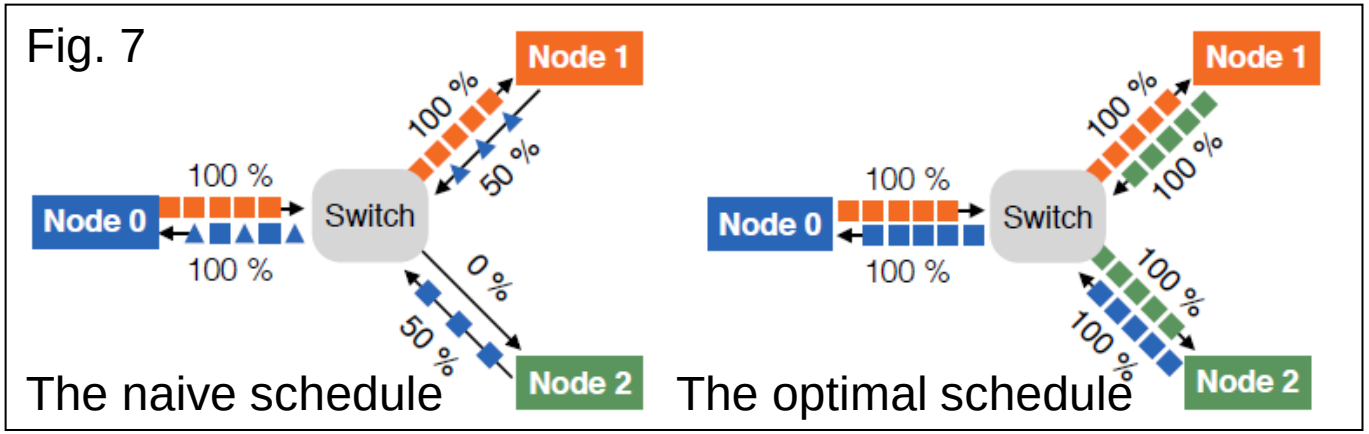
Fig. 6

目次

1. Exploiting Hardware Transactional Memory in Main-Memory Databases (DE140012)
2. Locality-Sensitive Operators for Parallel Main-Memory Database Clusters (DE140048)
3. Rethinking Main Memory OLTP Recovery (DE140247)
4. Optimal Hierarchical Layouts for Cache-Oblivious Search Trees (DE140646)

概要

- 背景：CPU性能がノード間の通信帯域を超えている
- 課題：クラスタ環境のJOIN処理における通信ネック
  - 大規模なインメモリDBのクラスタにおけるJOIN処理
  - ノード間は低帯域な通信



- 関連研究・キーワード
  - MapReduce, Fragment-Replicate, CloudRAMSort, ...

提案手法　1

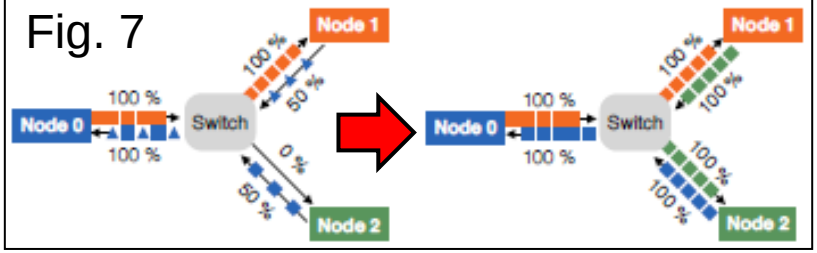
- 提案手法の効果
  - ノードへの最適なデータ配置によるデータ通信時間の削減
  - 通信スケジューリングによる通信帯域の効率化
  - パーティショニング指数によるデータのはずれ値を考慮
  - 選択的なブロードキャストによる通信帯域の効率化

提案手法 2

- 提案手法
  - 最適なパーティション方法
    - 同じJOINのキー値を同じパーティションとなるように分類する
  - パーティションの割り当て方法
    - ノード間の通信量が最小になるように割り当てる。
    - 式3：siは，ノードiのデータ送信量
    - 式4：riは，ノードiのデータ受信量

    これらの最小化が目的
  - 通信スケジューリング方法
    - スター型トポロジにおいて，
      リング型トポロジの様に通信する。

Fig. 7



提案手法 ３

- 提案手法
  - パーティションのブロードキャスト
    - ブロードキャストを考慮して式３，４を拡張
    - 下記の式が最小となるようにする。

$$w \geq s_i + s_i^B - \sum_{j=0}^{p-1} (y_j h_{ij} + z_j h_{ij}) \quad 0 \leq i < n$$

$$w \geq r_i + r_i^B \quad 0 \leq i < n$$

評価

- 実験環境
  - 4ノードのスター型クラスタ
    - ノード（Corei7 3770(4core, 3.4Ghz), 32GB），通信（**1Gb Eth**）
    - 最適解を導出するために，IBM CPLEX libraryを使用。

