TENET: Memory Safe and Fault Tolerant Persistent Transactional Memory

21st USENIX Conference on File and Storage Technologies

R. Madhava Krishnan; Diyu Zhou; Wook-Hee Kim; Sudarsun Kannan; Sanidhya Kashyap; Changwoo Min

Paper Notes

By JeongHa Lee

Abstract

Byte-addressable non-volatile memory (NVM) allows programs to directly access storage using memory interface without going through the expensive conventional storage stack. However, direct access to NVM makes the NVM data vulnerable to software bugs and hardware errors. This issue is critical because, unlike DRAM, corrupted data can persist forever, even after the system restart. Albeit the plethora of research on NVM programs and systems, there is little focus on protecting NVM data from software bugs and hardware errors.

In this paper, we propose TENET, a new NVM programming framework, which guarantees memory safety and fault tolerance to protect NVM data against software bugs and hardware errors. TENET provides the popular persistent transactional memory (PTM) programming model. TENET leverages the concurrency guarantees (i.e., ACID properties) of PTM to provide performant and cost-efficient memory safety and fault tolerance. Our evaluations show that TENET offers an enhanced protection scope at a modest performance overhead and storage cost as compared to other PTMs with partial or no memory safety and fault tolerance support.

Problem Statement and Research Objectives

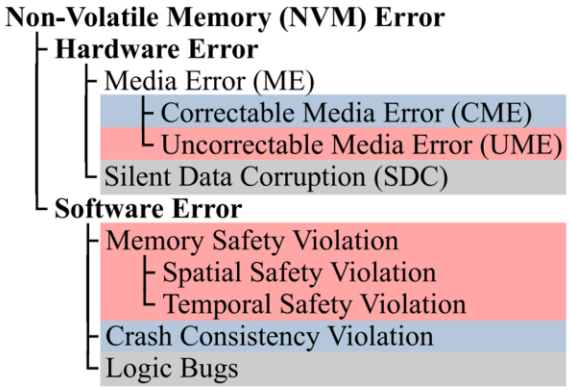

Figure 1: Classification of errors in NVM. TENET handles *UME*, *Spatial and Temporal Safety Violation* bugs (red). TENET relies on the hardware ECC to fix *CME* and the underlying PTM to handle *Crash Consistency Violations* such as atomicity and persistence ordering (blue). *Silent Data Corruption* in the hardware (*e.g.*, CPU faults) and *logical bugs* in the application are out of scope (grey).

The direct persistence of NVM opens several challenges in protecting data from software bugs (e.g., "memory scribbles") and **media errors**. NVM data can be permanently corrupted due to a single memory scribble, which roots from a **spatial safety violation** (e.g., buffer overflow) or a **temporal safety violation** (e.g., use-after-free) in a program.

Limitations of previous study

1. None of these approaches prevent NVM data corruption due to memory safety violations on "DRAM data".
2. None of them guarantee to protect NVM data from temporal safety violations.

* TENET uses TimeStone as its transaction abstraction.

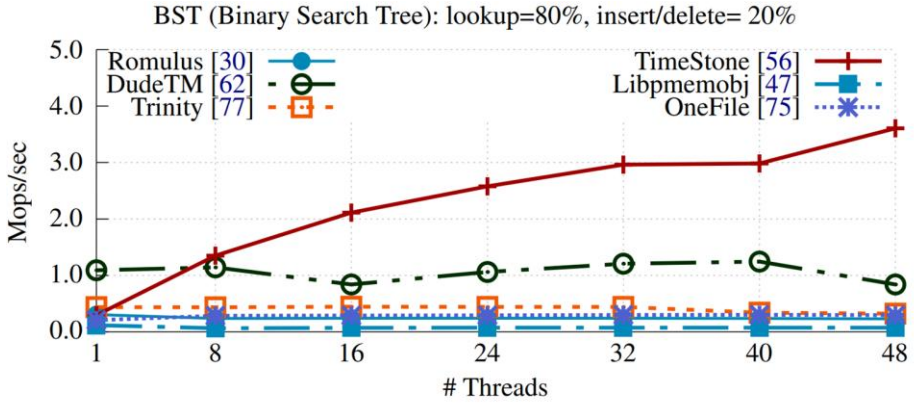

Figure 2: Performance of TimeStone against other PTMs. None of the PTMs are memory safe or fault tolerant against UME.

