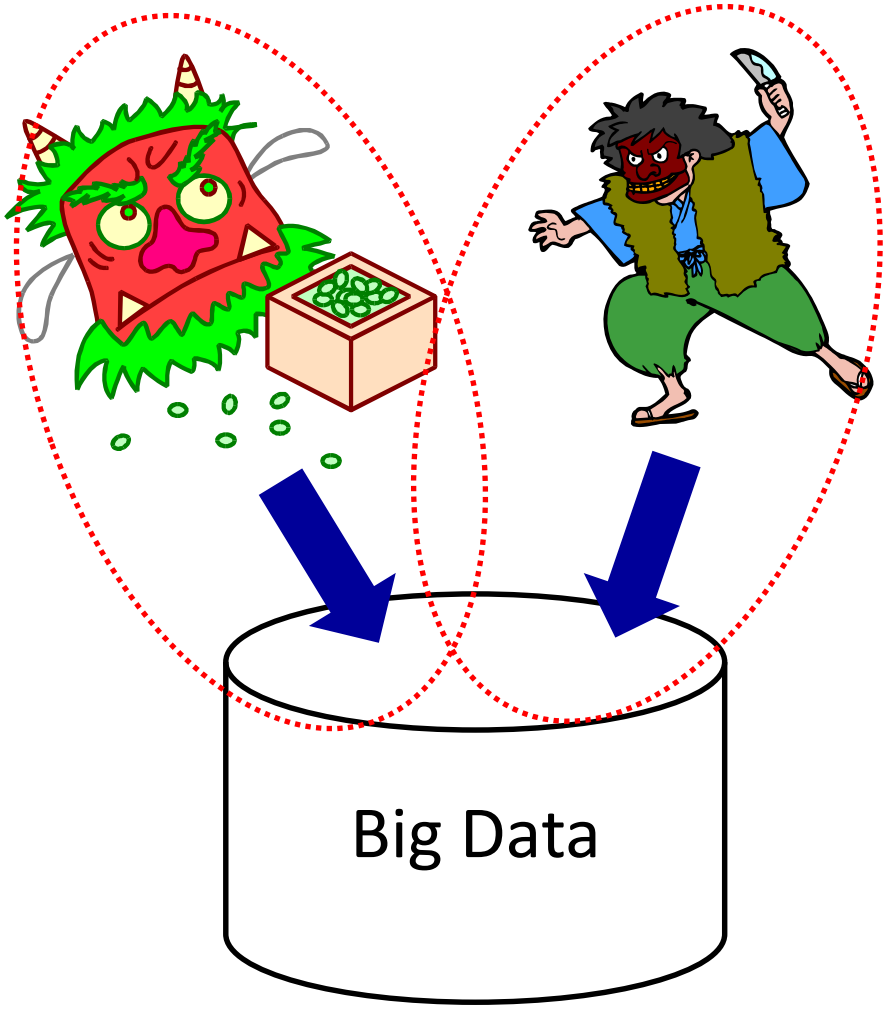
トランザクション処理技術

2020年10月27日

慶應義塾大学環境情報学部
川島英之

Transaction Processing

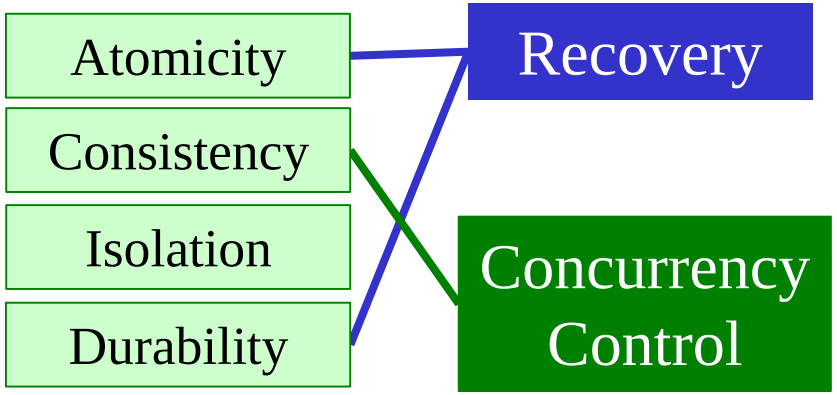


Can you access DB simultaneously?