- 実験結果

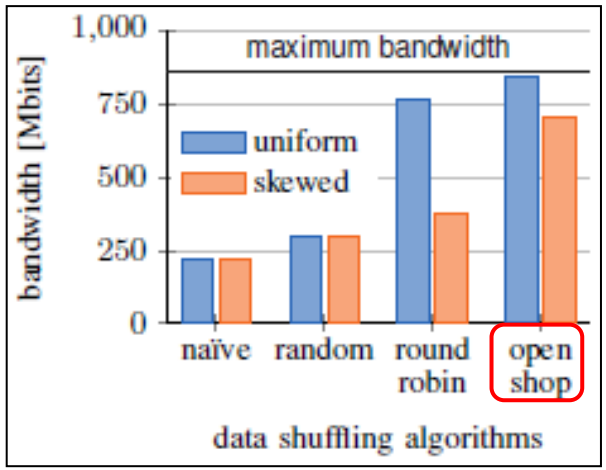


Fig.13. 通信帯域の利用率

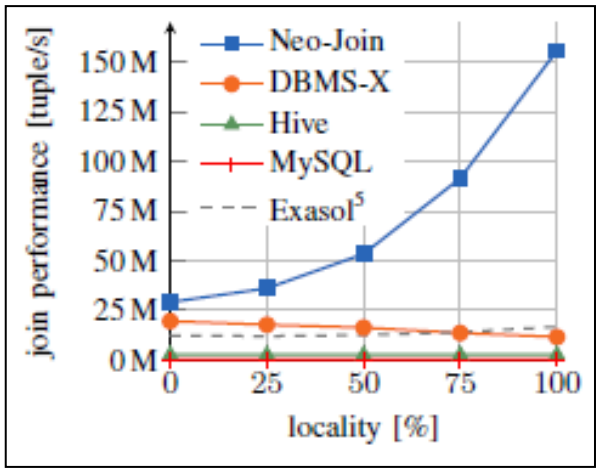


Fig.14. 局所性

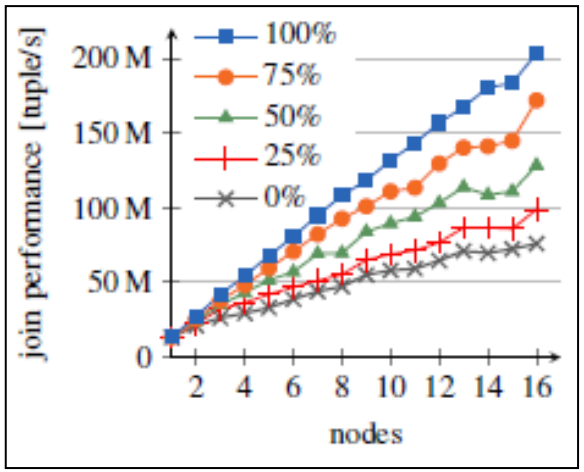


Fig.15. スケールアウト
*測定環境を変更

目次

1. Exploiting Hardware Transactional Memory in Main-Memory Databases (DE140012)
2. Locality-Sensitive Operators for Parallel Main-Memory Database Clusters (DE140048)
3. Rethinking Main Memory OLTP Recovery (DE140247)
4. Optimal Hierarchical Layouts for Cache-Oblivious Search Trees (DE140646)

概要

- 目的：OLTPの性能向上
- 課題：ログ処理ネックの改善
  - インメモリDB移行に伴い，
    ログ方式やログサイズが原因となるログ処理ネックが発生
- 関連研究・キーワード
  - ARIES Style Logging, 高頻度なチェックポイント [2]
    Logical Logging [19]

提案手法　1

- 提案手法の効果
  - Tx処理の情報をログとすることにより，ログ生成コスト・ログ情報量を削減（UPDATE情報の保持が不要）
  - ただし，障害回復時は，Tx処理を再実行しながら復元するため遅い（従来は差分（ログ）コピー程度）

提案手法 2

- 提案手法
  - ログ(Tx処理情報)を実行順に出力する。

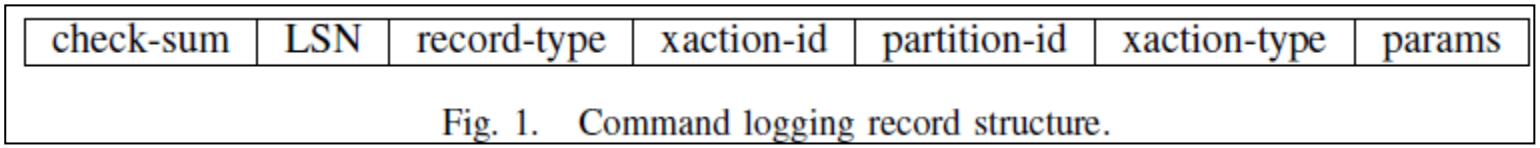


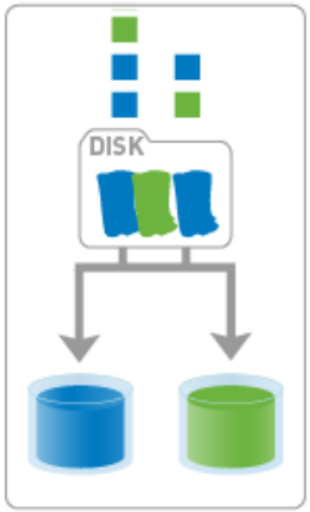
Fig. 1. Command logging record structure.

