ACM SIGMOD 日本支部第44回支部大会

Nb-GCLOCK: A Non-blocking Buffer Management based on the Generalized CLOCK

産業技術総合研究所 情報技術研究部門
サービスウェア研究グループ
油井 誠

ICDE 2010での発表内容

Nb-GCLOCK: A Non-blocking Buffer Management based on the Generalized CLOCK

Makoto YUI[1], Jun MIYAZAKI[2], Shunsuke UEMURA[3] and Hayato YAMANA[4]

1 .Research fellow, JSPS (Japan Society for the Promotion of Science) / Visiting researcher, Waseda University
2. Nara Institute of Science and Technology
3. Nara Sangyo University
4. Waseda University / National Institute of Informatics

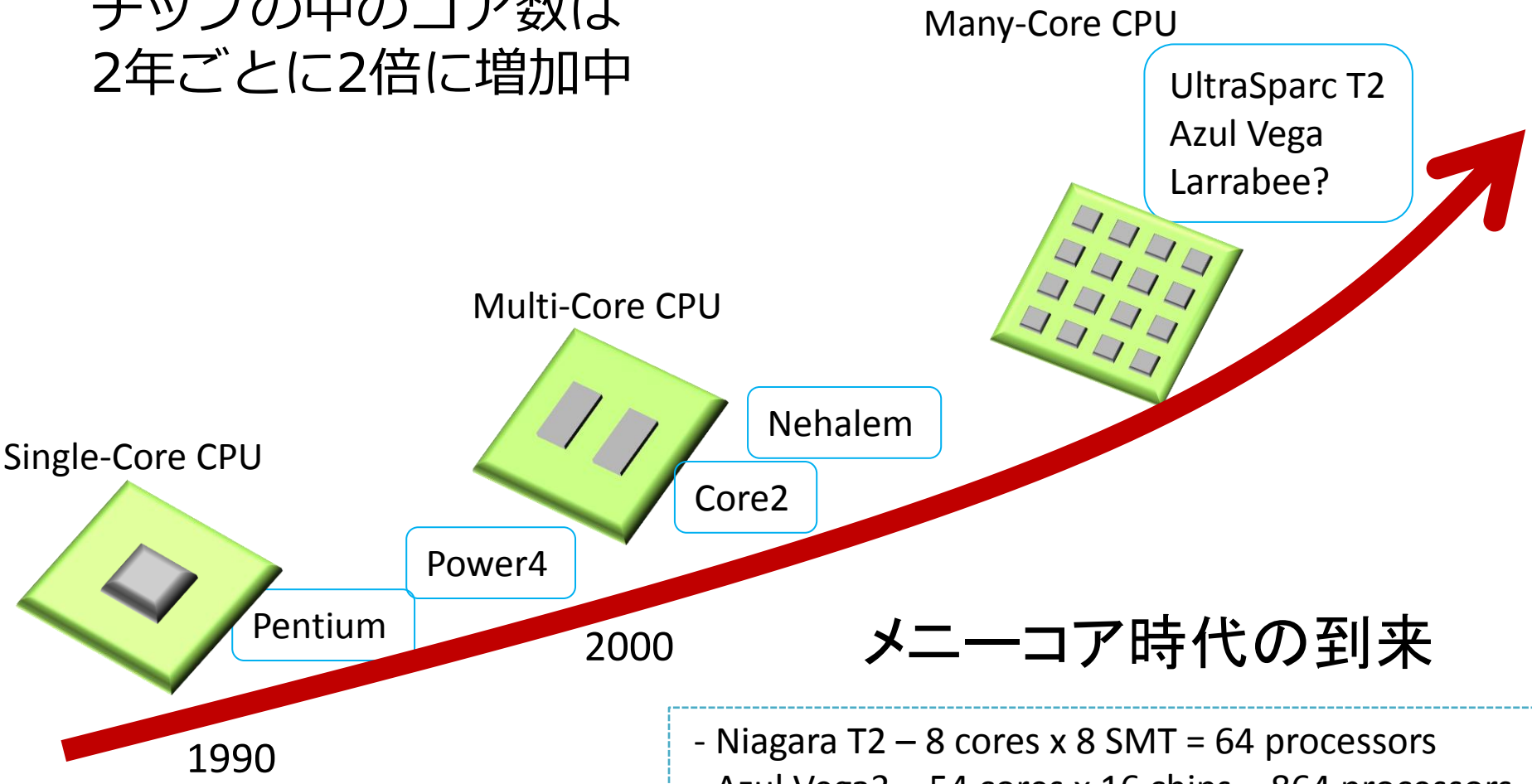
発表の構成

- 研究背景
- 提案手法
  - Non-Blocking同期手法
  - Nb-GCLOCK
- 評価実験
- 関連研究
- まとめ

- ICDE採択に至るまで

研究背景 – CPU開発のトレンド

チップの中のコア数は
2年ごとに2倍に増加中



メニーコア時代の到来

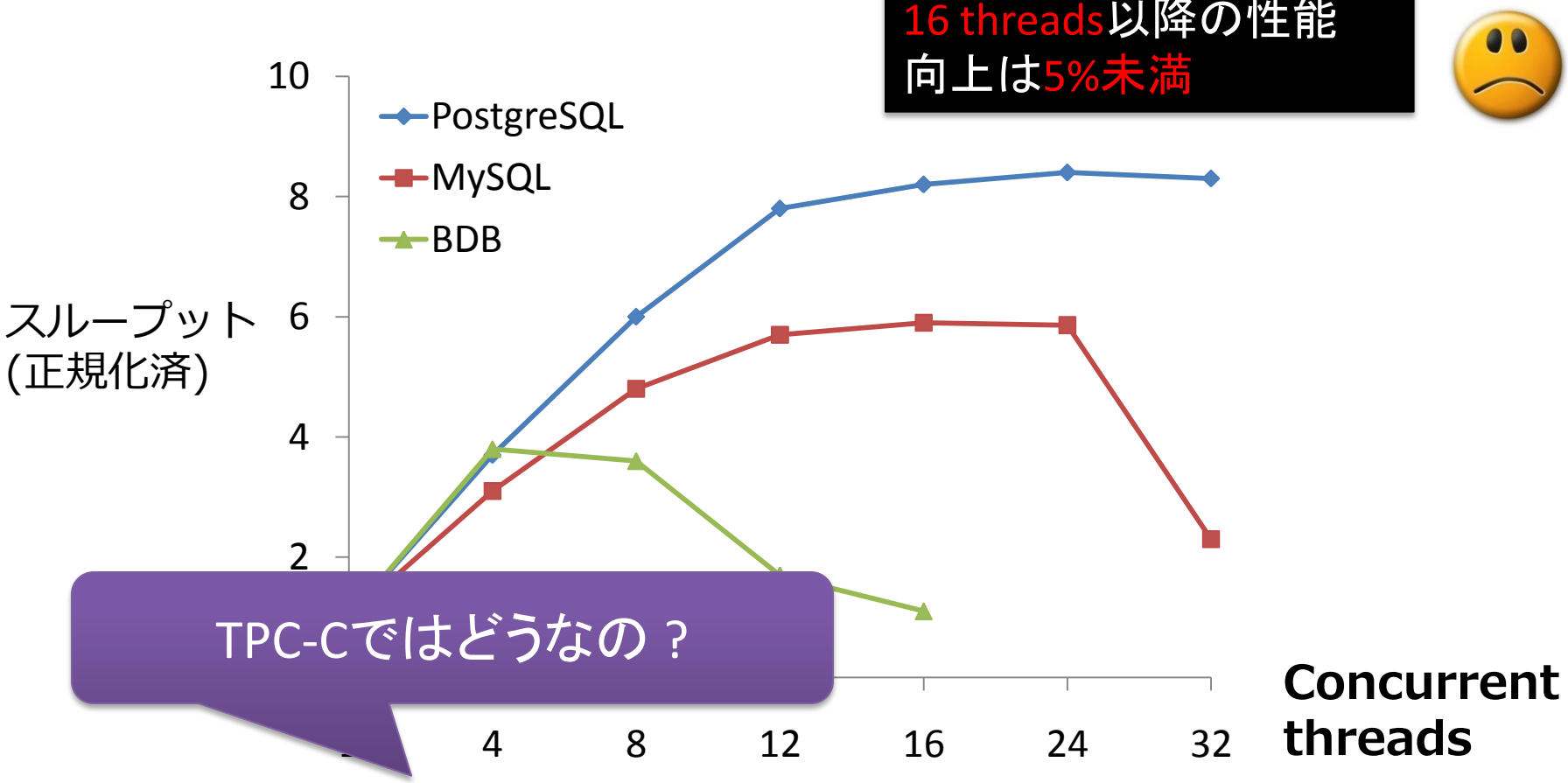
- Niagara T2 – 8 cores x 8 SMT = 64 processors
- Azul Vega3 – 54 cores x 16 chips = 864 processors

