Durable Transactional Memory Can Scale with TIMESTONE

R. Madhava Krishnan Jaeho Kim[†*] Ajit Mathew Xinwei Fu
Anthony Demeri Changwoo Min Sudarsun Kannan[‡]
Virginia Tech [†]*Huawei Dresden Research Center* [‡]*Rutgers University*

1. Introduction

New emerging non-volatile main memory (NVMM) technologies, such as Intel Optane, provides persistence property along with traditional main memory characteristics, such as byte-addressability and low access latency. In addition, the NVMM offers data durability and larger in-memory capacity at a significantly lower `$/GB` than the traditional DRAM. Despite NVMMs being slower in read and write latency than traditional DRAMs, they allow software to have a larger but slightly slower in-memory capacity and almost attain free durability of data. Nevertheless, manycore scalability is becoming an inevitable design principle when designing NVMM software as NVMMs are expected soon to be a part of data center scale manycore servers.

So a competent NVMM library should provide better performance and scalability, have a minimal write amplification, be memory efficient, and have a broad-ranging applicability. Unfortunately, none of the prior work exhibit all the above capabilities. Existing DTM approaches [3, 7, 5, 4, 1, 2] supports durable composability and provides full-data consistency. However, our analysis shows that none of the DTM systems scales beyond 16 cores (see Figure 2). For example, DudeTM [4] and Mnemosyene [7] scale poorly because of the underlying STM, which is known for its poor scalability. They extend STM with an extra durability layer, which incurs a high write amplification (∼4-7×), as shown in Figure 2 in the course of guaranteeing crash consistency.

On the other hand, Romulus [1] and KaminoTX [5] minimize the write amplification by maintaining a full backup of the NVMM, which derails the cost effectiveness of NVMM. Moreover, existing DTM systems supports a limited write parallelism, which impacts their scalability, or leave it entirely to application developers to implement by themselves with locks [3]. More recently, Pisces [2] attempts to provide scalability by providing snapshot isolation. But the dual version concurrency control and the synchronous write during log reclamation in Pisces are bound to affect scalability and increase the write amplification like other DTM approaches. In a nutshell, the prior DTM systems does not scale for two main reasons 1) limited read write parallelism 2) high write amplification in the critical path of the transaction.

To address all these problems, we propose a new DTM system, named TIMESTONE [1]. which achieves *1) scalability across multiple cores*, guarantees *2) crash consistency with significantly lower write amplification (< 1)* and also maintains *3) minimal additional memory footprint*.

2. Overview of TIMESTONE

In this section we briefly explain our design chocies and how we use them to achieve our goals in TIMESTONE.

2.1. Multi-Versioning

Unlike Software Transactional Memory (STM), Multi-Version Concurrency Control (MVCC) supports non-blocking reads and disjoint non-blocking writes thus supporting higher read-write parallelism. TIMESTONE adopts MVCC to achieve high scalability and to support mixed isolation levels [2]. Given these benefits, naive adoption of MVCC will incur a lot of write traffic, eventually increasing the write amplification in the TIMESTONE. To solve this problem we propose a multi-layered hybrid logging scheme called the *TOC logging*.

2.2. TOC Logging

As illustrated in Figure 1, in TOC logging, we use a volatile log on the DRAM, named transient version log (`TLog`), and two non-volatile logs, namely operational log (`OLog`) and checkpoint log (`CLog`). The TIMESTONE transactions are executed in the volatile `TLog` which significantly reduces the transactional latency in par to executing on a slower NVMM (step ❶ in Figure 1). Also, we do not immediately writeback the updates to NVMM instead we allow sufficient time for updates to coalesce (step ❸ and ❺ in Figure 1). This is key to achieving a lower write amplification in TIMESTONE as only the latest updates (step ❹ in Figure 1) are checkpointed to the `CLog` in NVMM by absorbing redundant writes back in the `TLog`. The `OLog` is important to guarantee immediate durability (step ❷ in Figure 1) and if there is a power failure before checkpointing the updates to NVMM then `OLog` can be replied to get back to the last-commit state before the failure occured. Finally, the `CLog` guarantees the consistency of the master objects and further reduces the write amplification by writing back (step ❻ in Figure 1) only the latest checkpoints to the master.