研究目的

世界最速のトランザクション処理技術

- トランザクション
  - Concurrency Control
  - Recovery



- アプローチ
  - ビッグアプリとの**co-design**
  - ハードを活用する**re-design**
    - メニーコア、NVRAM

振込と引出

最初10000，+1000, -1000,最後は10000?

Begin
Read(口座, x)
x := x + 1000

Write(口座, x)
Commit

バイト先

Begin
Read(口座, y)
y := y - 1000
Write(口座, y)
Commit

自分

近代的並行性制御法

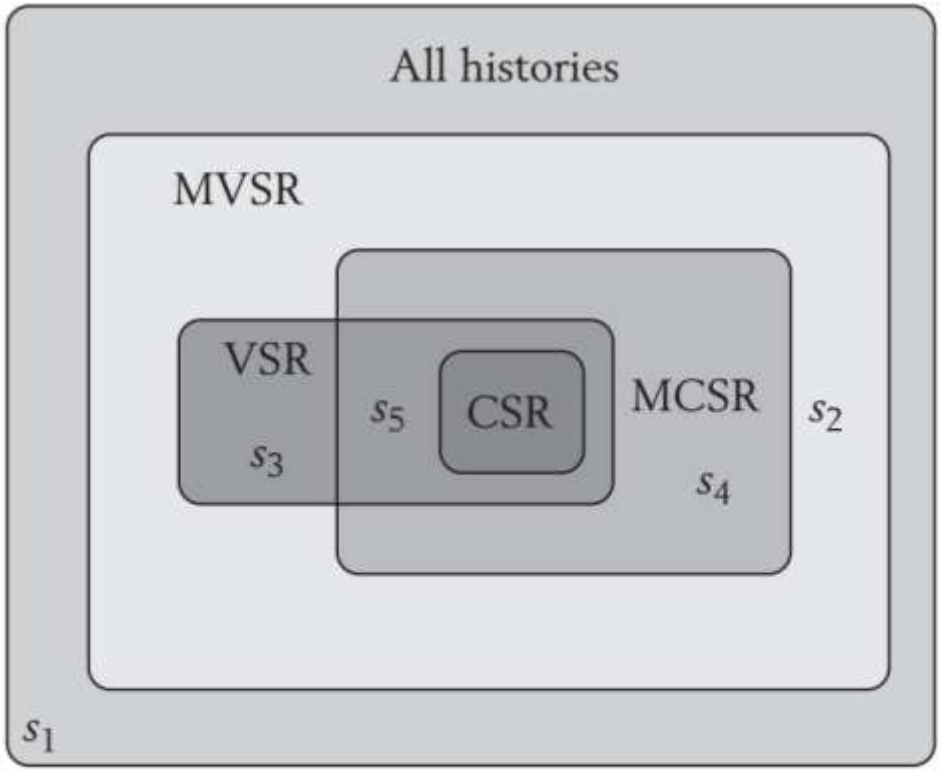
Method	Year	Conference	Features
CICADA	2017	SIGMOD	Optimistic Multi-Version
MOCC	2016	VLDB	Optimistic Pessimistic
TicToc	2016	SIGMOD	Optimistic
ERMIA	2016	SIGMOD	Multi-Version
Silo	2013	SOSP	Optimistic
SI	1995	SIGMOD	Multi-Version
2 Phase Lock	1976	Comm. ACM	Pessimistic

2つの謎

- 本当に速いのか？
  - CC性能は理論（スケジューリング空間）だけでは決まらない。実装が重要（索引で性能激変）
  - 実験で異なるプラットフォームを利用？
- なぜ速いのか？
  - 自分にとって有利な条件で実験してないか？
  - 様々な手法があるが、何が性能を決めているのか？

