This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

ReCoW: Kernel-User Collaborative Copy-on-Write Transactions for Persistent Memory via Address Remapping

Aoxin Wei, Jian Zhou, Shuhan Bai, Yufan Jia, Jintian Wu, Fei Wu
Huazhong University of Science and Technology
Hong Jiang
The University of Texas at Arlington

Abstract—**Copy-on-write (CoW) is a widely used technique to enable failure-atomic transactions for persistent memory (PM), which avoids the double-write problem in logging-based transactions. However, balancing copy granularity and metadata size for large objects is challenging. Oversized metadata introduces nontrivial overhead in tracking the out-of-place updated data blocks. To the best of our knowledge, prior work has not adequately addressed this metadata issue. This paper presents ReCoW, a kernel-user collaborative CoW transaction system that employs the hardware memory management unit (MMU) to accelerate the data referencing. ReCoW leverages virtual memory remapping to consolidate fragmented data into a contiguous virtual address space, thus significantly reducing the metadata sizes, which in turn substantially decreases metadata performance overhead. Our evaluation under real-world workloads indicates that ReCoW outperforms state-of-the-art competitors, PMDK, SpecPMT, and ArchTM by 3.90x, 2.76x, and 1.45x on average in throughput, respectively, while reducing metadata size to about 1% of that of comparable systems.**

I. Introduction

Persistent memory (PM) such as PCM [34], [51], [57], STT-RAM [27], ReRAM [46], and CXL-SSD [24] offers persistent data storage with significantly lower access latency than traditional storage devices. This advantage has made PM an intensive focus of research. However, when using PM, it is essential to ensure crash consistency. Modern processors employ an out-of-order execution strategy to improve instruction-level parallelism and evict dirty cache lines in an unmanageable manner for applications, thus violating the demanded PM persistence order. Therefore, it is crucial to develop a strict PM updating strategy that ensures atomic writes and prevents data corruption upon system failures.

There are two main types of crash consistency management schemes for PM. **On the one hand**, logging-based transactions [17], [39], [40], [45], [56] guarantee data consistency by appending the data to a write-ahead log (WAL) before updating the original data. However, such a double-write scheme significantly degrades performance because of limited PM write bandwidth [22], [54]. Furthermore, when updating the original data, there is typically a higher frequency of random writes. This inherently diminishes the efficiency of write merging in PM-internal buffers, such as XPBuffer [38], [48], [55] in Optane memory and the DRAM buffer [24] in CXL-SSDs. **On the other hand**, copy-on-write (CoW) transactions [13], [31], [36], [52] implement an out-of-place updating strategy and track the locations of new data using mapping info in CoW metadata, thereby avoiding the double-write problem. The out-of-place updating strategy also converts random updates to sequential PM writes. Upon any failure, the CoW transactions rely on the untouched original data for recovery.

Since CoW transactions are more PM-friendly for their higher write efficiency, prior works [1], [5], [52] have employed CoW or similar variants to optimize PM transaction memory write performance. However, they overlook the additional metadata overhead introduced by CoW. In logging-based transaction systems, data addresses are derived by adding data offsets to a base address pointer, as these systems perform in-place updates within a pre-allocated memory space, ensuring that the latest data location remains unchanged. In contrast, CoW transactions perform out-of-place updates, leading to data fragmentation and dispersion. As a result, they must rely on mapping info in metadata to track the locations of updated data, involving a query-based offset translation strategy to locate the correct addresses for data access. This additional metadata overhead incurs both increased access latency and significant memory consumption.

Two unavoidable factors further exacerbate this metadata overhead in CoW transactions. **On the one hand**, each transactional data access requires two metadata access, resulting in extra latency on the critical path. Specifically, metadata is first accessed to retrieve the latest data location, followed by data access and migration. During the commit phase, a second metadata access ensures thread safety. Each metadata access—whether hash-based or tree-based—typically involves multiple memory accesses, unlike data copying, which usually requires only one. **On the other hand**, CoW metadata grows rapidly, with a much higher metadata-to-data ratio than other systems, leading to considerable memory consumption and lower cache hit rates. For example, in file systems, metadata tracks 4KB blocks, representing about one-thousandth of total storage. However, in-memory applications typically involve many fine-grained data updates, e.g., at a cache line granularity (64 bytes). To enhance copy efficiency, data blocks are often aligned with hardware write units, such as processor cache lines or PM buffers (e.g., 256 bytes [8]). This alignment can result in metadata sizes reaching up to one-sixth of the data size [52]. For small metadata sets, the overhead remains minimal as processors can cache them. However, as PM scales to several terabytes, metadata overhead can exceed data copying costs, becoming the primary performance bottleneck.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

Our experiments show that metadata overhead can be up to twice that of data copying.