研究背景 – オープンソースRDBMSのCPUスケーラビリティ

オープンソースRDBMSはCPUスケーラビリティの問題に直面

Ryan Johnson et al., “Shore-MT: A Scalable Storage Manager for the Multicore Era”, In Proc. EDBT, 2009.

16 threads以降の性能向上は5%未満

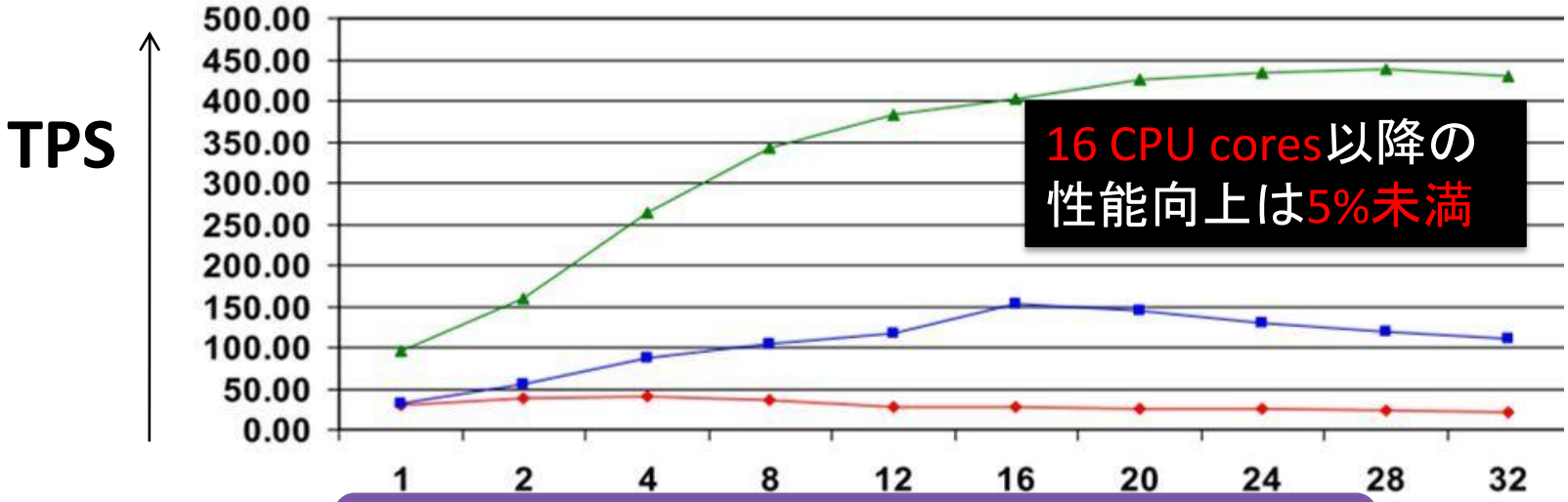


UltraSparc T1 (32 procs)でのマイクロベンチマーク

PostgreSQLのCPUスケーラビリティ

Unisysのハイエンドリナックス構成でのTPC-C ベンチマーク結果
（Xeon-SMP 32 CPUs, Memory 16GB, EMC RAID10 Storage)

Doug Tolbert, David Strong, Johney Tsai (Unisys),
“Scaling PostgreSQL on SMP Architectures”, PGCON 2007.