網羅的に分析可能なプラットフォームCCBench

Takayuki Tanabe, Takashi Hoshino, Hideyuki Kawashima, Osamu Tatebe: An Analysis of Concurrency Control Protocols for In-Memory Database with CCBench. PVLDB, 2020.



$$s_1 = r_1(x)r_2(x)w_1(x)w_2(x)c_1c_2$$
$$s_2 = w_1(x)c_1r_2(x)r_3(y)w_3(x)w_2(y)c_2c_3$$
$$s_3 = w_1(x)c_1r_2(x)r_3(y)w_3(x)w_2(y)c_2c_3w_4(x)c_4$$
$$s_4 = r_1(x)w_1(x)r_2(x)r_2(y)w_2(y)r_1(y)w_1(y)c_1c_2$$
$$s_5 = r_1(x)w_1(x)r_2(x)w_2(y)c_2w_1(y)w_3(y)c_1c_3$$

Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery (The Morgan Kaufmann Series in Data Management Systems), Weikum, Vossen

本当に速いのか？(MOCC)

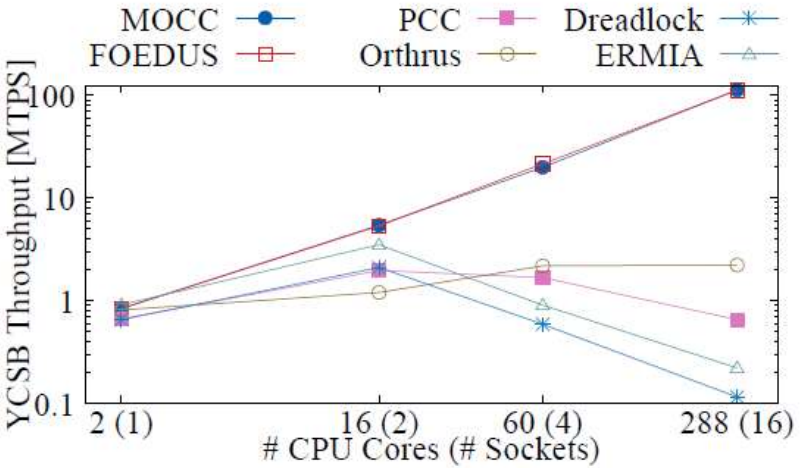
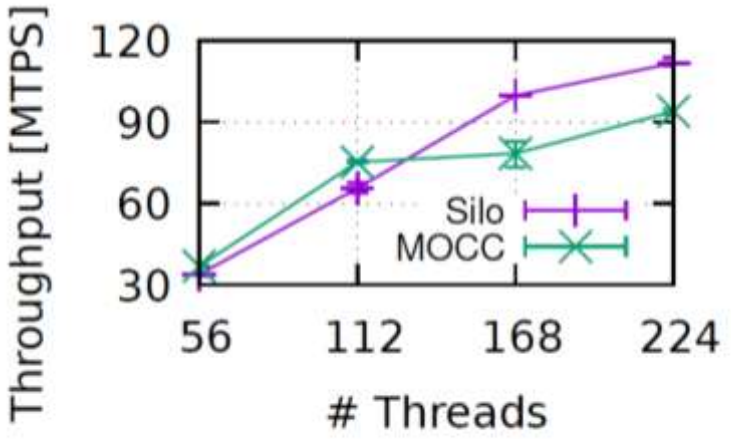


Figure 6: Throughput of a read-only YCSB workload with high contention and no conflict on four machines with different scales. MOCC adds no overhead to FOEDUS (OCC), performing orders of magnitude faster than the other CC schemes.