2.2.1. Scalable Garbage Collection TIMESTONE maintains fixed size logs and hence the memory usage of logs are limited. If one or more logs becomes full, this could block all writes until logs are reclaimed. Hence garbage collection can directly impact write throughput. Also, a synchronous and non-scalable garbage collection scheme can quickly become a bottleneck hampering the performance of the system [8].

For the garbage collection to be scalable, TIMESTONE must identify safe objects to reclaim without any centralized lookup or coordination. Importantly, garbage collection must be

[*]Jaeho Kim had contributed this work while he was at Virginia Tech.

[1]This work has been accepted to ASPLOS 2020, the full paper is attached with this submission (Page 3-16)

[2]Please refer our full paper [6] for more details on mixed isolation.

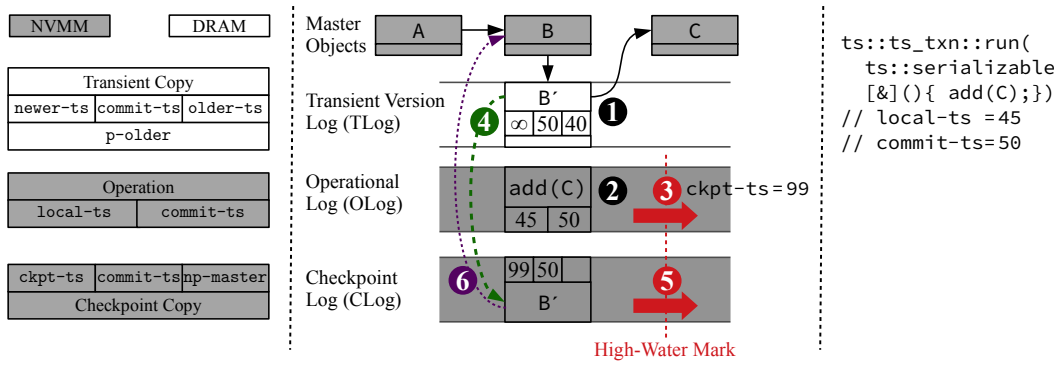

Figure 1: Illustrative example of adding a node in a TIMESTONE linked list. A thread adds a node `C` to a linked list in a TIMESTONE transaction (`ts_txn::run()`) with a serializable isolation level (`ts::serializable`). Consider a transaction that starts at timestamp 45 (*i.e.*, `local-ts = 45`) and commits at 50 (*i.e.*, `commit-ts= 50`). TIMESTONE first creates a copy of node `B` in `TLog` (`B'`) and updates its next pointer to node `C` ❶. When the transaction commits, TIMESTONE persists the executed operation (`add(C)`) to `OLog` making the transaction immediately durable ❷. Steps ❸ and ❺ denotes the log capacity crossing the high-water mark and this triggers the checkpointing for `TLog` reclamation ❹ and writeback for `CLog` reclamation ❻. During checkpointing, TIMESTONE checkpoints the latest transient copy (node `B'`) to the `CLog` so the `TLog` can be reclaimed ❹. In the `CLog` reclamation, TIMESTONE writes back the latest checkpoint copy (node `B'`) to the master object and the checkpoint log can be reclaimed safely ❻. The reclamation process is detailed in [6].

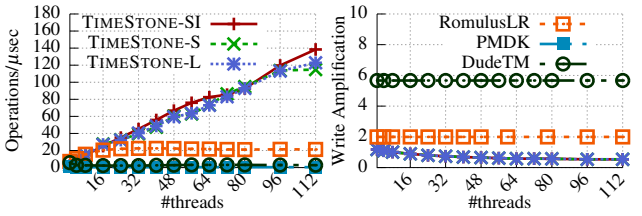

Figure 2: Performance comparison of DTM systems for concurrent hash tables with 2% update. Except TIMESTONE, prior systems suffer from poor scalability and high write amplification.