Version 8.2
Version 8.1
Version 8.0

Q. Postgres開発陣は何をしたか?

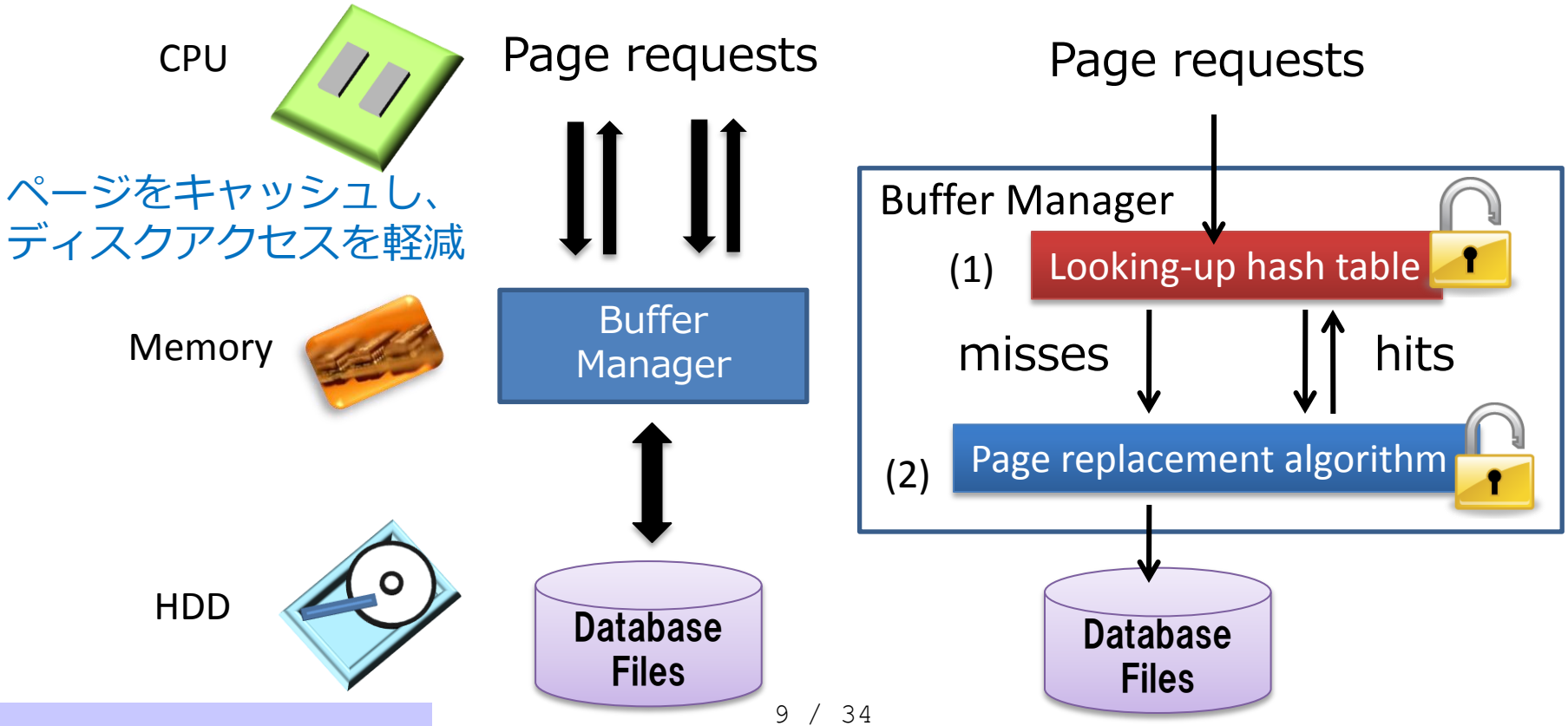
バッファ管理モジュールの
同期処理メカニズムを見直した

バッファ管理モジュールでの同期

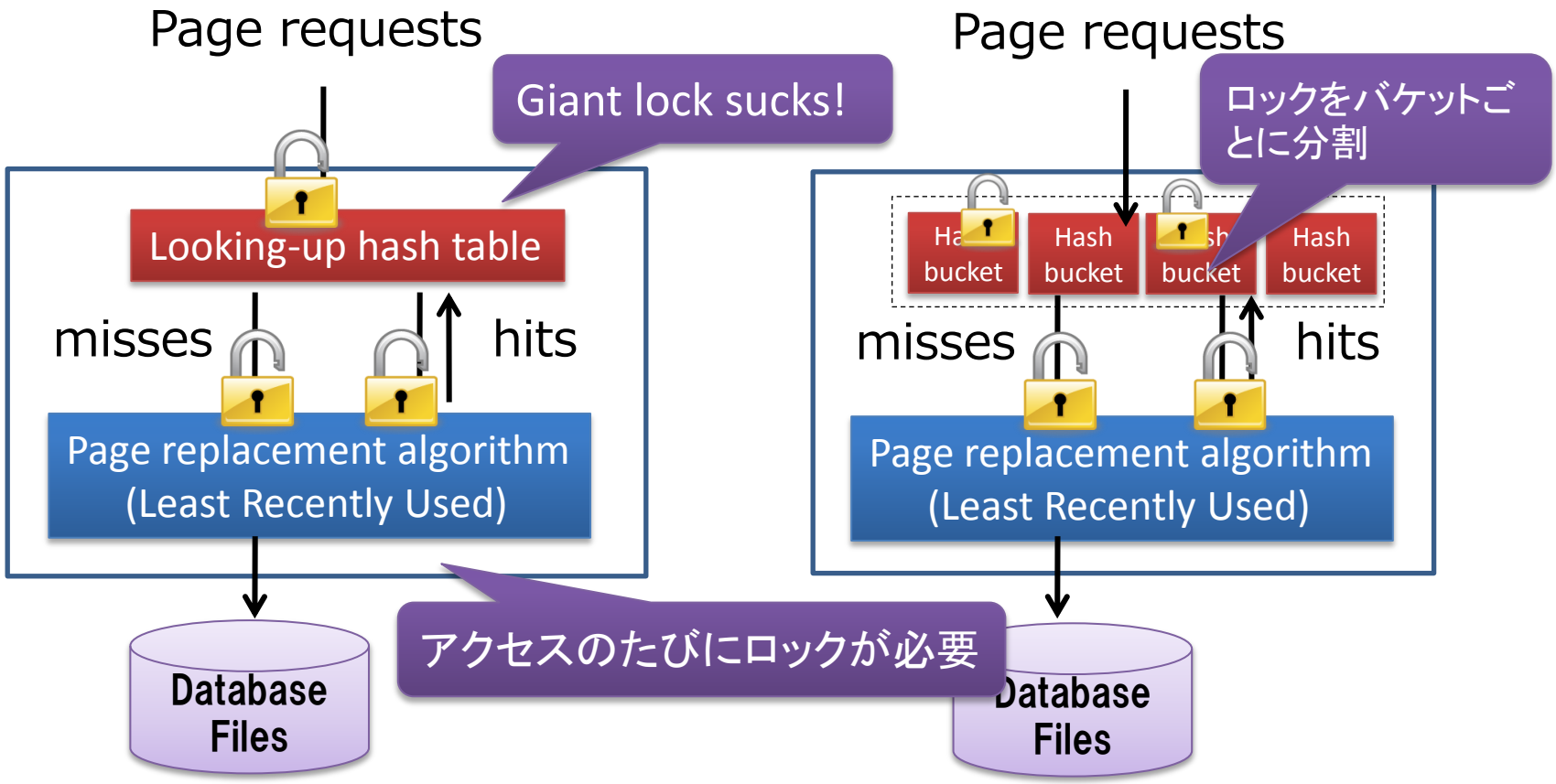
複数の経験的な研究によって、最大のボトルネックが次にあることが指摘されている

バッファ管理モジュールでの同期処理

[1] Ryan Johnson, Ippokratis Pandis, Anastassia Ailamaki:
“Critical Sections: Re-emerging Scalability Concerns for Database Storage Engines”, In Proc. DaMoN, 2008.
[2] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker:
OLTP Through the Looking Glass, and What We Found There, In Proc.SIGMOD, 2008.