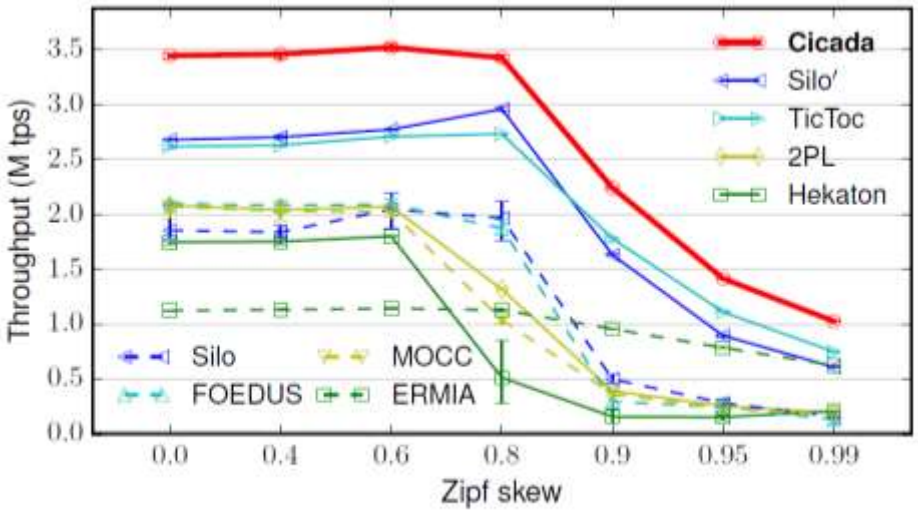


(a) Read Only Workload.

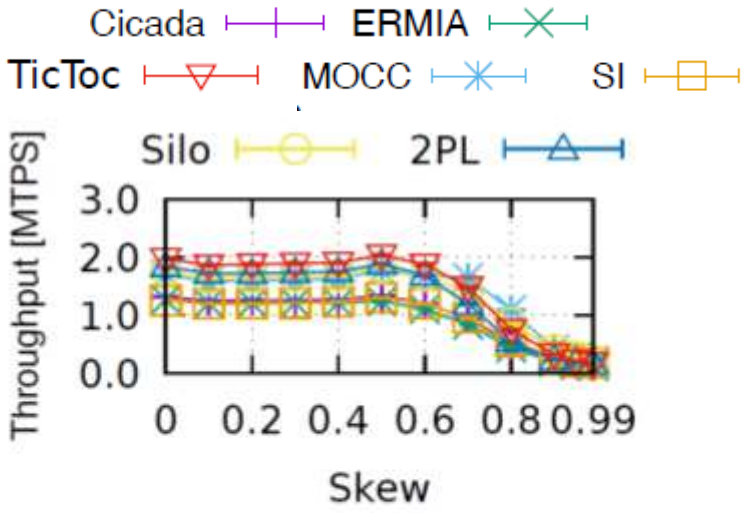
概ね consistent

T. Wang and H. Kimura. Mostly-Optimistic Concurrency Control for Highly Contended Dynamic Workloads on a Thousand Cores. PVLDB, 10(2):49–60, 2016.

本当に速いのか？(Cicada)



(b) Write-intensive, 28 threads.



(b) Write-Intensive. 28 threads. 16 requests/trans, 50% RMW - 50% pure reads

理由：複数システムの利用 (DBx1000, Original)

Lim, M. Kaminsky, and D. G. Andersen. Cicada: Dependably fast multi-core in-memory transactions. In SIGMOD Conf., pages 21-35, 2017.

