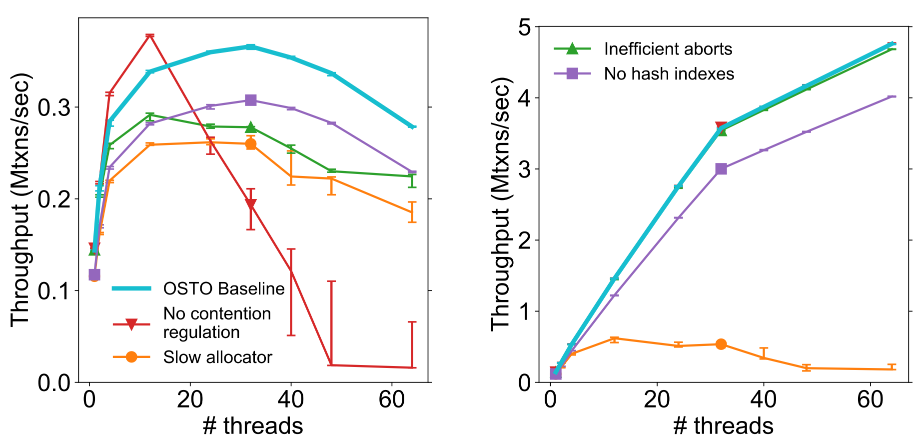
Opportunities for Optimism in Contended Main-Memory Multicore Transactions

Yihe Huang, William Qian, Eddie Kohler, Barbara Liskov, Liuba Shrira

Abstract. Optimistic concurrency control, or OCC, can achieve excellent performance on uncontended workloads for main-memory transactional databases. Contention causes OCC's performance to degrade, however, and recent concurrency control designs, such as hybrid OCC/locking systems and variations on multiversion concurrency control (MVCC), have claimed to outperform the best OCC systems. We evaluate several concurrency control designs under varying contention and varying workloads, including TPC-C, and find that implementation choices unrelated to concurrency control may explain much of OCC's previously-reported degradation. When these implementation choices are made sensibly, OCC performance does not collapse on high-contention TPC-C. We also present two optimization techniques, *commit-time updates* and *timestamp splitting*, that can dramatically improve the high-contention performance of both OCC and MVCC. Though these techniques are known, we apply them in a new context and highlight their potency: when combined, they lead to performance gains of 3.4× for MVCC and 3.6× for OCC in a TPC-C workload.

Description. The performance of multicore main-memory transactional systems is a subject of intense study [1, 3, 4, 6–11, 13]. Techniques based on optimistic concurrency control (OCC) perform extremely well on low-contention workloads, thanks to their efficient use of shared memory bandwidth and avoidance of unnecessary memory writes. On high-contention workloads, however, OCC can experience frequent aborts and, in the worst case, *contention collapse*, where performance for a class of transactions crashes to nearly zero due to repeated conflicts.

Recent designs targeted at high-contention workloads, including partially-pessimistic concurrency control [10], dynamic transaction reordering [13], and multiversion concurrency control (MVCC) [5, 6], change the transactional concurrency control protocol to better support high-contention transactions. The evaluations of these designs show dramatic benefits over OCC for high-contention workloads, including TPC-C, and some show benefits over OCC even at low contention [6].



(a) *One warehouse (high contention).* **(b)** *One warehouse per worker (low contention).*

Figure 1: *OCC throughput under TPC-C full-mix showing impact of basis factors. Factor optimizations are* individually *turned off from the optimized baseline to demonstrate the capping effect of each factor.*

Many of these evaluations compare different code bases, however, which could cause mere implementation differences to unduly influence the results. We therefore analyzed several main-memory transactional systems, including

Benchmark	OSTO	OSTO+CU	OSTO+TS	OSTO+CU+TS	MSTO	MSTO+CU	MSTO+TS	MSTO+CU+TS
TPC-C	276	286 (1.04×)	432 (1.57×)	1001 (3.63×)	431	269 (0.62×)	410 (0.95×)	1456 (3.38×)
YCSB	473	855 (1.81×)	466 (0.99×)	844 (1.78×)	326	1851 (5.68×)	687 (2.11×)	2487 (7.64×)
Wikipedia	170	487 (2.86×)	167 (0.98×)	483 (2.84×)	128	311 (2.43×)	128 (1.01×)	449 (3.52×)
RUBiS	1378	1924 (1.40×)	1368 (0.99×)	1957 (1.42×)	1475	1692 (1.15×)	1505 (1.02×)	1721 (1.17×)

Figure 2: *Throughput of STOv2 with high-contention optimizations (CU and TS) in Ktxns/sec at 64 threads in high-contention benchmarks, with improvements over respective baselines in parentheses.*

Silo [9], DBx1000 [12], Cicada [6], ERMIA [5], and MOCC [10]. We found several underappreciated engineering choices – we call them *basis factors* – that dramatically affect these systems' high-contention performance. For instance, some abort mechanisms exacerbate contention by obtaining a hidden lock in the language runtime. Figure 1 summarizes results when measured using a high-contention TPC-C benchmark under OCC.