ナイーブなバッファ管理手法



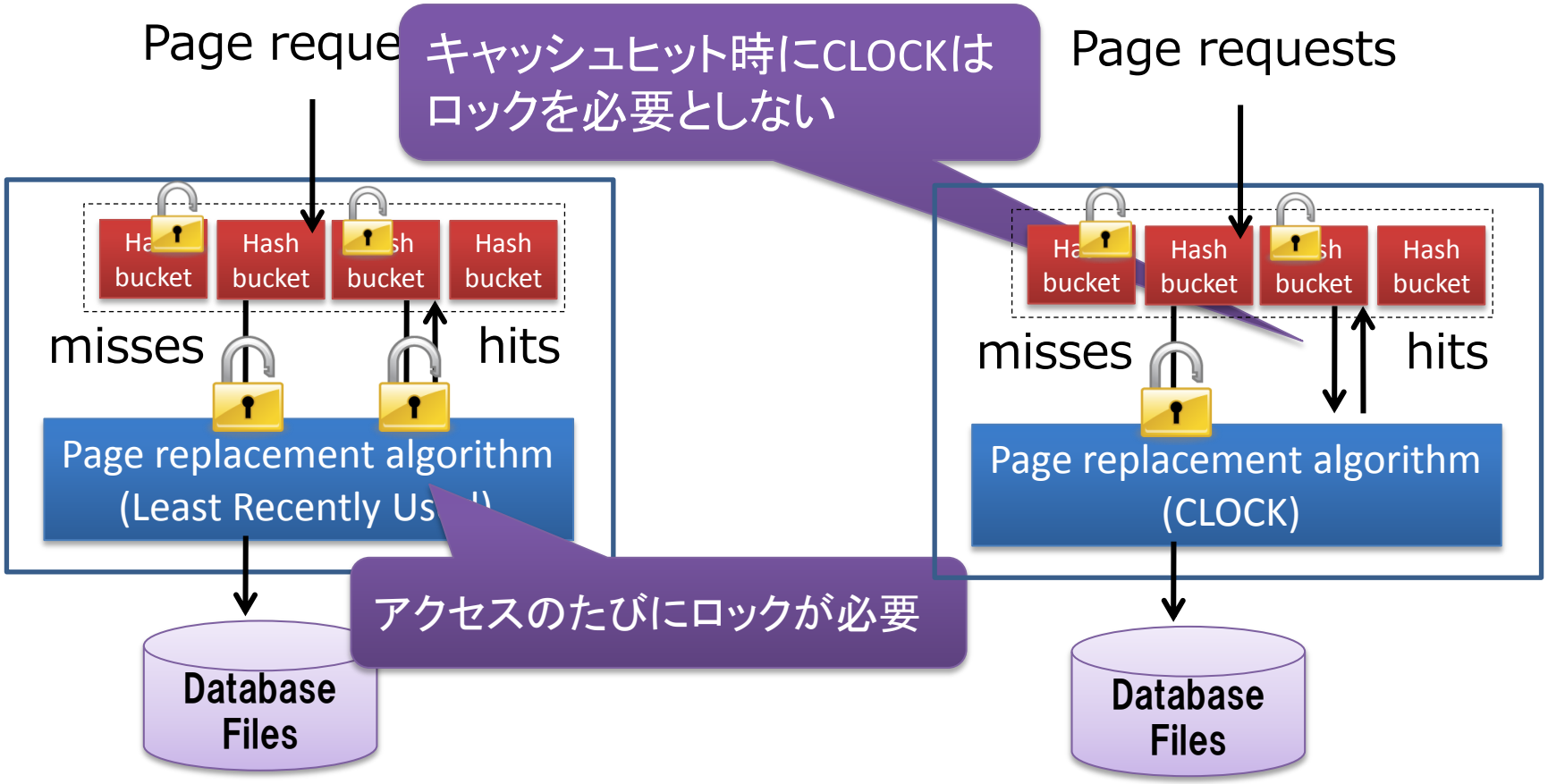
PostgreSQL 8.0

✓まったくスケールしない

PostgreSQL 8.1

✓8プロセッサまでスケール

少しナイーブなバッファ管理手法



PostgreSQL 8.1

✓ 8プロセッサまでスケール

PostgreSQL 8.2

✓16プロセッサまでスケール

発表の構成

- 研究背景
- 提案手法
  - Non-Blocking同期手法
  - Nb-GCLOCK
- 評価実験
- 関連研究
- まとめ

- ICDE採択に至るまで

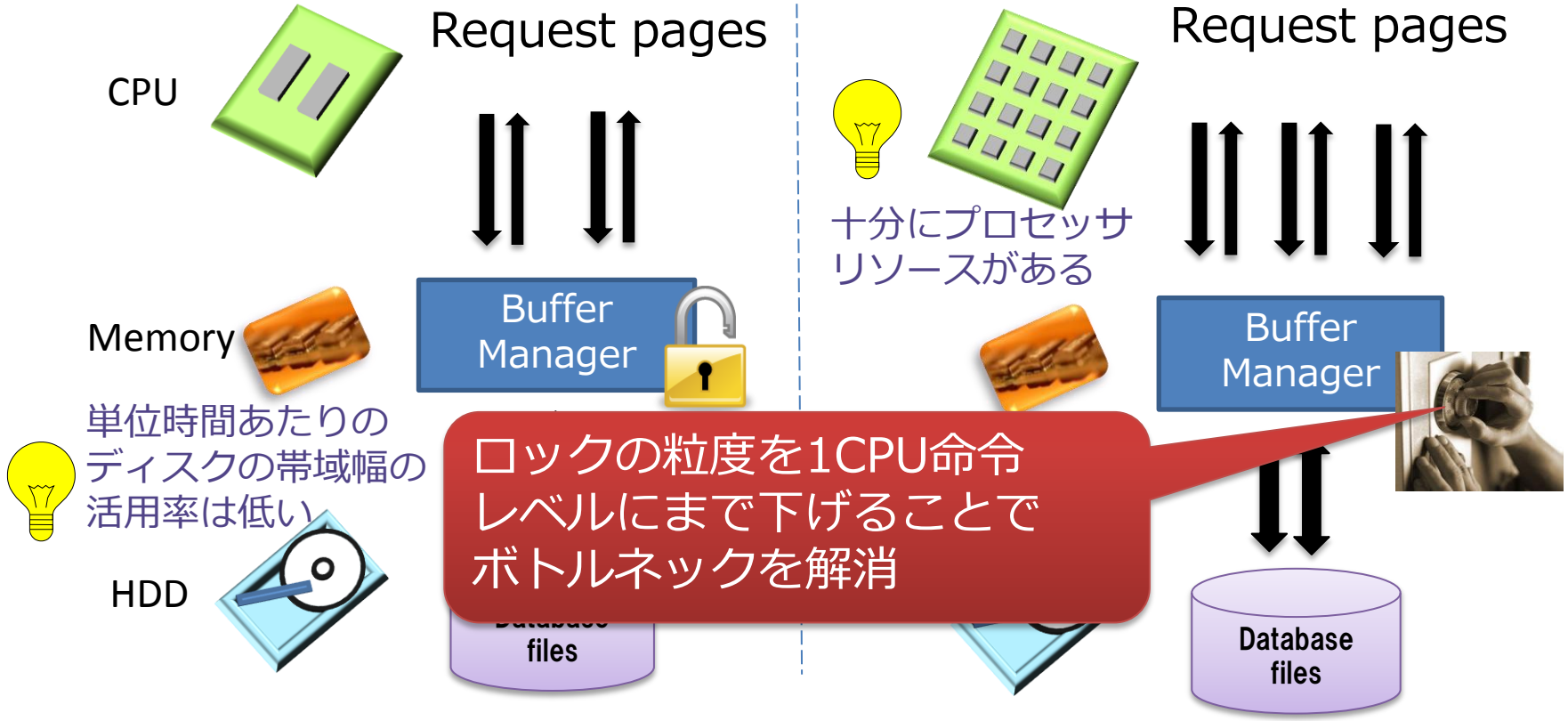
基本となるアイデア

既存のアプローチ

○ ディスクI/Osが減る
× ロックの競合が起きる

提案の楽観的アプローチ

△ ディスクI/Oは少し増える
○ ロックの競合が発生しない



ノンブロッキング同期

共有リソースへの並行アクセスを実現する上で
いかなるロックも獲得しない同期手法

- CPU命令を活用
  - CAS (compare-and-swap) cmpxchg on X86
  - メモリバリア

Blocking

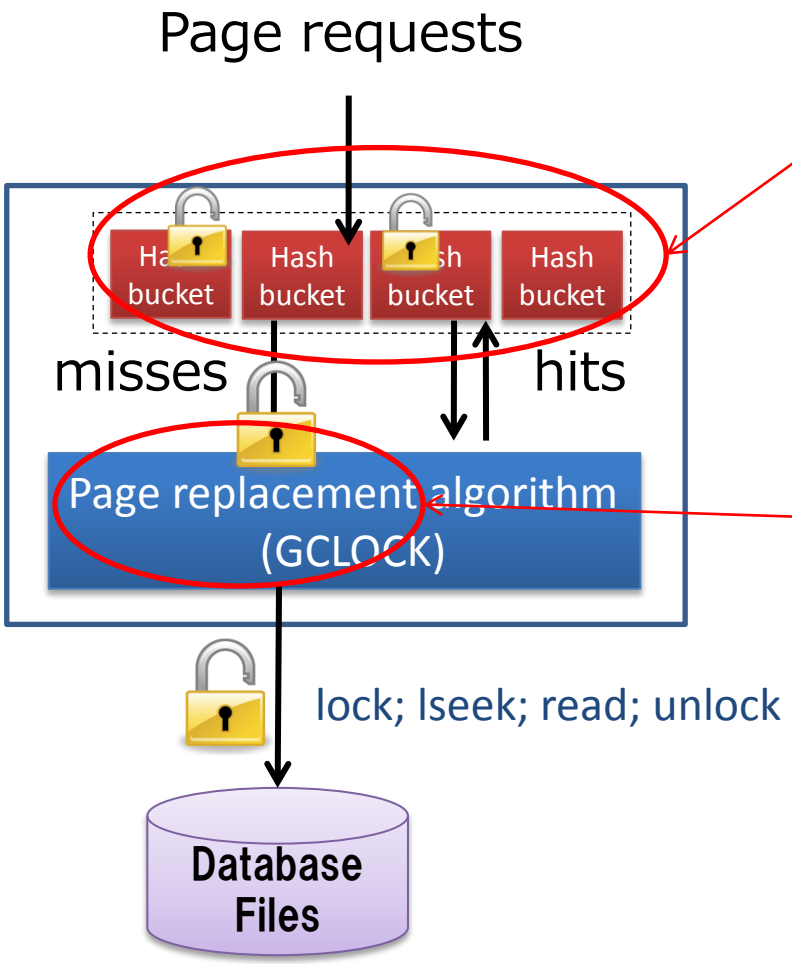
```
acquire_lock(lock);
counter++;
release_lock(lock);
```

