SpecPMT: Speculative Logging for Resolving Crash Consistency Overhead of Persistent Memory

Chencheng Ye[1], Yuanchao Xu[2], Xipeng Shen[3], Yan Sha[1], Xiaofei Liao[1], Hai Jin[1], Yan Solihin[4]
[1]Huazhong University of Science and Technology [2]University of California Santa Cruz
[3]North Carolina State University [4]University of Central Florida

1 INTRODUCTION

Byte-addressable persistent memory features high density, byte-addressability, and data persistency, allowing software to access durable data in main memory. Experiments in cloud services and high-performance computing have demonstrated the potential of persistent memory in building crash-resilient software, improving software performance, and increasing development efficiency.

Programmers often use *persistent memory transactions* to achieve crash consistency — transactions provide simple yet powerful semantics of crash-atomic updates, i.e., they ensure either all or no transactional updates on persistent memory locations are observable after a crash.

Existing persistent memory transactions, however, incur large overheads, because of the need to log data updates to provide transactional semantics. For example, the most commonly used implementation, Intel PMDK, was reported to incur 6× slowdowns to program executions. This *crash consistency overhead* must be substantially lowered before persistent memory becomes attractive for wide adoption.

This important problem has been the focus of recent studies. Some of the proposed solutions are specially designed for a certain algorithm or a data structure, and are not applicable to general applications. More general solutions include pure software methods [1] and hardware support [3] to mitigate the logging overhead. Although these studies have made important contributions and reduced the logging overheads by as much as 2.6×, crash consistency overheads are still too large for practical use. As shown in Figure 1, even after applying state-of-the-art software (SPHT [1]), the programs in STAMP still suffer from an average of 50–161% execution time overheads compared to versions without persistent memory transactions.

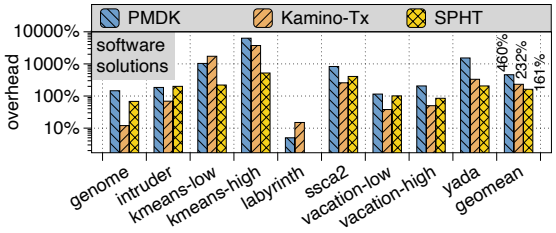

Figure 1: Overhead incurred by state-of-the-art software persistent memory transactions.

In this work, we present *speculative logging*, a novel approach to reduce crash consistency overheads. The design was motivated by the fact that the main sources of crash consistency overheads in persistent transactions are the use of memory fences and the persisting of data. The design builds upon the *key insight* that data can be speculatively logged early, and doing so removes the sources of performance overheads.

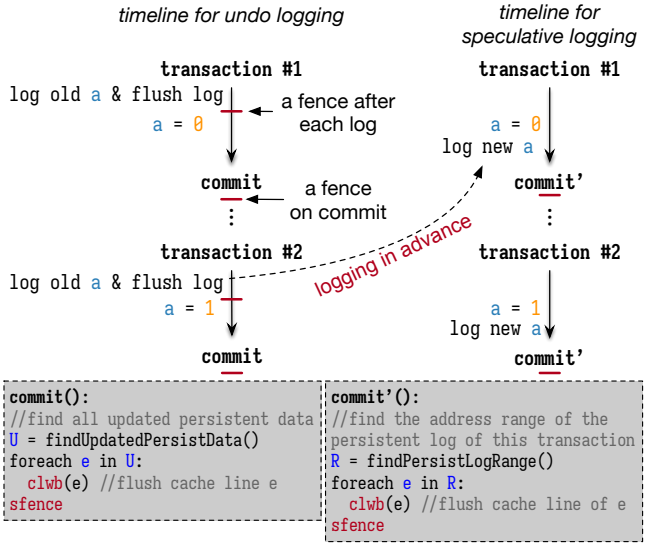

Figure 2: Timeline for two consecutive persistent transactions. The undo logging transaction (left) records the old value of a datum, using multiple fences in a transaction. Speculative logging transaction (right) records the new value of the memory location without fences. It persists all log records but data with only one fence each transaction before the transaction commits.

Figure 2 illustrates the basic idea of speculative logging. In a typical persistent transaction (Figure 2 left), a datum is (undo) logged before being updated in the transaction. A flush and fence ensure that the log write persists before the data write. In contrast, speculative logging (Figure 2 right) moves up the logging to a point as early as the last transaction where the datum was updated. By doing that, several benefits are achieved. First, the log write leverages the commit of the first transaction to persist the log, forgoing the need for memory fences for persisting the log. Second, it defers the flush to the transaction commit, making the transaction execute faster. Third, as the log persists the most recent value of the datum once a transaction commits, the data write in the same transaction becomes optional. If the data write fails to persist, the post-crash recovery can rely on log records to rebuild the updated data. Thus, the data write no longer requires a flush. Because persisting log

* This work has been published at ASPLOS 2023[4]. Chencheng Ye, Yan Sha, Xiaofei Liao, and Hai Jin are with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China.

records involve sequential writes, they have better spatial locality and are faster than persisting data writes (which may be more random).

The work realizes the speculative logging transaction in both software and hardware to demonstrates the potential applicability and benefits of this concept. The software-only solution is compatible with existing hardware. It uses novel speculative log management featuring a compact log format and background memory reclamation, and a crash recovery protocol based on the compact log format. The proof-of-concept implementation of the solution achieves about 2.7× speedup over the state-of-the-art in-place solution Kamino-Tx [2]. However, two key drawbacks remain: (1) memory space overheads, which is 3× persistent memory space, and (2) the reliance on dedicated background memory reclamation threads.