Problem Statement and Research Objectives

- **Multi-version concurrency control**: TimeStone follows multi-version concurrency control (MVCC). With MVCC, TimeStone supports **non-blocking reads and concurrent disjoint writes**, achieving high concurrency. For each object created by the application (e.g., B-tree node), TimeStone allocates a ***master object*** on NVM.

- **Operational log based immediate durability**: TimeStone uses a **DRAM-NVM hybrid logging technique**, named TOC logging for efficient crash consistency. The TOC logging consists of ***Transient Version Log (TLog)*** on DRAM, ***Operational log (OLog)*** and ***Checkpoint log (CLog)*** on NVM.

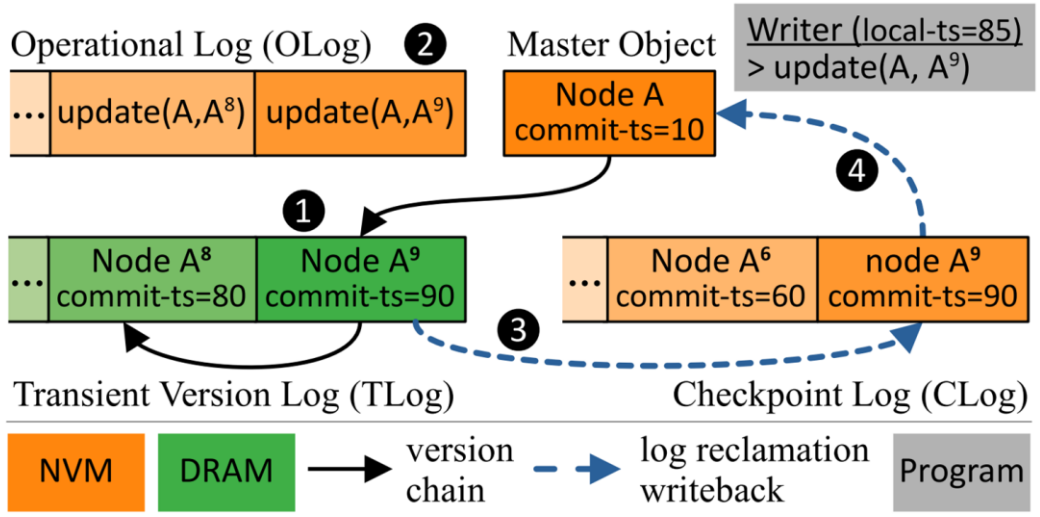

Figure 3: An illustrative example of updating `Node A` to its 9th version (A^9) in TimeStone.