Non-Blocking

```
int old;
do {
  old = *counter;
} while (!CAS(counter, old, old+1));
```

値がoldが等しいときに限り、
Counterの値をカウントアップ

バッファ管理をノンブロッキングにする

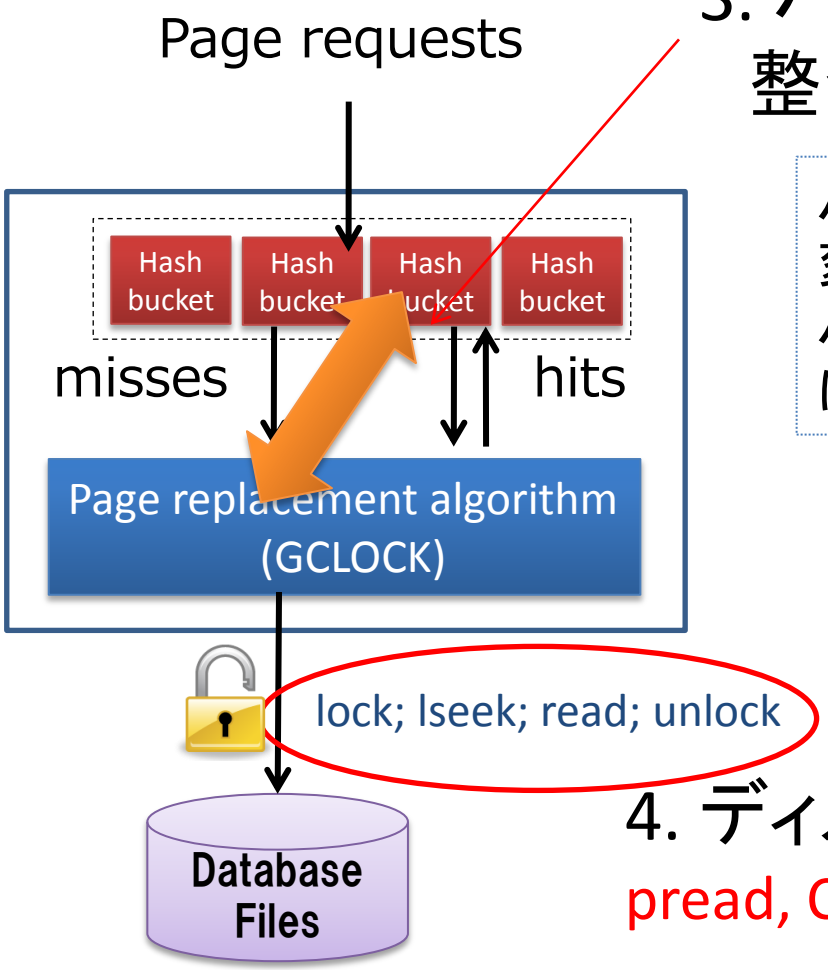


1. ロックフリーのハッシュテーブルを用いる

[1] Ori Shalev, Nir Shavit: Split-ordered lists: Lock-free extensible hash tables, J. ACM, Vol. 53, No. 3, pp. 379-405. 2006.
[2] Chris Purcell, Tim Harris: Non-blocking Hashtables with Open Addressing, Distributed Computing (DISC), pp. 108-121, 2005.

2. キャッシュミス時のロックを取り除く

バッファ管理をノンブロッキングにする



3. ハッシュ表とGCLOCKの整合性を保つ

バッファフレームへのページの割り当てを変えた直後、
ハッシュ表でそのバッファフレーム(value)は異なるページID(key)と結び付いている

Optimisticな振る舞いを許容するアルゴリズム（状態遷移機械）を設計

4. ディスクI/Oでのロックを取り除く

pread, CAS, memory barriersを活用

楽観的ページ読み込みと固定

提案システムはバッファフレームへのページ読み込み処理をpreadシステムコールとCASを利用した楽観的I/Oにより行う

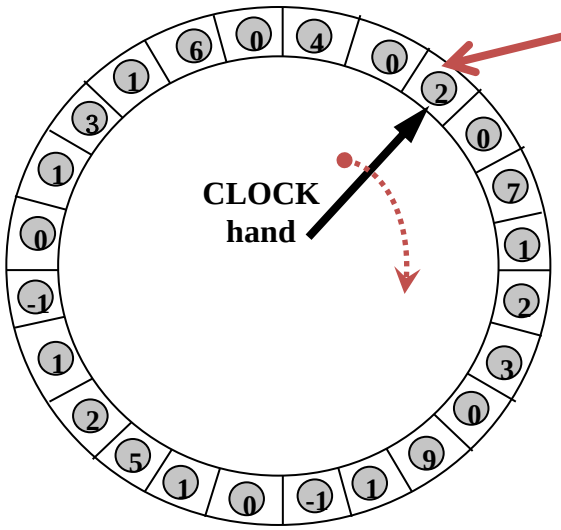
- **メモリフェンス**を張ってからページを取得　　3と6の呼び出し順序を保証
- **pread**によりディスクから要求ページを読み込む
- 指定されたスロットにページが割り当てられていなければupdateに書き換え(**compare-and-swap**)

```
1  Frame slot =
     PAGE_CACHE.fixEntry(pageId);
2  try {
3    V page = slot.volatileGetValue();
4    if(page == null) {
5      V update = read-in a page of the pageId from disk
6      slot.CASValue(update);
7    }
8    do application logic for the page
9  } finally {
10   slot.unpin();
11 }
```

冗長なDisk I/Oを許す

これまでの常識と異なる

Generalized CLOCK (GCLOCK)