To reduce metadata size and mitigate its associated overhead, several straightforward approaches can be considered: (1) coarse-grained tracking [5], [25] and (2) an asynchronous copy-back strategy [52]. **First,** coarse-grained tracking reduces metadata size by copying larger chunks of PM pages at once. This approach minimizes the number of metadata entries needed, as each entry covers a larger data block than fine-grained tracking. However, this causes substantial write amplification when handling frequent fine-grained updates. **Second,** an asynchronous copy-back strategy asynchronously moves out-of-place data to its original location as updates accumulate. When data is initially allocated contiguously, this approach maintains data continuity and reclaims invalid metadata mappings. However, it reintroduces the double-write problem—requiring one write for the CoW operation and another for the copy-back process—which increases write overhead, risks saturating PM bandwidth, and shortens its lifespan. In summary, such naive solutions to reduce metadata size often introduce significant costs that may outweigh their intended advantages.

In this paper, we present ReCoW, a kernel-user collaborative CoW transaction mechanism that leverages virtual-to-physical remapping to perform data copying virtually rather than physically during transactional writes. Modern processors widely implement the memory management unit (MMU) to accelerate virtual memory (VM) address translation. Assuming that the data is contiguously allocated on the virtual memory, then the MMU can enable fast offset-based address translation. However, contiguous virtual address spaces can be fragmented by CoW updates from different locations and contiguously physically rearranging the data blocks is costly. Our key insight is that address records maintained both by user space and kernel space become redundant after a CoW operation. Specifically, the user space's metadata lookup table and the kernel space's page table both record the data's location post-CoW. By remapping the original virtual addresses to the updated data locations, we eliminate the need for extensive user-space metadata, significantly reducing its size.

However, realizing ReCoW involves key challenges, such as granularity mismatches between kernel remapping and user writes, maintaining consistency during concurrent operations, minimizing remapping overhead, and ensuring efficient crash recovery (as detailed in Section III). We address these through targeted optimizations like hybrid metadata tracking and lazy batching. With ReCoW, we make the following contributions:

- We conduct in-depth experimental analysis and observe that the metadata overhead in CoW transactions not only becomes a major **performance bottleneck** but also imposes considerable **memory consumption**, as the data scale reaches the gigabyte level.
- We propose a virtual address defragmentation approach through page table remapping, effectively reducing metadata size and performance overhead. Additionally, we optimize ReCoW's design to enhance its performance further.
- We evaluate the performance of ReCoW, on both Optane PM and **CXL memory**, against state-of-the-art alternatives, and the results show an average improvement of 44.7%. Moreover, Unlike prior CoW systems that rely solely on user-space metadata, ReCoW offloads tracking to the kernel's page tables, reducing metadata size by up to 99%.

II. Background and Motivation

A. Persistent Memory

Persistent memories (PMs) provide durable semantics in main memory. They employ battery-backed DRAM or equip new memory technologies such as PCM [34], [51], [57], ReRAM [46], FeRAM [41], or 3D-XPoint [19]. Since the Intel Optane DC is the only available persistent memory product on the market, we use it as the default hardware model without the loss of generality. Its distinct characteristics bring several challenges.

The volatility-persistence boundary: Persistent memory moves the volatility-persistence boundary between the processor cache and the main memory. However, a processor's cache does not ensure updating consistency. Moreover, processors improve memory access performance by employing cache hierarchies and disrupting request orders with out-of-order executions. Thus, software relies on additional processor support to specify when to persist data. Since the Skylake architecture, Intel provides clwb instruction (in addition to clflush and clflushopt) to flush dirty cache lines and sfence instruction to conduct a persistent barrier. The flush and fence instructions ensure that the data stores reach the processor's asynchronous DRAM refresh (ADR) domain and guarantee persistent order. The ADR domain will be flushed to the NVDIMM within the hold-up time ($< 100\mu s$), thus surviving a power failure [55]. Measuring when a store physically hits the PM is difficult. However, when the PM's bandwidth is throttled, the explicit cache flush and fence instructions become expensive and consume thousands of CPU cycles [50], [53], [55].

The Post-Optane Evolution of Persistent Memory: Intel's discontinuation of Optane, driven largely by commercial factors, does not mark the end of persistent memory but rather a shift in its evolution. The PM landscape is now advancing along two distinct paths: the development of **new storage media** like PCM [34], ReRAM [27], and CXL-attached SSDs [28], [30]; and the rise of **new interconnects** like CXL [14], which enable disaggregated architectures such as memory pooling. Both paths reaffirm the critical need for crash consistency. While new storage media inherently require it, the demand is particularly acute in CXL-based disaggregated systems. Here, the significantly higher access latency compared to local NVDIMMs makes the write amplification of traditional consistency schemes (e.g., logging) a critical performance bottleneck. Moreover, compute and memory nodes can fail independently, meaning a transaction is only durable once its data is safely persisted on the remote memory node. This imposes software-level consistency demands functionally identical to those of traditional PM [6], [16], [59].