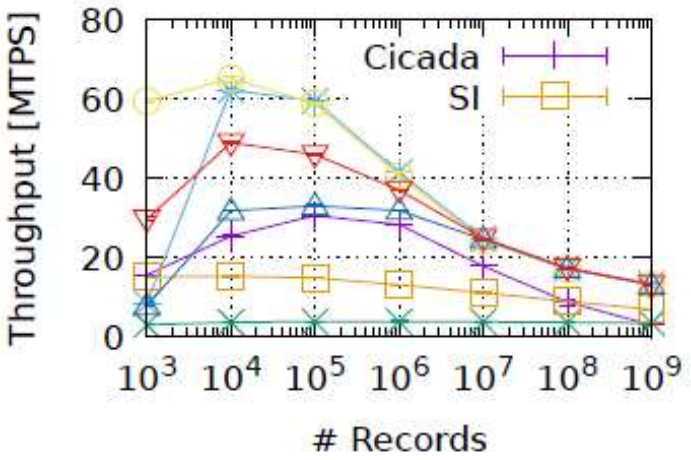
なぜ速いのか？

224並列環境での分析

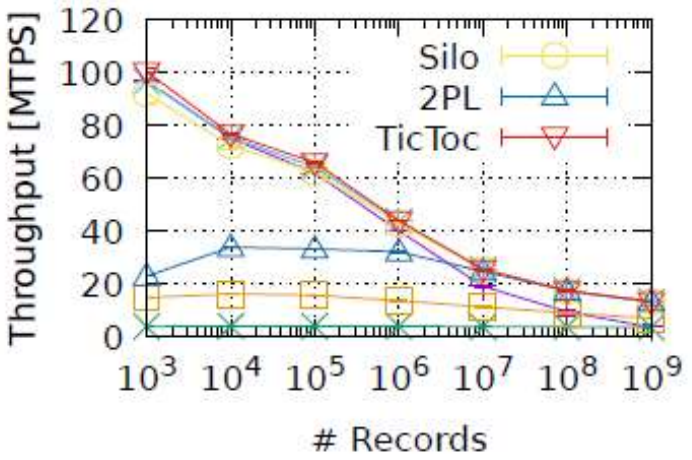
Table 2: Analyzed parameter sets. (α) Cardinality (from 10^3 to 10^9 records); (β) Cardinality (10^3 or 10^6 records); (γ) Read ratio (from 0% to 100%); (δ) Transaction size (from 10 to 100 operations); (ϵ) Payload size (from 4 to 1000 bytes); (ζ) Skew (from 0.6 to 0.99).

	Cache		Delay		Version
Figure Number	F5	F7	F9	F10	F11
Skew	0	0	0.8	0.9	ζ
Cardinality	α	β	10^8	10^8	10^8
Payload (byte)	4	4	4	ϵ	4
Xact size	10	10	δ	10	10
Read ratio (%)	0,5	γ	50,95	50	50,95
Thread count	Always 224 except from reproduction				
Read modify write	Always off except from reproduction				