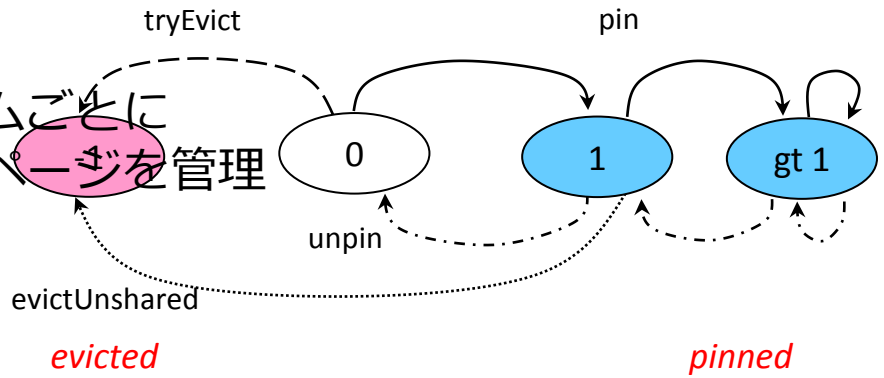


バッファフレームが管理する変数

- ✓ 参照カウント
- ✓ pinされている数
- ✓ evictされているか否か

原子的に管理する必要があり，短時間の
ロックをフレームごとに取得するのが一般的

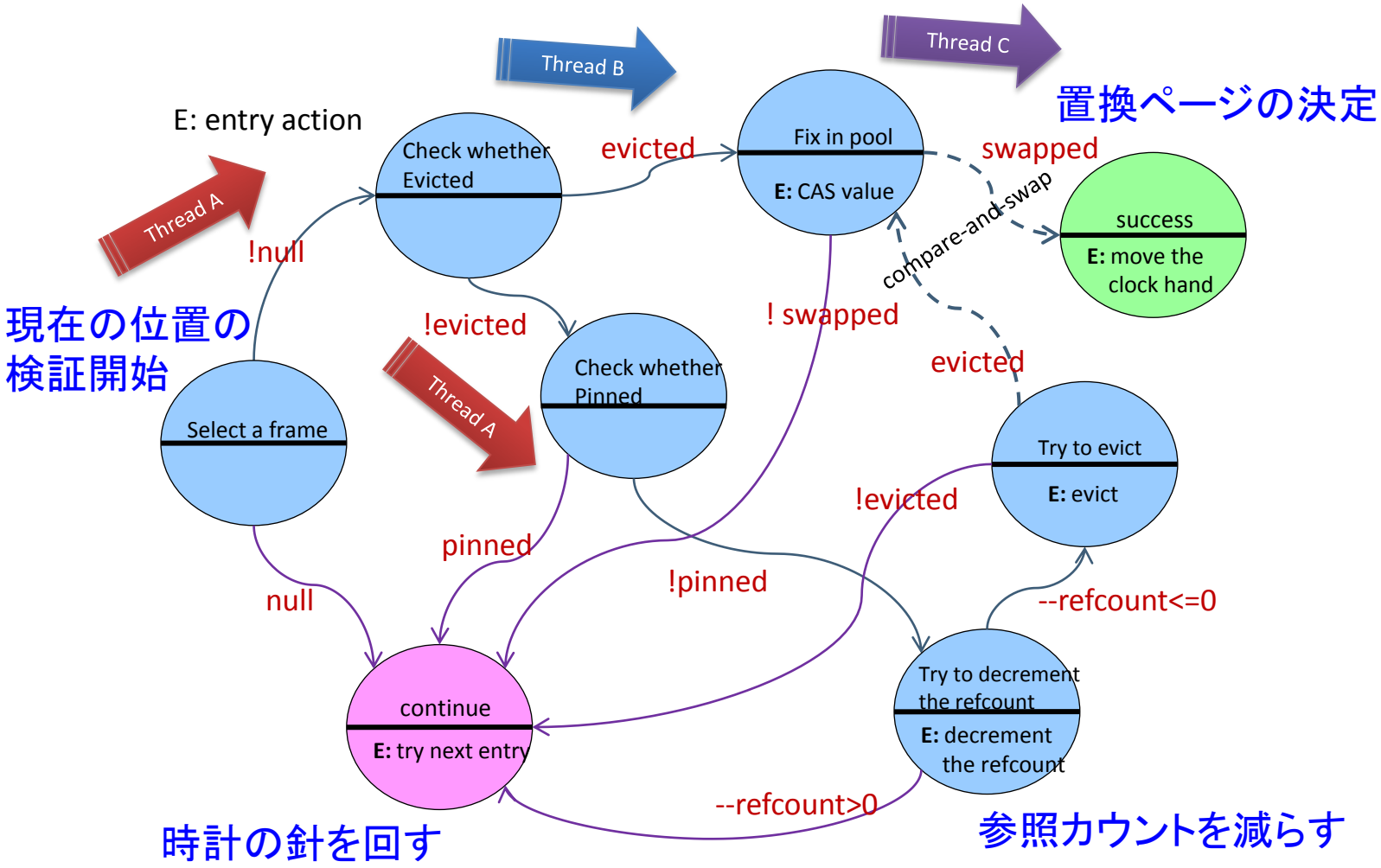
循環バッファでバッファフレームごとに
参照数を記録することで，置換ページを管理



提案手法は，値域を設定し，単一の変数(1 word)を共有することで
複数の変数の更新を同期基本命令によりロックなしに実現

バッファ管理の状態遷移機械を設計

共有データに対する複数のスレッドの読み書きがInconsistentな状態を作らないようにDFAベースに設計



発表の構成

- 研究背景
- 提案手法
  - Non-Blocking同期手法
  - Nb-GCLOCK
- 評価実験
- 関連研究
- まとめ

- ICDE採択に至るまで

実験設定

- ワークロード
  - ✓ Zipf 80/20 分布 (有名なpower law)

    2Qの例にならい20%のシーケンシャルスキャンを含む
  - ✓ データサイズは32GB
- 利用した計算機

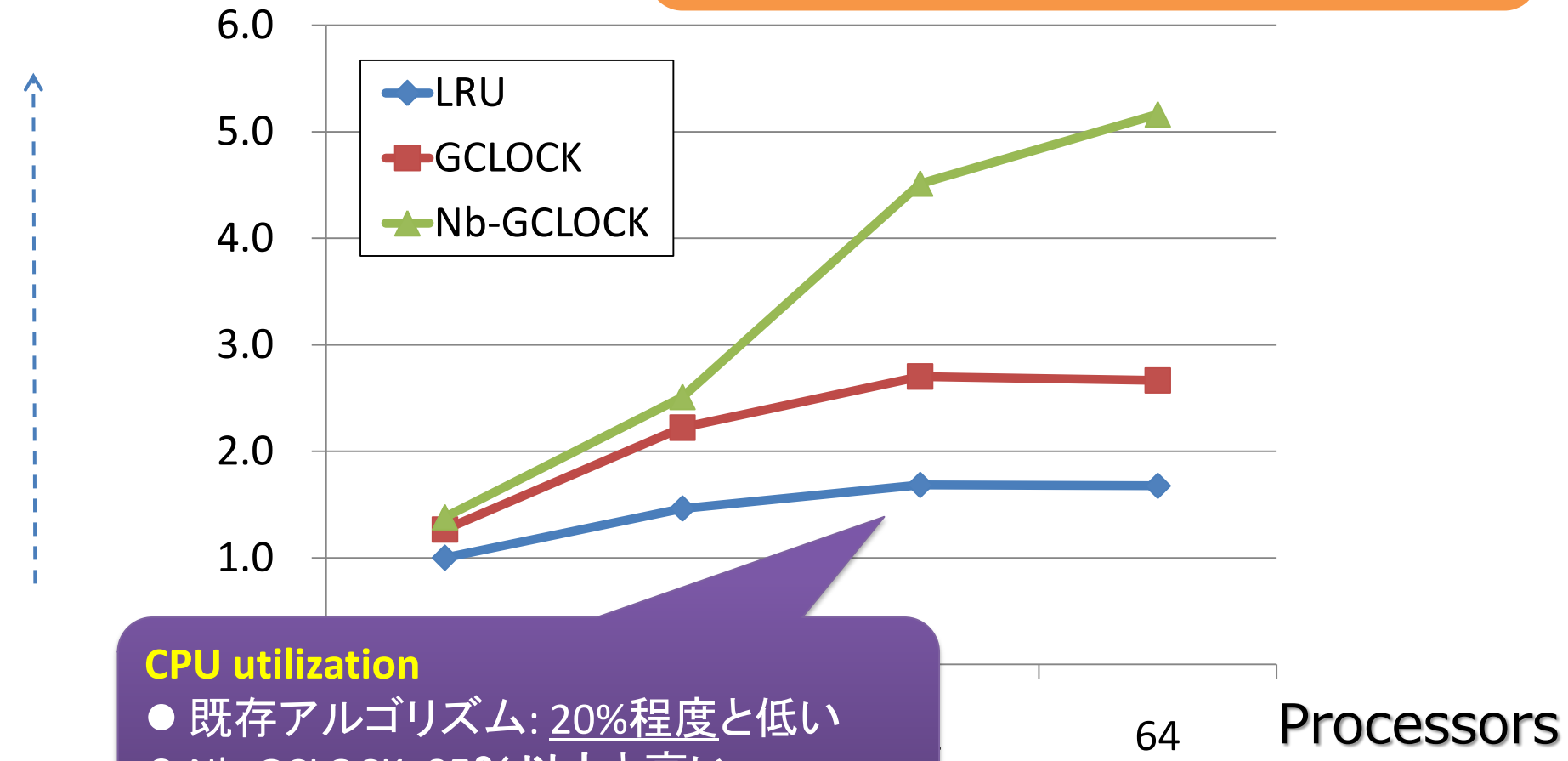
  UltraSPARC T2

Operating System	Solaris 10 8/07
Core (Threads/Core)	8 (8)
Processor frequency	1.2 GHz
Main memory	16GB
Disk	SAS (10000rpm)
L2 cache per core	4M