ReCoW's design directly addresses these emerging performance challenges. Its approach of providing low-overhead

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

Copy-on-Write (CoW) transactions is media- and interconnect-agnostic, preserving low write amplification—a decisive advantage in high-latency CXL environments. Our evaluation on CXL-attached memory (Section VII-E) validates this, confirming ReCoW as a vital and efficient solution for next-generation persistent memory systems.

Read and write asymmetry: What makes PM management costly is the fact that its write performance is often inferior to the read performance. Directly comparing read and write latency is challenging because of different latency-hiding techniques introduced by the processor cache and ADR domain. We focus more on the bandwidth. The aggregated read bandwidth of six Optane persistent memories can reach up to 39.4 GB/s, while the write bandwidth can only reach 13.9 GB/s [22]. Therefore, reducing write operations on the PM is critical. However, failure-atomic transactions for PM bring in additional writes. For example, logging-based transactions must write twice, and CoW transactions introduce write-inducing metadata maintenance overhead.

PM-induced amplification Due to high storage density, PMs usually have larger internal write granularity than a 64B cache line. To serve cache line-grained CPU writes, PMs employ a persistent buffer to allow read-modify-write, which introduces internal write amplification. For example, Optane persistent memory has a 256B access granularity called XPLine. It then utilizes an XPBuffer, approximately 16KB in size [38], [48], [55], to combine adjacent 64B writes. Note that such a hardware structure is not Optane-exclusive. The CXL-enabled SSD, which provides memory access semantics with flash memory, also relies on an internal DRAM buffer to fill the flash write granularity gap [33]. The internal persistent buffer plays a crucial role in absorbing small writes. Hence, many works promote XPLine-grained adjacent writes for different applications to better unleash the Optane memory bandwidth [11], [47], [49]. However, applying such a design in transactions requires more fine-grained data management, which could be overburdened.

B. Persistent Transaction Mechanisms

Logging-based transactions: In logging-based transactional memory [17], [20], [39], [58], crash consistency is maintained by utilizing write-ahead logs including undo logs and redo logs. In such systems, data updates are first written to the log before being applied in place. If a crash or power failure occurs during this process, the transaction can be restored using the log. Since the updates are made in place, logging-based transactions do not require the additional metadata management that is necessary for copy-on-write transactions. However, this approach introduces the issue of double writes—one to the log and one to the data itself—which is not ideal for PM with limited write bandwidth and endurance.

CoW-based transactions: Copy-on-Write (CoW) transactions [2], [9], [52] ensure crash consistency through out-of-place updates. As shown in Figure 1, taking a commonly used Optimistic Concurrency Control (OCC) [32]-based CoW transaction as an example (the process is similar for Two-Phase Locking (2PL) [3]), the update process during an OCC-based CoW transaction involves several steps. First, the metadata table is queried to obtain the latest data location and version (black ①). Then, the update is written to a separate location as a thread-local write (black ②). During the commit phase, the metadata version is checked for conflicts. If the transaction commits successfully, the metadata pointer is updated (black ③), and the metadata log is flushed to ensure crash consistency (black ④). In the event of a crash or power failure during this process, the old version of the data can be used for recovery. Similarly, when reading data, the metadata table must be queried to obtain the latest data location(blue ①) before reading from PM(blue ②). During the commit phase, the metadata must again be accessed to verify the absence of conflicts(blue ③). If snapshot-level isolation is used, the third step can be omitted. Compared to logging, CoW transactions avoid the problem of double writes and reduce random write by aggregation, making them more friendly to PM's write characteristics (write endurance). However, due to the out-of-place update scheme, CoW transactions must maintain a global metadata table to track the new data versions. Given that PM is byte-addressable and can scale to terabyte levels, combined with a very high metadata-to-data ratio, the global metadata table can become extremely large, making it difficult for the CPU cache to store it all, leading to significant metadata overhead in CoW transactions. We will evaluate the metadata overhead in section III.