Cache
(1) Cache-Line-Conflicts

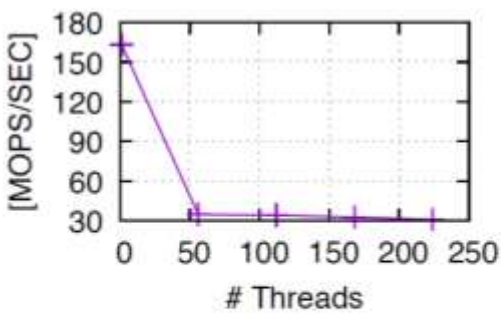


(a) YCSB-B

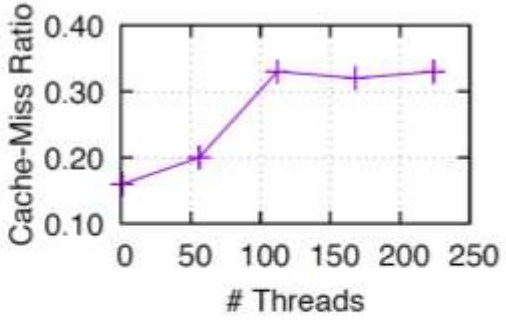


(b) YCSB-C

Figure 5: Varying cardinality: payload 4 I



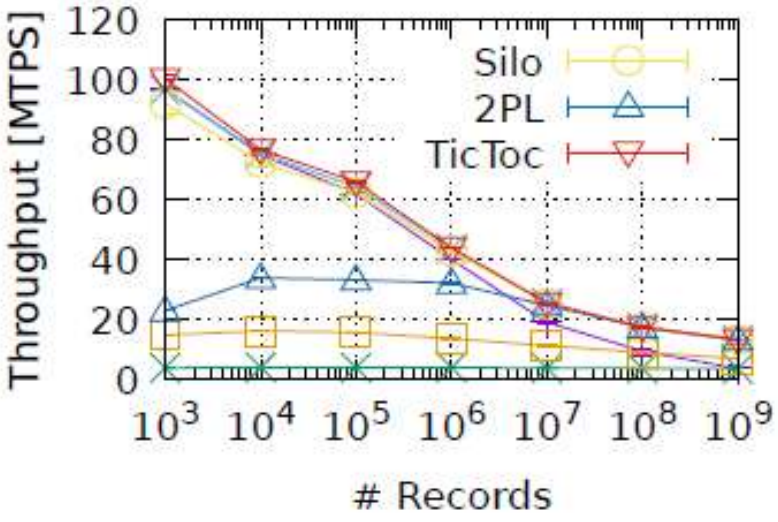
(a) Throughput



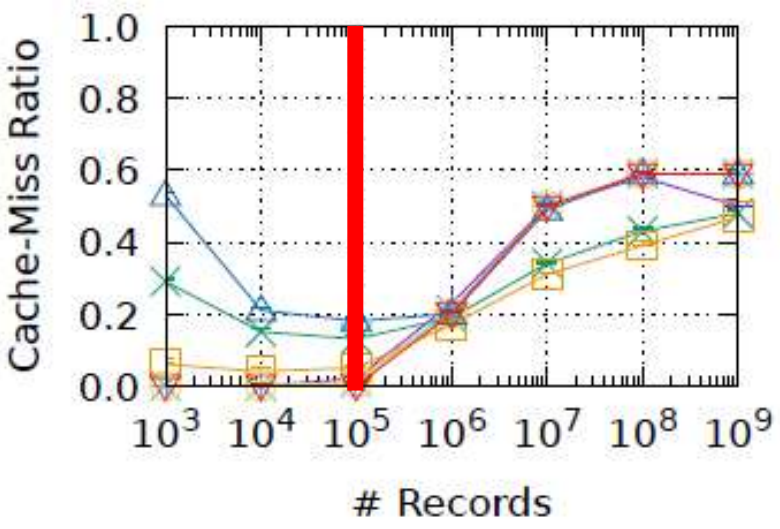
(b) Cache miss ratio

Figure 6: Scalability of fetch_add

Cache
(2) Cache Replacement



(b) YCSB-C



(d) YCSB-C

1. Silo, TicToc, Cicadaのヘッダは64B. 64B x 10^5 < 38.5 MB. これを超えて性能劣化。
2. それ以下でも性能劣化？レコード数増加に伴うL1/L2ミスの増加。

Delay
高競合時には並行性削減