To better isolate the effect of concurrency control (CC) on performance, we implement and evaluate three CC mechanisms – OCC, TicToc [13], and MVCC – in a new system, *STOv2*, that makes good, consistent implementation choices for all basis factors. We show results up to 64 cores and for several benchmarks, including low- and high-contention TPC-C, YCSB, and benchmarks based on Wikipedia and RUBiS. With basis factors controlled, OCC performance does not collapse on these benchmarks, even at high contention, and OCC and TicToc significantly outperform MVCC at low and medium contention. This contrasts with prior evaluations, which reported OCC collapsing at high contention [2] and MVCC performing well at all contention levels [6].

In addition, we introduce, implement, and evaluate two optimization techniques that can improve performance on high-contention workloads for all concurrency control schemes we evaluated (OCC, TicToc, and MVCC). These techniques safely eliminate classes of conflict that were common in our workloads. First, the *commit-time update* (CU) technique eliminates conflicts that arise when read-modify-write operations, such as increments, are implemented using plain reads and writes. Second, many records have fields that rarely change; the *timestamp splitting* (TS) technique avoids conflicts between transactions that read rarely-changing fields and transactions that write other fields. These techniques have workload-specific parameters, but they are conceptually general, and we applied them without much effort to every workload we investigated. Like MVCC and TicToc, the techniques improve performance on high-contention workloads. However, unlike MVCC, these optimizations have little performance impact at low contention; unlike TicToc and MVCC, they help on every benchmark we evaluate, not just TPC-C; and they benefit TicToc and MVCC as well as OCC. Though the techniques are widely known, our variants are new, and we are the first to report their application to TicToc and MVCC. Figure 2 summarizes the results with high-contention optimizations.

This work is based on a paper accepted to VLDB 2020.

References

[1] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI '14, pages 401–414. ACM, 2014.

[2] J. M. Faleiro and D. J. Abadi. Rethinking serializable multiversion concurrency control. *PVLDB*, 8(11):1190–1201, 2015.

[3] N. Herman, J. P. Inala, Y. Huang, L. Tsai, E. Kohler, B. Liskov, and L. Shrira. Type-aware transactions for faster concurrent code. In *Proceedings of the 11th European Conference on Computer Systems*, EuroSys '16. ACM, 2016.

[4] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-Store: A high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2):1496–1499, Aug. 2008.

[5] K. Kim, T. Wang, R. Johnson, and I. Pandis. ERMIA: Fast memory-optimized database system for heterogeneous workloads. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1675–1687. ACM, 2016.

[6] H. Lim, M. Kaminsky, and D. G. Andersen. Cicada: Dependably fast multi-core in-memory transactions. In *Proceedings of the 2017 International Conference on Management of Data*, SIGMOD '17, pages 21–35. ACM, 2017.

[7] S. Mu, S. Angel, and D. Shasha. Deferred runtime pipelining for contentious multicore software transactions. In *Proceedings of the 14th European Conference on Computer Systems*, EuroSys '19, pages 40:1–40:16. ACM, 2019.

[8] N. Narula, C. Cutler, E. Kohler, and R. Morris. Phase reconciliation for contended in-memory transactions. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '14, pages 511–524. ACM, 2014.

[9] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, SOSP '13, pages 18–32. ACM, 2013.

[10] T. Wang and H. Kimura. Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. *PVLDB*, 10(2):49–60, 2016.

[11] Z. Wang, S. Mu, Y. Cui, H. Yi, H. Chen, and J. Li. Scaling multicore databases via constrained parallel execution. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1643–1658. ACM, 2016.

[12] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the abyss: An evaluation of concurrency control with one thousand cores. *PVLDB*, 8(3):209–220, 2014.

[13] X. Yu, A. Pavlo, D. Sanchez, and S. Devadas. TicToc: Time traveling optimistic concurrency control. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1629–1642. ACM, 2016.