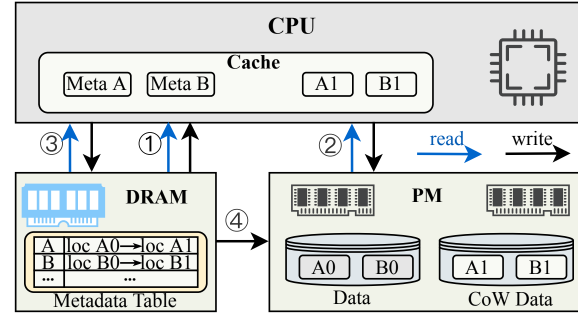


Fig. 1: OCC-based CoW Transaction

C. Virtual Memory Management

Modern operating systems leverage virtual memory mapping with PM direct access (DAX) in file systems, such as ext4-DAX and XFS-DAX, to bypass the page cache. DAX maps one or more virtual memory pages to PM via the *mmap* system call. Such primal PM support only introduces static one-to-one mapping. Recent work further customizes more advanced virtual memory mapping management for PMs. ctFS [35] is a PM file system that utilizes contiguous memory allocation to reduce the file indexing overhead and minimize fragmentation. To protect crash consistency, it writes to two shadow memory regions alternatively. Such a consistency protection scheme does not support efficient fine-grained data updating because frequent swapping of page tables imposes extra burdens on the OSes. In recent years, the XFS file system released its copy-on-write feature - Reflink [29] to support shallow copy for PM files. Reflink allows two or more files to share the physical PM page via DAX semantic. Upon any write to the shared page, the page fault handler in XFS will create

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

a private copy for the file being written. Although it protects consistent files sharing the PM pages, there is no support for transactional updating at fine granularity. In summary, virtual memory mapping already attracts attention in accelerating memory indexing and sharing; however, how to leverage it to improve transaction efficiency remains underexplored. This is because, fundamentally, DRAM's virtual memory management centers on 'caching and swapping', while PM's (especially with DAX) must shift to center on 'direct mapping and persistent ordering', requiring cacheline-level control of flushes, fences, and crash-consistent recovery mechanisms.

III. Motivation and Challenge

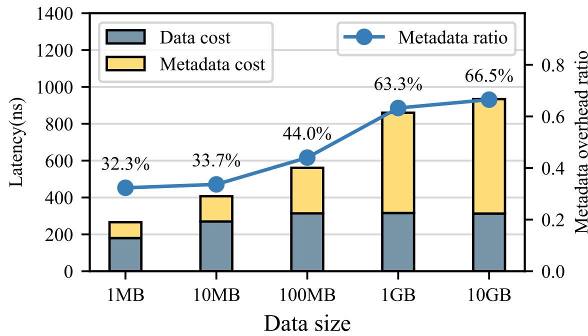
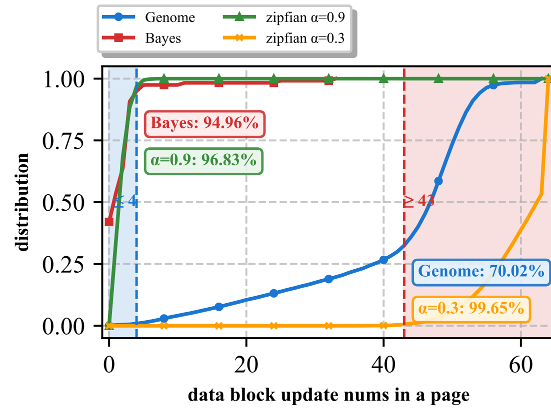


Fig. 2: CoW Metadata Overhead (L3 Cache Size = 36M). Metadata ratio refers to the proportion of metadata latency relative to the total latency, calculated as metadata latency / (metadata latency + data access latency).

While CoW transactions avoid the double-write problem of logging, they introduce a new bottleneck: metadata overhead. To track out-of-place updates, CoW systems must maintain and frequently access mapping metadata, especially in fine-grained workloads. As data scales up, the size of this metadata can far exceed processor cache capacity, becoming a major performance and memory consumption issue. We quantify this overhead in Figure 2, which shows the breakdown of CoW transaction latency with varying data sizes. As the metadata size grows beyond the cache (e.g., 100MB), metadata access latency dominates total transaction latency—up to 66.5%, roughly 2x the cost of data copying. This problem stems from two factors: (1) each transactional read/write requires at least two metadata accesses, and (2) the metadata-to-data ratio is particularly high at cache-line granularity, often reaching up to 1/6 of the data size. The metadata quickly outgrows the cache, leading to poor hit rates and high memory access latency. Our key insight is that CoW metadata and page tables both record the same information post-update: the physical location of the new data. Thus, by remapping the original virtual addresses to the new physical pages after out-of-place writes, we can eliminate the need for user-space metadata altogether. This remapping preserves the illusion of in-place updates while enabling fast access via simple offset calculation, effectively addressing the metadata overhead problem.

A. Grain mismatch behind remapping

A key challenge for ReCoW is the **granularity mismatch** between application writes and kernel remapping. Applications



Workload	Genome	Bayes	α=0.9	α=0.3
WA	1.51x	32.00x	30.11x	1.06x

Fig. 3: Cumulative distribution of pages by the number of updated cache lines under a different workloads.

perform fine-grained updates at cache-line granularity, whereas the kernel's remapping mechanism operates on coarse-grained 4KB pages. This disparity means that even if only a few cache lines are modified, the entire 4KB page must be consolidated and written to a new physical location before it can be remapped. This process leads to significant write amplification. To quantify this, we analyzed the update density per page under different workloads (Figure 3). The analysis reveals that updates in some workloads are typically sparse: over 95% of modified pages have four or fewer updated cache lines. This sparsity is problematic; for instance, in the STAMP [42] `bayes` workload, a naive remapping strategy results in a 30× write amplification. Conversely, for workloads with dense updates, the amplification is minimal. This variability necessitates an adaptive mechanism that can distinguish between sparse and dense update patterns. To address this, ReCoW employs a Hybrid Metadata Manager, detailed in Section V-A, which dynamically selects the optimal strategy to minimize write amplification.