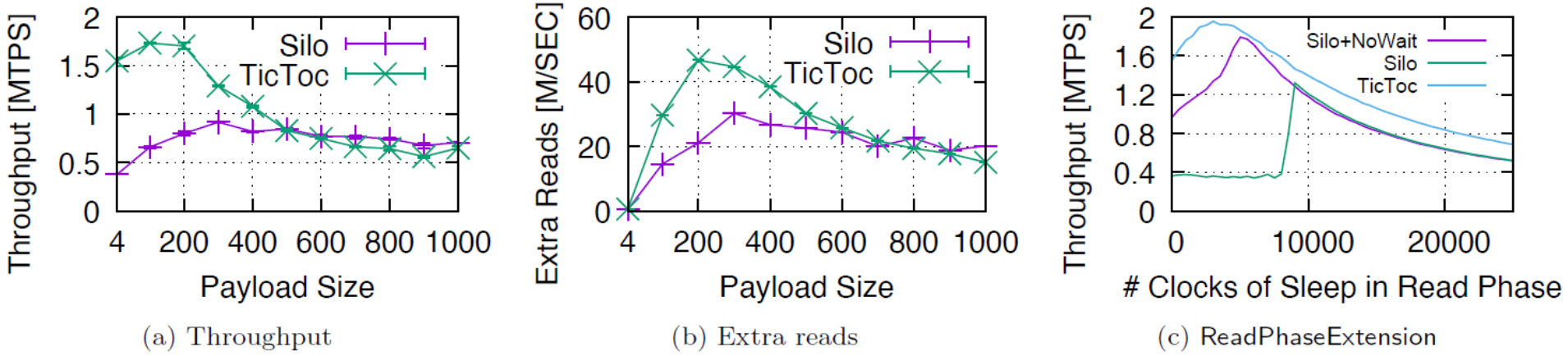


Figure 10: Effect of delay provided by extra read: YCSB-A, 100M records, 224 threads, skew 0.9, 10 ops/trans.

1. Payloadが大きくなれば性能が落ちる筈
2. しかし上がっている？なぜ？
3. 理由は余計な競合が減るから。強制sleepで確かに性能向上。
4. Silo + NoWaitで性能大幅改善。
5. Read Phase Extension

Version Lifetime Management

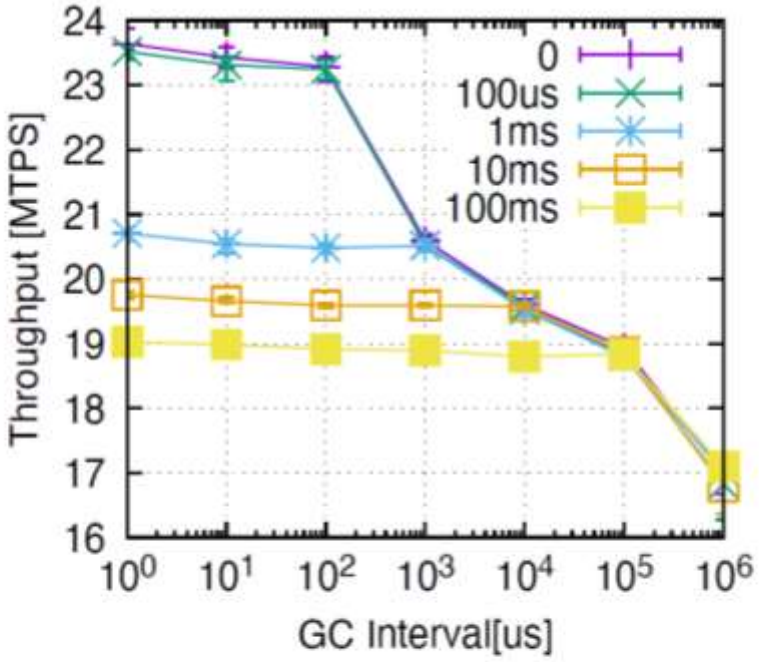


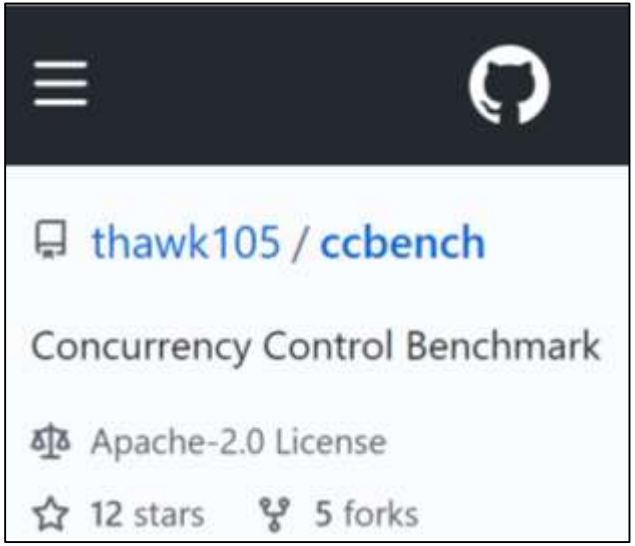
Figure 14: Analysis of RapidGC: YCSB-A, skew 0, 224 threads, 1M records, payload 4 bytes, 10 ops/trans. Inserted delays for long transactions: 0 to 10 ms.