The hardware solution reduces memory space overheads while preserving performance with a novel architecture for a hybrid logging model. The model allows softwares to use speculative logging for *hot* (i.e., frequently updated) data and undo logging for cold data. By controlling the threshold of data hotness, the user can set an arbitrary bound on the size of the speculative log area. The hybrid logging model also enables a software-hardware co-designed *log reclamation scheme*. The log reclamation runs in the foreground, and therefore does not require dedicated reclamation threads. This allows softwares to reclaim speculative log records with a few instructions without blocking other running threads. By simply clearing the bits associated with a given epoch with new instructions, the hardware can easily reclaim all the log records created in the epoch.

Together, these solutions provide an alternative approach to current undo logging–based persistent transactions, namely *speculatively persistent memory transactions (SpecPMT)*. Comparisons with state-of-the-art in-place update methods (Kamino-Tx [2] and EDE [3]) show that SpecPMT reduces the execution time overheads of the prior methods from 232% and 50% to 10% and 7%, respectively.

SpecPMT achieves all the desirable properties of a persistent memory transaction while keeping transaction overheads low: (1) it uses in-place data updates; (2) it eliminates fences between logging and data updates; (3) it does not block transaction commit with data persistence; (4) it supports a software-only or a lightweight hardware implementation; and (5) it is data structure agnostic rather than data structure specific.

2 SPECULATIVE LOGGING EXAMPLE

We illustrate speculative logging with a running example in Figure 3, which shows two transactions that both update memory locations *a* and *b* (The example applies to both software and hardware SpecPMT proposed in this paper.). The persistent memory state for data and logs are shown for different snapshots in time. The snapshots begin when the first transaction has just committed. Here both locations have been updated, and speculative log records for *a* and *b* have been created. As the second transaction executes (second snapshot with *b=10*), it creates log records for *a* and *b*, and appends them in the log. Note that at this point, if a crash occurs, the first transaction log records are sufficient to restore data to the point before the second transaction by undoing any changes to *a* and *b* in the second transaction. Hence, new data values and the associated log records in the second transaction may remain in the volatile memory. When the second transaction commits (third snapshot), the commit ensures the new speculative log records persist along with the transaction commit metadata. Note that updates on locations (e.g., *a*) do not need to persist (e.g., be flushed) at this point, as second transaction log records are sufficient to replay the non-persisted data updates if a crash occurs (the speculative log entries function as a redo log for the just committed transaction). Finally, when log reclamation is triggered (last snapshot), explicitly or implicitly, the reclamator finds that log records from the first transaction are stale and thus can be removed. The first transaction metadata is also removed since no fresh log record remains.

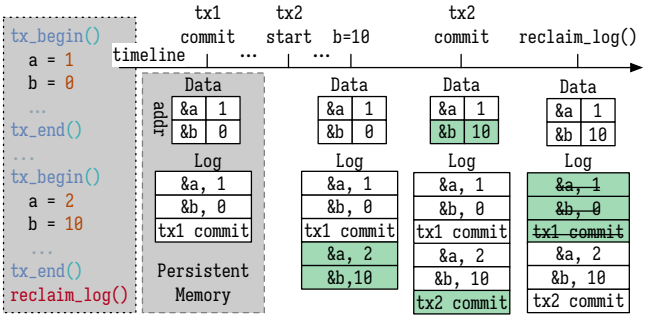

Figure 3: A codelet and the memory state snapshots with timeline, illustrating the mechanism of speculative logging

The post-crash recovery is straightforward, as described in the full paper [4].

3 EVALUATION

The speculative logging approach removes most in-transaction fences and data persistence, enforces immediate persistence, and performs direct memory loads and in-place data updates. The software-only design achieves a low 10% execution time transaction overhead, compared to a state-of-the-art solution of 232%. The hardware-supported design keeps the performance of the software-only solution while bounding its memory consumption. Compared to the state-of-the-art undo and redo logging, it lowers execution time overheads by 86% and 76%, respectively, while requiring a modest 0.91KB on-chip storage overhead.

REFERENCES

[1] Daniel Castro, Alexandro Baldassin, João Barreto, and Paolo Romano. 2021. SPHT: Scalable Persistent Hardware Transactions. In *Proceedings of the 19th USENIX Conference on File and Storage Technologies*. 155–169.

[2] Amirsaman Memaripour, Anirudh Badam, Amar Phanishayee, Yanqi Zhou, Ramnatthan Alagappan, Karin Strauss, and Steven Swanson. 2017. Atomic in-place updates for non-volatile main memories with kamino-tx. In *Proceedings of the Twelfth European Conference on Computer Systems*. 499–512.

[3] Thomas Shull, Ilias Vougioukas, Nikos Nikoleris, Wendy Elsasser, and Josep Torrellas. 2021. Execution Dependence Extension (EDE): ISA Support for Eliminating Fences. In *Proceedings of the ACM/IEEE 48th Annual International Symposium on Computer Architecture*. 456–469.

[4] Chencheng Ye, Yuanchao Xu, Xipeng Shen, Yan Sha, Xiaofei Liao, Hai Jin, and Yan Solihin. 2023. SpecPMT: Speculative Logging for Resolving Crash Consistency Overhead of Persistent Memory. *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (2023).