B. Concurrent Problem during remapping

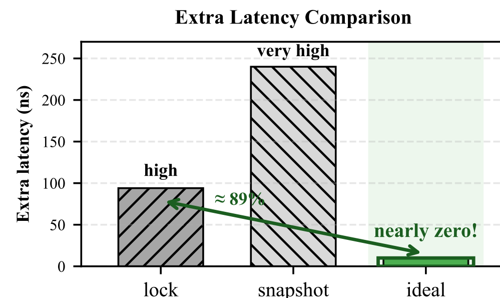


Fig. 4: Transaction Stall Latency During Remapping

ReCoW's collaborative nature, where both kernel-space remapping threads and user-space transactions access the same metadata, introduces a complex **concurrency control**

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

problem. As illustrated in Figure 4, traditional solutions are ill-suited for this cross-space environment. On the one hand, **cross-space locks** are notoriously complex to implement efficiently and can easily become a performance bottleneck, forcing user transactions to stall. On the other hand, **snapshotting** requires freezing the old metadata table (marking it as immutable) for the remapping operation, while directing any subsequent updates to a new table. This forces transactions to perform a costly "double-query" by checking both the old (frozen) table and the new table to locate the latest data, leading to unacceptable application stalls. Both approaches lead to unacceptable application stalls. The key challenge, therefore, is to design a lightweight, non-blocking concurrency mechanism. ReCoW achieves this through its `Concurrency Coordinator`, a lock-free state machine detailed in Section V-B, which safely coordinates kernel and user spaces without stalling foreground transactions.

C. Overhead introduced by remapping

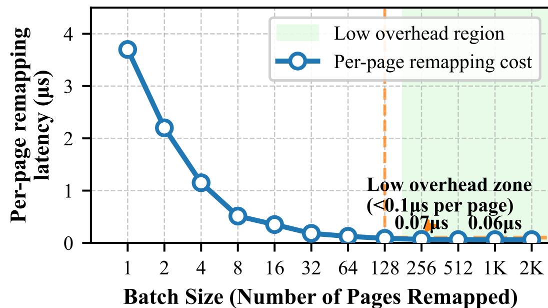


Fig. 5: Remapping Latency under different Batch Size.

While kernel-space remapping is a powerful tool, its execution is far from free, posing a significant performance challenge. The overhead stems from two primary sources: substantial write amplification from page-level consolidation and the high runtime cost of the remapping process itself. The runtime overhead is particularly severe. Each `remap` process involves a costly kernel entry, virtual memory subsystem updates, and TLB flushes. To quantify this, we measured the latency of the remapping operation on 32 threads, as shown in Figure 5. The results reveal that remapping a single page is exceptionally expensive, taking nearly **4 µs**. Even in small batches, the per-page cost remains in the microsecond range. Under concurrent workloads, frequent invocations of such a heavyweight operation would create a severe scalability bottleneck, drastically increasing transaction latency and throttling overall throughput. We introduce ReCoW's approach to handling this challenge in Section V-C.

IV. Design of ReCoW

A. Overview

This paper proposes ReCoW, a kernel-user collaborative system that provides low-overhead, copy-on-write (CoW) based persistent transactions. The core idea of ReCoW is to leverage **kernel-level page table remapping** to eliminate the metadata overhead inherent in traditional CoW schemes. By remapping virtual addresses to new physical locations after an update, ReCoW creates an "in-place update illusion," which maintains a contiguous virtual address space and removes the need for expensive user-space metadata lookups. As shown in Figure 6, this is orchestrated by four key components that work in concert:

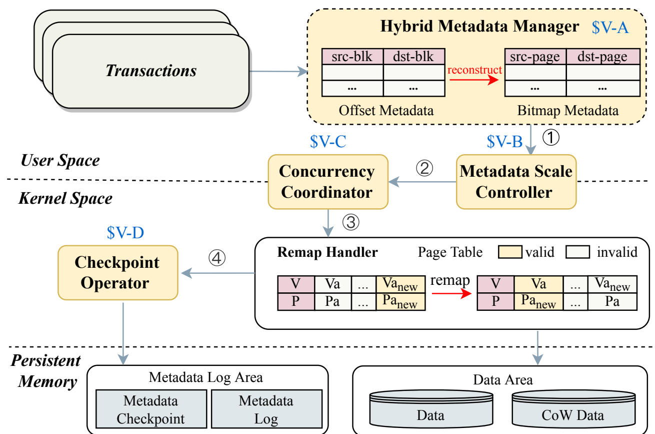


Fig. 6: An Overview of ReCoW Design, the remap syscall is triggered when metadata exceeds threshold