1. MVCCはversionにより性能向上
2. Version costが負担である点は周知の事実
3. Long transaction が 1つでも入ってきたら性能律速。

TPC-C, Indexの影響

Figure 19: Breakdown of TPC-C-NP-Insert, 224 threads, 224 warehouses, NewOrder and Payment transactions

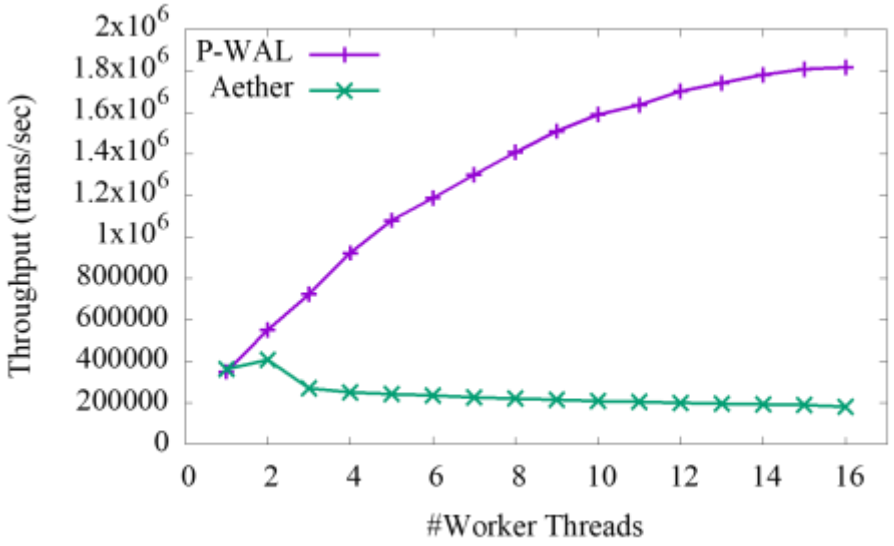
- まとめ
  - 目標：世界最速のトランザクション技法を開拓
  - 成果：CCBenchで近代的並行性制御法を分析



- 予定
  - 最速の並行性制御法を実現
    - Silo++
    - 新規手法
  - 永続化法を導入
    - 並列書き込み（例：P-WAL）
    - Epoch方式

Concurrency Control

Recovery



Yasuhiro Nakamura, Hideyuki Kawashima, Osamu Tatebe: Integration of TicToc Concurrency Control Protocol with Parallel Write Ahead Logging Protocol. Int. J. Netw. Comput. 9(2): 339-353 (2019)

成果一覧

1. Takayuki Tanabe, Takashi Hoshino, Hideyuki Kawashima, Osamu Tatebe: An Analysis of Concurrency Control Protocols for In-Memory Databases with CCBench. Proc. VLDB Endowment 13(13): 3531-3544 (2020), 査読有,
2. Jun Nemoto, Hideyuki Kawashima, Motomichi Toyama: Exploiting SIMD Instructions in Partial-Evaluation-Based Query Compiler. Proc. The 1st Workshop on Distributed Infrastructure, Systems, Programming and AI (DISPA), 査読有
3. Kohei Hiraga, Osamu Tatebe, Hideyuki Kawashima: Scalable Distributed Metadata Server Based on Nonblocking Transactions. J. Univers. Comput. Sci. 26(1): 89-106 (2020)
4. Yasuhiro Nakamura, Hideyuki Kawashima, Osamu Tatebe: Integration of TicToc Concurrency Control Protocol with Parallel Write Ahead Logging Protocol. International Journal on Network Computing 9(2): 339-353 (2019), 査読有
5. Harunobu Daikoku, Hideyuki Kawashima, Osamu Tatebe: Skew-Aware Collective Communication for MapReduce Shuffling. IEICE Transactions on Information Systems 102-D(12): 2389-2399 (2019), 査読有