Proposed Method

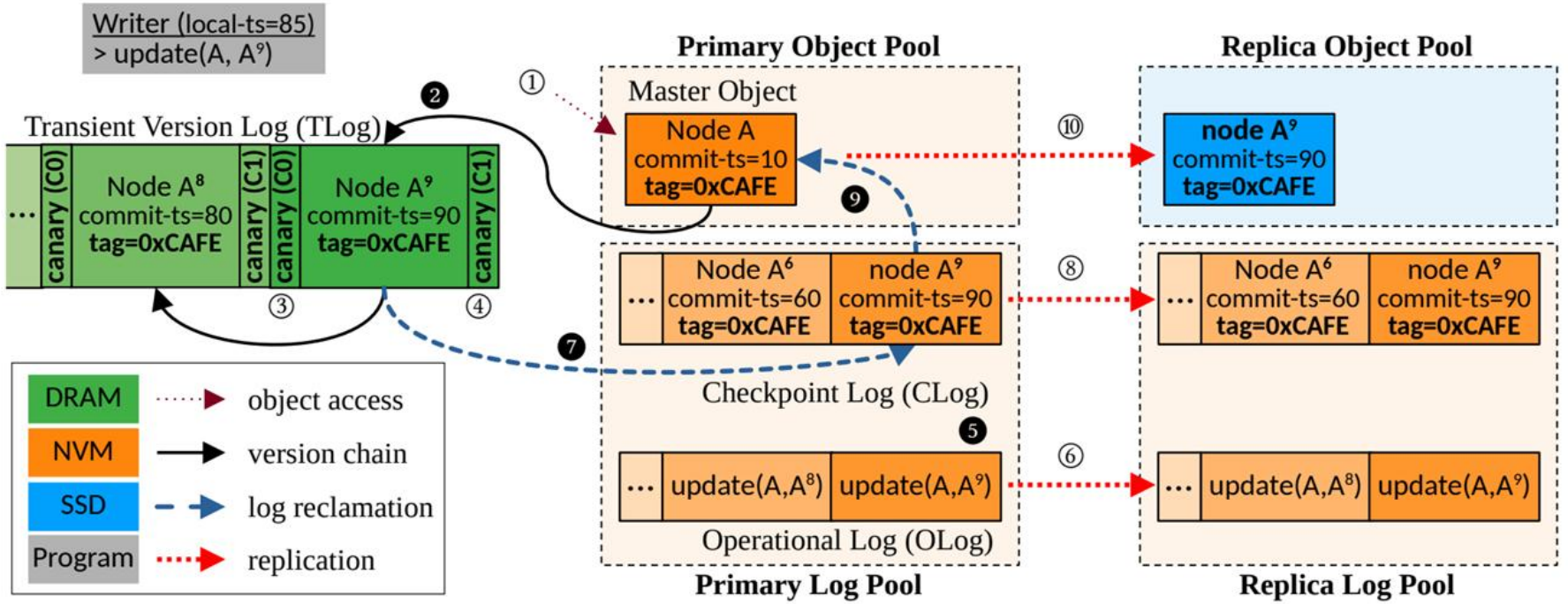

Figure 4: Overall architecture of TENET with an example of updating `Node A` to its 9th version (A^9). ⓝ denotes the newly added memory safety checks and replication to the TimeStone transaction. Note that the application has read/write access to DRAM and read-only permission for NVM. When accessing `Node A`, TENET validates its temporal safety by comparing the tags, `0xCAFE` (①). If the tags do not match, the transaction is aborted. Otherwise, the writer proceeds to traverse the `Node A`'s version chain, makes a copy of the latest version (A^8) in its `TLog` and updates it to A^9 (❷). Upon commit, `Node` A^9 is validated for spatial safety by checking the canary values (③ and ④). The transaction is aborted if the validation fails. Otherwise, the writer commits the transaction by updating its `OLog` (❺) for durability and it also synchronously updates the replica `OLog` for fault tolerance (⑥). When reclaiming the `TLog`, `Node` A^9 is once again validated for spatial safety before checkpointing it to the `CLog` (❼) followed by synchronously updating the replica `CLog` (⑧). Similarly, when the `CLog` is full, TENET writes back the latest checkpoint (`Node` A^9) to the original master object `Node A` (❾). The updated Node A is then *asynchronously* replicated to the disk (⑩).