- **Hybrid Metadata Manager:** Manages both fine-grained (cache-line) and coarse-grained (page-level) updates. It prepares metadata in a remap-compatible format, ensuring seamless coordination between user-space writes and kernel-space remapping.
- **Concurrency Coordinator:** Ensures that background remapping operations do not interfere with foreground user transactions. It uses a lock-free mechanism to maintain data consistency and high performance during concurrent access.
- **Metadata Scale Controller:** Implements a lazy, batch-remapping policy. It monitors the metadata size and triggers remapping only when a threshold is reached, balancing the benefits of reduced metadata against the overhead of system calls.
- **Checkpoint Operator:** Provides fast and consistent recovery after a crash. It creates comprehensive checkpoints that capture both user-space and kernel-space state, minimizing log replay time.

Figure 6 also illustrates how these modules work together through a single remapping process. The Hybrid Metadata Manager processes transaction commits, converting relevant metadata into a form compatible with remapping. As the transaction progresses, the metadata size increases. The Metadata Scale Controller monitors this growth and triggers remapping when the threshold is reached(①). During remapping, the Concurrency Coordinator maintains consistency with ongoing transactions(②). After the remapping is complete(③), the Checkpoint Operator creates a metadata checkpoint to speed up recovery in the event of a crash(④)

B. Contiguous Memory Reallocation

The primary challenge in traditional CoW is the metadata overhead required to track out-of-place updates. While sim-

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

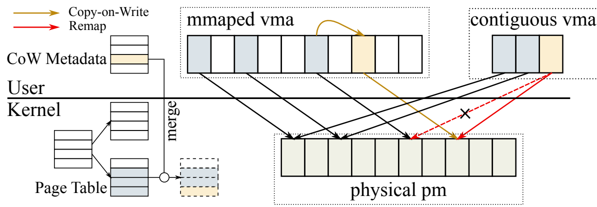


Fig. 7: Remapping-based Defragmentation. ReCoW allocates a contiguous VMA and remaps virtual addresses to new physical pages after CoW updates, enabling metadata-free access and in-place illusion.

ple contiguous memory allocation can reduce this overhead initially, it inevitably succumbs to fragmentation as updates accumulate, leading to a metadata lookup bottleneck.

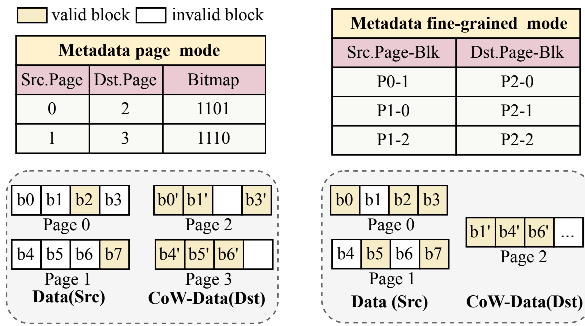
ReCoW's core innovation is to eliminate this user-space metadata lookup entirely by leveraging **virtual memory remapping**. Our key insight is that by manipulating page table entries, we can make a *virtually contiguous* address space point to *physically fragmented* data blocks. This creates the powerful **illusion of an in-place update**: applications interact with a stable and contiguous memory region, while physically, the data is safely updated out-of-place, thus preserving the benefits of CoW without its metadata cost.

This mechanism, illustrated in Figure 7, is implemented through a set of new syscalls and a two-VMA approach:

1) **Creating a Stable Virtual View:** When an application opens a CoW object (e.g., a B+ tree), it invokes a new `obj_map` syscall. This call allocates a read-only, contiguous virtual memory area (the *contiguous vma*), which provides a stable, direct-access view of the object's data. Applications can now access data via simple pointer arithmetic within this VMA, completely bypassing metadata lookups.
2) **Performing Out-of-Place Writes:** All transactional writes are performed out-of-place, with new data blocks allocated from a separate, general-purpose writable memory region (the *mmapped VMA*).
3) **Atomically Remapping the View:** Once the transaction commits, an `obj_remap` syscall is called. This syscall passes the physical addresses of the new data blocks to the kernel. The kernel then atomically updates the page table entries of the *contiguous vma*, redirecting it to point to these new physical pages. From the application's perspective, the data appears to have been updated in-place.

Finally, an `obj_unmap` syscall releases all associated resources. Through this process, the application's view is seamlessly switched to the latest data version, rendering explicit CoW metadata tables unnecessary for data access.

This fundamental remapping technique is most efficient for page-aligned updates. Section V will detail the additional mechanisms ReCoW employs to make this approach practical and efficient for fine-grained, real-world workloads.