    - 乱数や時間などは，Tx.ID(タイムスタンプ)を用い再現性を得る
    - ACID維持のために同期ログ出力(グループコミット使用)

  - 障害回復は，チェックポイントまでを復元し，以後はログ(Tx処理情報)を再実行する。
    1. チェックポイントデータをメモリ上に再現(インデックス以外)
    2. インデックスを生成
    3. チェックポイント以後のログを再実行
       # クライアントへ未応答かつログ完了のTxも再実行

検証環境（VoltDB）

- 概要
  - インメモリDB
  - パーティションは，テーブルをキー値によって分割
  - パーティションを冗長化することで信頼性を向上
  - クエリがパーティション内に閉じるような分散処理（最適化）
  - ストアドプロシージャ（ワークロードの関数化）で実行



Command Logs

To protect data between snapshots, VoltDB logs transaction invocations to disk. VoltDB refers to this as the command log.

Command logs are also written at each server.

To recover, the snapshot is restored and the command log is replayed. Together snapshots and command logs create durable, replicated copies of the database across all servers.

提案手法は実装済み： https://voltdb.com/products/key-features/

評価

- 実験環境
  - Benchmark：Voter, TPC-C
  - 1ノード（2 Xeon (8core, 2.4Ghz), 24GB）

- 実験結果
  - 性能は1.5倍に向上。障害回復時間は1.5倍に低下。

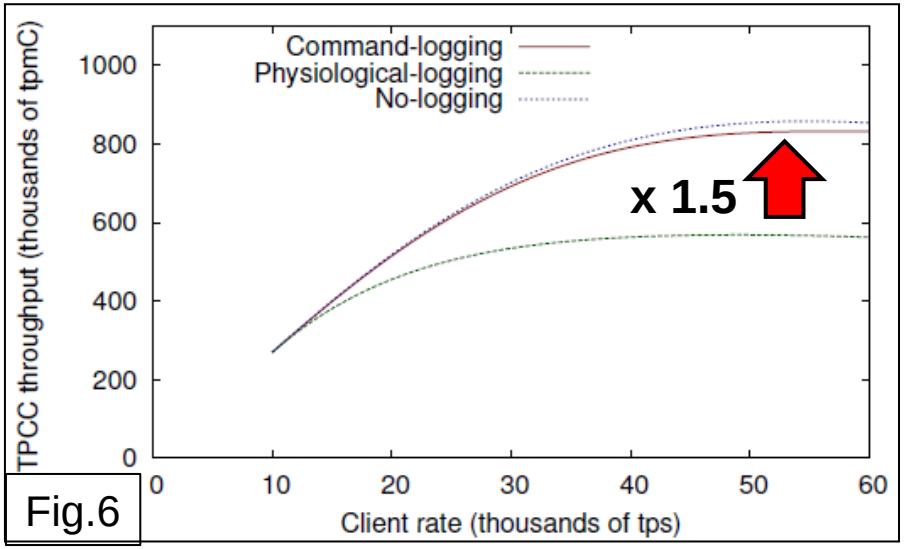


Fig.6

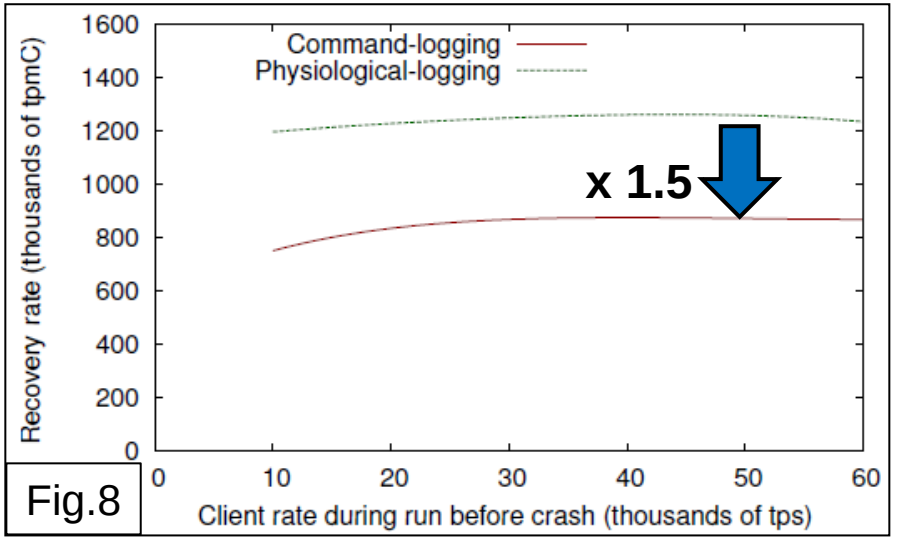


Fig.8