64 processors

キャッシュヒット率が6割弱の時の性能比較

CPU数が増えるほどに、CPUの消費時間に差が増えることが期待できる
→ よりスループットに差が開く

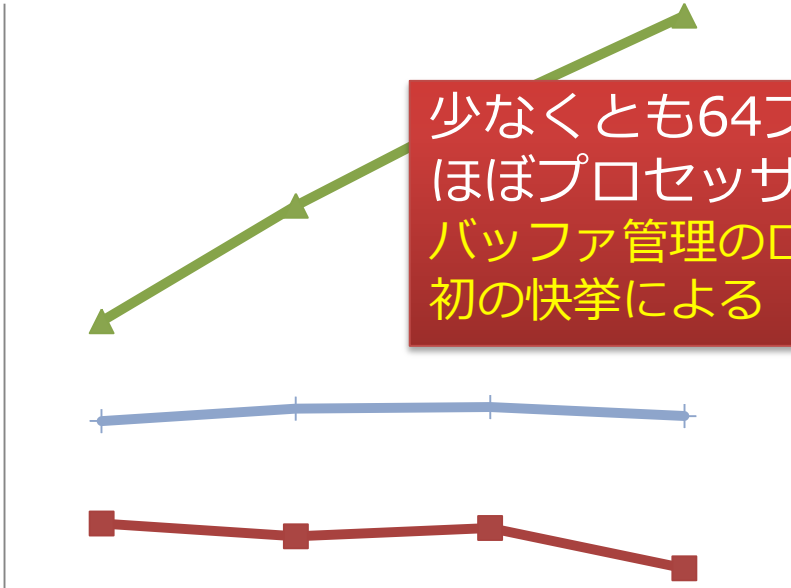


CPU utilization
- 既存アルゴリズム: 20%程度と低い
- Nb-GCLOCK: **95%以上**と高い

プロセッサ数に対するスループットの上限

全てのページがディスク上ではなくメモリにあるとして、I/O待ちを極小化し、各アルゴリズムレベルでのCPUスケーラビリティの限界を見る

Throughput (log scale)



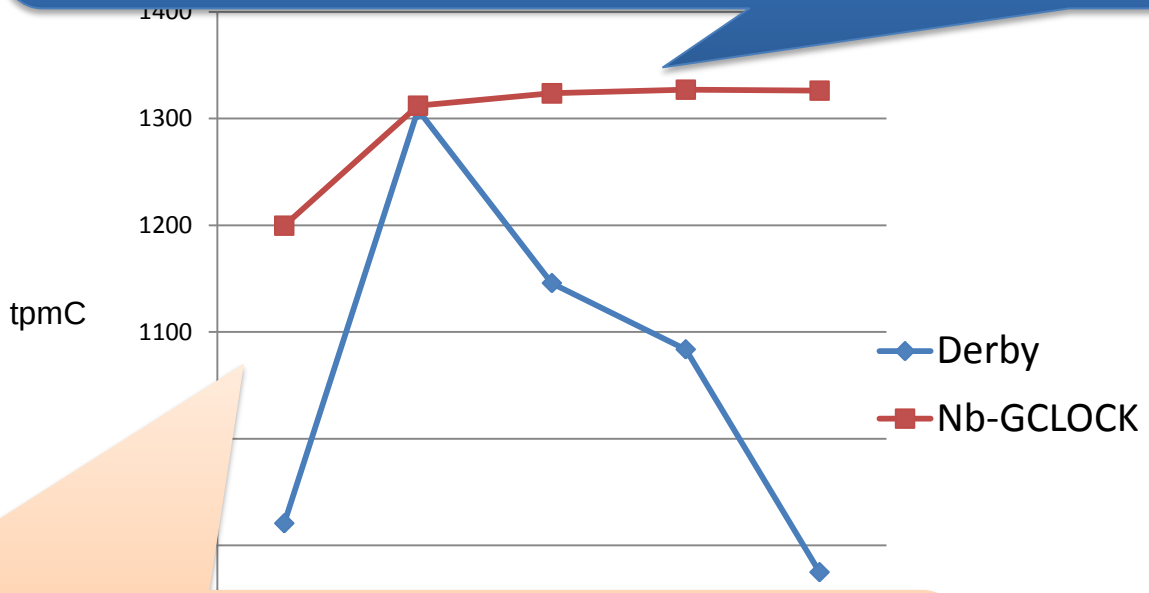
少なくとも64プロセッサまで
ほぼプロセッサ数に対し線形の性能
バッファ管理のロックを完全に取り除く
初の快挙による

	8 (1)	16 (2)	32 (4)	64 (8)
2Q	890992	819975	866009	662782
GCLOCK	1758605	1912000	1931268	1817748
Nb-GCLOCK	3409819	7331722	14245524	25834449

Processors (cores)

Apache DerbyでTPC-C評価

そもそもバッファ管理モジュールへのスループットが
B+木のルートページのラッチで低下
→ ロック競合の少ないB+木が必要 (OLFIT[1]等)



Derbyがもともと採用していたCLOCKでは
バッファ管理によってスループットの低下を確認
→Nb-GCLOCKはスループットを落とさないことを確認

[1] Sang K. Cha et al: Cache-Conscious Concurrency Control of Main-Memory Indexes on Shared-Memory Multiprocessor Systems, Proc. VLDB, pp. 181-190, 2001.
[2] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, Michael Stonebraker: OLTP through the looking glass, and what we found there, Proc. SIGMOD, pp. 981-992, 2008.

発表の構成

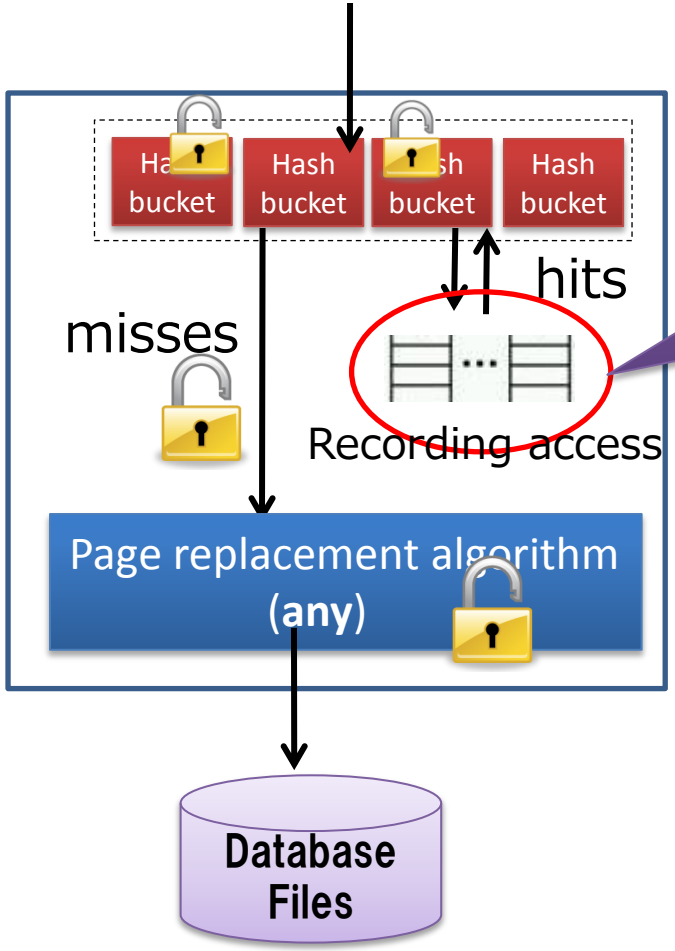
- 研究背景
- 提案手法
  - Non-Blocking同期手法
  - Nb-GCLOCK
- 評価実験
- 関連研究
- まとめ

- ICDE採択に至るまで

Bp-wrapper

Xiaoning Ding, Song Jiang, and Xiaodong Zhang:
Bp-Wrapper: A System Framework Making Any Replacement Algorithms (Almost) Lock Contention Free, Proc. ICDE, 2009.