(a) Page-grained Tracking: Random Writes for Minor Updates.

(b) Fine-grained Tracking: Merged Sequential Writes.

Fig. 8: Hybrid-granularity Tracking Metadata.

V. ReCoW Optimizations

A. Hybrid Metadata Manager

To resolve the core challenge of **granularity mismatch**, ReCoW implements a **Hybrid Metadata Manager**. This component enables ReCoW to handle both fine-grained and coarse-grained updates efficiently by adaptively switching between two metadata tracking strategies, thereby minimizing write amplification. As illustrated in Figure 10, a flag bit distinguishes between two metadata formats. Both formats use the address of the original data page (the "source page") as the key but differ in their value handling. For frequently updated pages, we employ page-granularity metadata to track the address of the new page (the "destination page") created after a CoW operation. A 64-bit bitmap indicates whether each committed cache line remains on the source page or has been moved to the destination page. However, if only a few cache lines are updated, this approach can lead to random partial-page writes, increasing PM's internal write amplification (see Figure 8a). For less-frequently updated pages, we use fine-grained metadata to track individual cache line addresses. As shown in Figure 3, 96.6% of pages in the Zipfian workload have fewer than 4 updated cache lines, so fine-grained metadata defaults to tracking four cache lines to align with the random update pattern. This ensures efficient use of metadata entries for sparse updates. When more than four cache lines are updated within a page, we convert to page-granularity metadata by copying the associated cache lines to a new page. As shown in Figure 8b, fine-grained tracking consolidates writes on less-frequently updated pages into sequential streams, improving PM bandwidth.

Defragmentation: Two different defragmentation processes are invoked when the metadata table becomes full and probing an open slot becomes expensive. For the page-granularity tracked pages, we check the bitmap to calculate the number of committed cache lines in source and destination pages. We then move all the data into the one with more cache lines. Finally, if the destination page carries all the data, we mark the corresponding metadata entry as owned by the kernel before invoking the remap syscall. For the fine-grained tracked pages, we only perform user space defragmentation, which copies the out-of-place updated cache lines back to the source page.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

B. Cost-effective Consistency Coordinator

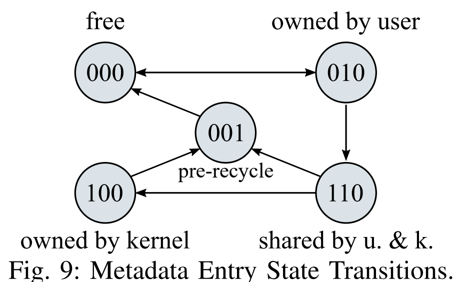


Fig. 9: Metadata Entry State Transitions.

To resolve the complex **kernel-user concurrency problem**, ReCoW employs a **State-Machine based Coordinator** that enables low-overhead, non-blocking coordination. Unlike traditional locks (which cause stalls) or snapshotting (which incurs query overhead), our approach ensures that user transactions and kernel threads can operate concurrently without blocking one another. As shown in Figure 9, each metadata entry follows a 5-state state machine. Initially, an entry is in the *free* state. When a new entry is inserted, the indexing mechanism finds a free slot, atomically marks it as *owned by user*, and writes the data. Before invoking the remap syscall, candidate slots are transitioned to the *shared by user and kernel* (u. & k.) state. This makes them read-only for user space, allowing the kernel to safely merge mappings into the page table while user transactions continue to access them. If a new write arrives for an entry in this shared state, the indexing mechanism simply finds a new slot via linear probing. The old entry is then transitioned to *exclusively owned by kernel*, allowing the kernel to complete its operation without interfering with the new write. To ensure consistency with in-flight transactions, metadata entries are lazily reclaimed. After remapping, the kernel transitions the entry to the *pre-recycle* state. This state prevents new transactions from accessing it while allowing any existing transactions that already hold a reference to safely complete their operations. After the remap syscall finishes, the defragmentation thread waits for all such active transactions to conclude before finally recycling the metadata entry and its associated data page. As detailed in Figure 10, we use atomic operations on state bits to manage ownership and reclamation, ensuring safe, concurrent access throughout the entry's lifecycle.

We employ another lock flag bit and a version number to implement the important control of concurrent updates to metadata from the user space. All the flag bits and the version for an entry reside in an 8B integer and thus can be updated atomically via a compare-and-swap (CAS) instruction. When accessing a page, we first find a matching source page ID with a valid flag status in the metadata table. Note that the entries owned by the kernel won't be locked and updated by users. Then, we set the lock flag via a CAS instruction to ensure that no other threads are accessing the entry. If the CAS operation fails, the transaction is aborted to ensure consistency. Otherwise, the system proceeds with the update. Finally, the version is committed, and the lock is released via another CAS instruction.