NVMM-write aware so that it does not increase direct writes to NVMM. Hence, TIMESTONE employs a *timestamp-based reclamation* scheme where decisions like what/when to reclaim are solely made based on the object-local timestamp without accessing shared structures. To harness concurrency in the garbage collection, TIMESTONE delegates responsibility of reclamation to each thread that holds the log itself (*i.e.*, *concurrent reclamation*). To further reduce NVMM writes, we introduce *best-effort reclamation*, which reclaims objects that do not incur NVMM writes. Please see our full paper [6] for more details.

3. Evaluation

We use a system with Intel Optane DC Persistent Memory (DCPMM) for our evaluation. We evaluated the TIMESTONE against all of the latest DTM works and as shown in Figure 2 it outperforms all of them upto 40× and shows a better scalability. While the prior DTM systems suffers from 2×-6× write amplification, TIMESTONE maintains it below 1. We also evaluated the real world impact of TIMESTONE with KyotoCabinet and YCSB workloads. TIMESTONE enabled KyotoCabinet scales upto 64 cores while the vanilla KyotoCabinet saturates at the 16-core mark. For YCSB, TIMESTONE enabled B+-tree outperforms DudeTM by upto 6× and shows a better scalability across different workloads.

4. Conclusion

TIMESTONE at its' core adopts *MVCC* for achieving high scalability and *TOC logging* to always keep write amplification < 1. TIMESTONE outperforms all the prior DTM systems for different of workloads and benchmarks. For more details on TIMESTONE mixed isolation support, concurrent log reclamation, decentralized log syncronization and a thorough evalaution, please refer to our full paper [6].

References

[1] Andreia Correia, Pascal Felber, and Pedro Ramalhete. Romulus: Efficient Algorithms for Persistent Transactional Memory. In *Proceedings of the ACM symposium on Parallelism in algorithms and architectures (SPAA)*, Vienna, Austria, July 2018.

[2] Jinyu Gu, Qianqian Yu, Xiayang Wang, Zhaoguo Wang, Binyu Zang, Haibing Guan, and Haibo Chen. Pisces: A Scalable and Efficient Persistent Transactional Memory. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, pages 913–928, Renton, WA, July 2019.

[3] INTEL. Persistent Memory Development Kit, 2019.

[4] Mengxing Liu, Mingxing Zhang, Kang Chen, Xuehai Qian, Yongwei Wu, Weimin Zheng, and Jinglei Ren. DudeTM: Building Durable Transactions with Decoupling for Persistent Memory. In *Proceedings of the 22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Xi'an, China, April 2017.

[5] Amirsaman Memaripour, Anirudh Badam, Amar Phanishayee, Yanqi Zhou, Ramnatthan Alagappan, Karin Strauss, and Steven Swanson. Atomic In-place Updates for Non-volatile Main Memories with Kamino-Tx. EuroSys17.

[6] Madhava Krishnan R., Jaeho Kim, Ajit Mathew, Xinwei Fu, Anthony Demeri, Changwoo Min, and Sudarsun Kannan. Durable Transactional Memory Can Scale with Timestone. In *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Lausanne, Switzerland, March 2020.

[7] Haris Volos, Andres Jaan Tack, and Michael M. Swift. Mnemosyne: Lightweight Persistent Memory. In *Proceedings of the 21st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Atlanta, GA, April 2016.

[8] Yingjun Wu, Joy Arulraj, Jiexi Lin, Ran Xian, and Andrew Pavlo. An Empirical Evaluation of In-memory Multi-version Concurrency Control. In *Proceedings of the 43rd International Conference on Very Large Data Bases (VLDB)*, pages 781–792, TU Munich, Germany, August 2017. VLDB Endowment.