バッチ処理とプリフェッチによって
バッファヒット時のロック競合を削減する



物理的な仕事(バッファ置換リストの調整)を遅延し、
論理的に仕事が終わったものとして直ぐに結果を返す

業界用語でLazy synchronizationと呼ばれる

Pros.

- どんなページ置換アルゴリズムにも適用可能

Cons.

- CLOCK系のアルゴリズムには効果がない
  CLOCKはページヒット時にロックを必要としないため
- キャッシュミスはバッチ処理を招く
  長いロック保持時間の発生がより多くの競合を招く

まとめ

メニーコア時代のDBの問題提起と現実的な解決策を提示

オープンソースDBMSや商用DBMSが採用している従来のバッファ管理手法には，
16プロセッサ付近にCPU数に対するスケーラビリティの限界があることを示した

- ロックフリーのGCLOCKページ置換アルゴリズム Nb-GCLOCKを提案
  - ✓ 64プロセッサまでほぼ線形の性能
    既存手法は16プロセッサまでしかスケールしない
  - ✓ ノンブロッキング同期を初めてDBのバッファ管理に紹介
  - ✓ pread, CASそしてmemory barriersを利用した楽観的ディスクI/O

- 線形化可能性（Linearizability）とロックフリー（lock-freedom）であることを論文中で証明
  - ✓ ロックフリーであるため、ある(certain)スループットが保証される

発表の構成

- 研究背景
- 提案手法
  - Non-Blocking同期手法
  - Nb-GCLOCK
- 評価実験
- 関連研究
- まとめ

- ICDE採択に至るまで

発想～採択に至るまでの道のり（発想編）

データ構造とアルゴリズム
が大好き

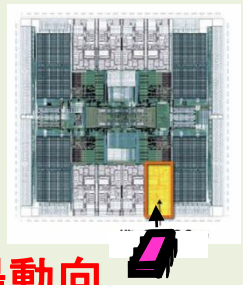
嗜好性

他分野からの知識習得

他分野に
アンテナを張っ
ておくと良い

バッファ管理が好き
各種既存アルゴリズムを趣味で実装

メニーコアCPUが
一般市場に

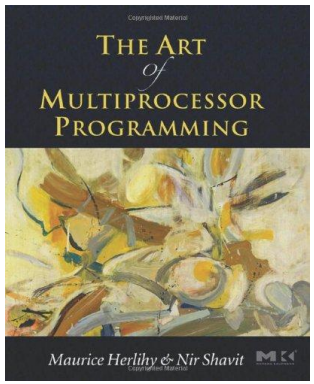


市場動向

ノンブロッキングのデータ構造
とアルゴリズムを知る

ACM Symposium on Parallel Algorithms and Architectures (SPAA)
ACM Symposium on Principles of Distributed Computing (PODC)

先行利益



Prepared mind

マルチプロセッサ上での既存
アルゴリズムの問題に直面

Bp-wrapperは
DB/OS分野だ
けの文献参照

バッファ管理の着眼点がバッファヒット率
を上げることしかないことに気づく

世界で誰も取り組んでいない問題に
効率の良い経路でアプローチできた

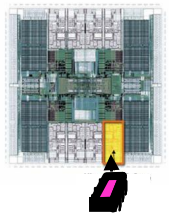
発想の転換

ノンブロッキングのバッファ管理アルゴリズムの開発

発想～採択に至るまでの道のり（論文執筆編）

ノンブロッキングのバッファ管理アルゴリズムの開発

Niagara T2 (64 Threads)で
評価し、手ごたえを得る



ICDE 2009に投稿 ⇒ Reject

自分で弱いと思っているところは、つつかれることを認識

- **Weak reject (Novelty unclear)**
  - 私みたいなOutside-readerに対する配慮に欠ける
  - (オープンソース)DBMSに適用して、オフィシャルのベンチマーク評価せよ
  - 無待機のハッシュ表を使うことがdominantじゃないの？
- **Accept (Novel)**
  - 重要な問題に対する解だ
  - Proofはケアフルだし、評価実験は私に湧いた疑問点に答えるものだ
- **Weak reject (With some new ideas)**
  - 2Q, LRU, Nb-GCLOCKを比較していたがGCLOCKとも比較すべき
  - プロセッサのアイドル時間を示したら？というアドバイス
  - 評価データがメモリに収まっているので大きく

In spite of these flaws the paper is good first attempt for addressing processor scalability problem that current database systems face.

Confidence=Highで建設的な意見をくれた。細かいgrammarミスまで指摘

発想～採択に至るまでの道のり（論文執筆編）

ICDE 2009に投稿 ⇒ Reject

- TPC-Cでの評価はほんとはした方が良いよなぁ…
- ロックフリーの証明はもっと厳格にすべきだよなぁ…

査読結果だけに落ちた原因が全てあるとは考えない（つまらん…などは査読結果には見えない）

- 博論執筆で暫くお休み
- 学位を採ったあとは国際会議に集中
- WebDBfでの査読結果も反映

トップカンファレンスに採択されるために労力を割く
- Apache Derbyに移植
- TPC-Cで評価
- ～切直前にpublishされた競合手法を実装して比較
- Lock-freeの証明追加

指摘に関しては殆どが自分でも思い当たりのある指摘 → 自分を信じてやれることはすべてやるしかない

やれるところまでやったので自分的にも満足

ICDE 2010に投稿 ⇒ Accept

Author Feedback

Accept, Weak Accept/Marginal, Acceptといった反応

Nb-GCLOCK is clearly a promising algorithm and needs to be published.

特に致命的なコメントはなかったが
Marginal気味のレビューに注意深く対応

基本的にReview内容が誤りでもそれを導いた論文の書き方が悪いので低姿勢で臨む　⇒　基本的に反論はしない

それでも意見が異なるところは根拠を明確にして
回答した上で、ミスリーディングさせたところを訂正する

I thank the authors for their thoughtful replies to the 3 reviews.

論文を書く上で気をつけたこと

- 一人はいる否定的な査読者に落とされないようにする
  査読者が論文を不採録にする理由を消していく
  - 意見が分かれそうな導入部分やセンテンスになるべく参照を入れる
    主張が相いれないケースへの対応として、**主張に客観性**を持たせる
  - 評価実験、関連論文が足りないなどの下らない(落としやすい)理由で落とされないようする

    少なくとも良い査読が貰えるように…
    下らない理由で落とされるのは論文に非がある

- 有効範囲、限界を明らかにする

  Q. 64プロセッサ以上ではどうなるか理論的に明らかにせよ

[37]にあるように、プロセッサ数が大きく変わると同じアーキテクチャでは済まない(SPAAとかでも実機で評価している)。

今回、64プロセッサまでではあるが、ブロッキングアルゴリズムに比べてX倍の性能が得られることを明らかにしたのは大きな成果である。
少なくとも一般的にプロセッサ数が上がる程、ノンブロッキングアルゴリズムはブロッキングアルゴリズムより巧く機能する。　貢献内容を明確に

論文を書く上で気をつけたこと

- 関連研究について
  - 必要なところに必要な参照を入れられるようにする

    （著名会議への）参照が少ない論文は、それだけで印象が悪い
    自分が査読者ならば不信感を抱く

    ちょっとでも読んだ論文はタグをつけて管理している
    http://www.citeulike.org/user/myui （2010/05/27現在860本）

  - 防衛的引用

    関連する著名なグループの論文は引用しておく
    （査読者になる可能性もある）

    今回の場合、AilamakiとKenneth A. Rossのところ

    査読者の攻撃材料になりそうな話題は、先回りして、知っていることを明示しておく。関連研究に入れておく。

    今回の場合、Transactional MemoryやOSの話題etc…