Evaluation and Results

1. What are the performance overhead of TENET's memory safety and off-critical path disk replication techniques?

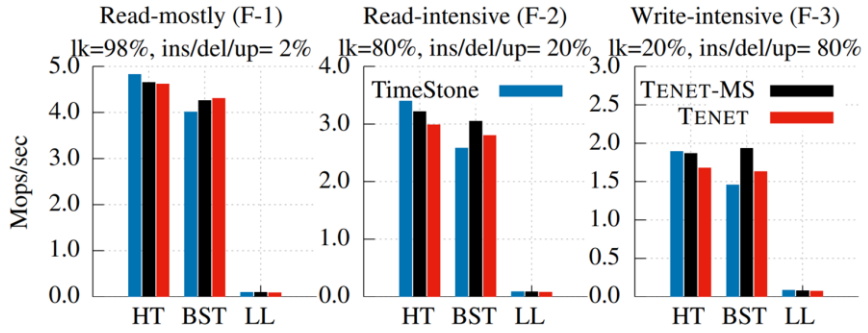

Figure 6: Performance comparison of TENET-MS and TENET against TimeStone for Hash Table (HT), Binary Search Tree (BST), and Linked List (LL) for 24 threads.

2. What is the tail latency of TENET?

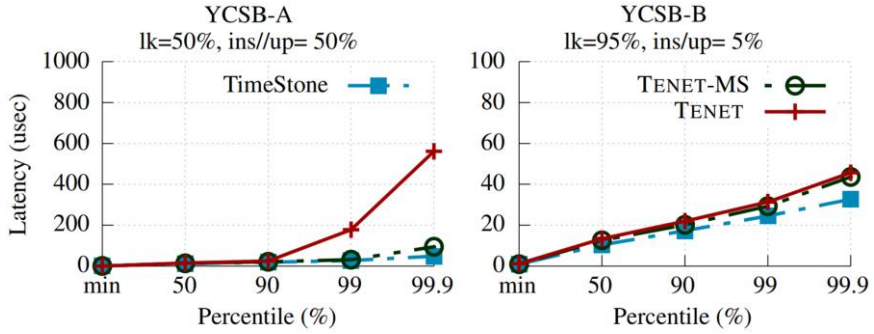

Figure 10: Tail latency comparison of TENET-MS and TENET against TimeStone for B+tree with 24 threads.

3. How does TENET perform in comparison with the other state-of-the-art memory safe PTMs?

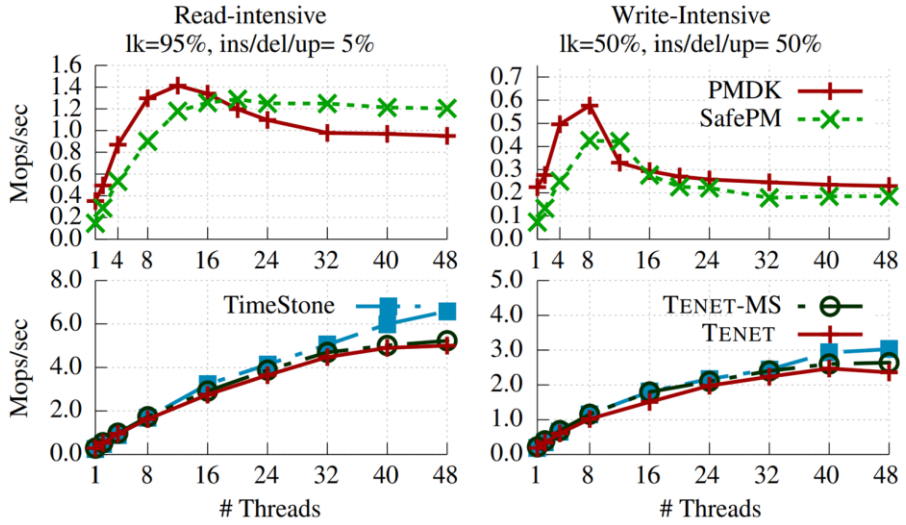

Figure 9: TENET-MS vs SafePM: performance overhead study with hash table for read-intensive and write-intensive workloads.

PTM	Spatial Safety	Temporal Safety	UME	NVM Cost
Libpmemobj [47]	No	No	Yes	High
TimeStone [56]	No	No	No	None
SafePM [27]	Yes	Yes	No	Moderate
Pangolin [94]	Partial	No	Yes	Moderate
TENET-MS	Yes	Yes	No	None
TENET	**Yes**	**Yes**	**Yes**	**Low**

Table 1: Comparison of TENET against other PTMs.