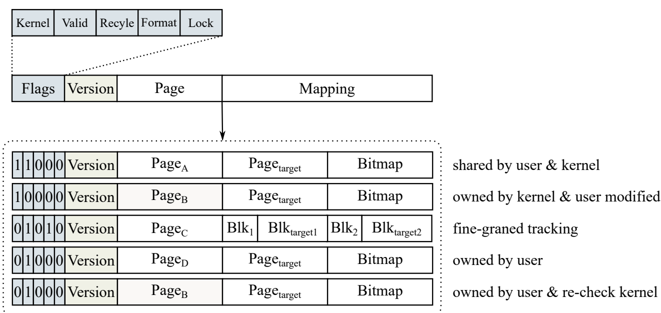


Fig. 10: Metadata Format.

C. Metadata Scale Controller

To mitigate the high remapping overhead identified in Section III-C, ReCoW introduces a **Metadata Scale Controller** that applies a *lazy* and *batched* consolidation policy. Instead of remapping immediately, the controller defers the operation, triggering it only when the metadata size exceeds a configurable threshold (e.g., 8MB) to keep lookups cache-resident and fast. Once triggered, the controller does not remap pages indiscriminately. It selectively scans the metadata and gathers only the pages most suitable for remapping—specifically, **those already promoted to the page-grained tracking format** by the Hybrid Metadata Manager. These remap-friendly candidates are then processed in large batches. As shown in Figure 5, batching over 128 pages at once is highly effective, reducing the average cost to under 70ns per page by amortizing syscall and TLB flush overheads.

All remapping is offloaded to background threads, ensuring non-blocking execution for foreground transactions. This design balances two competing goals: maintaining compact metadata for low-latency access and minimizing kernel interaction to sustain throughput. The consolidation threshold is tunable for different hardware and workload profiles.

D. Kernel-User Collaborative Checkpoint and Recovery

To ensure fast recovery, ReCoW implements a **Kernel-User Collaborative Checkpoint** mechanism. While append-only logging is efficient for runtime durability, its unbounded growth can lead to unacceptably long recovery times. A naive user-space-only checkpoint is insufficient because ReCoW's persistent state is *bifurcated*—split between user-space metadata and kernel-managed page mappings. A purely user-level snapshot is blind to this kernel state, necessitating our collaborative approach.

Checkpoint Creation: The process begins by briefly pausing the background remapping thread to ensure a stable state. Then, two components are captured: a **User-Checkpoint (User-CP)**, which is a consistent snapshot of the user-space metadata table, and a **Kernel-Checkpoint (Kernel-CP)**, a compact summary of the virtual-to-physical page mappings obtained via a read-only kernel interface. Both are persisted to PM, after which a single directory record containing pointers to them is written atomically to commit the entire checkpoint.

Fast Recovery: This design enables a fast, two-phase recovery. The system first restores from the latest valid checkpoint by loading the User-CP to restore baseline metadata and using

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

the Kernel-CP to reconstruct the page table structure. Second, it performs a bounded replay of only the small portion of the log created after the checkpoint. For robustness, if the checkpoint is invalid, the system falls back to a traditional log-only recovery, guaranteeing correctness.

VI. ReCoW Implementation

A. Memory Management

ReCoW applies different memory management strategies for page- and cacheline-granularity metadata. For page-granularity tracking, we adopt per-page allocation within a contiguous virtual memory region, avoiding variable-sized allocation overheads seen in systems like ArchTM. For fine-grained tracking, we implement a sequential ring buffer on PM to manage cacheline-level updates, enabling efficient allocation and reclamation.

Two background GC threads handle metadata cleanup: one for page-granularity data, and another for cacheline data. In page-level GC, ReCoW merges updated cache lines from source and destination pages into the one with more valid data. If the source page is selected, the destination page is recycled directly; if not, a remap syscall is issued before reclaiming the source page. All user-space metadata operations respect concurrent access control to avoid conflicts. Free pages are managed via a lock-free FIFO queue for fast reuse.

B. Concurrency Control

Existing persistent transactional memory relies on software or hardware methods to provide concurrency control. ReCoW is designed to be compatible with existing software-based concurrency mechanisms. Aligning with prior works [18], [52], our implementation adopts a concurrency scheme based on **snapshot isolation**. During the transaction's read phase, ReCoW determines whether to abort or read a specific version based on the timestamp in the metadata. If no metadata entry is found, ReCoW directly reads the data from its original address. During the write phase, ReCoW maintains a write set and resolves conflicts by locks during the commit phase. To enhance performance, ReCoW leverages RDTSCP [12] for timestamp. If a transaction is successfully committed, ReCoW sequentially flushes the metadata log to ensure crash consistency and updates the metadata in DRAM.

C. Using ReCoW Example

To ensure crash consistency, applications must utilize the APIs provided by ReCoW. Our implementation consists of a user-space library and a kernel component, which we designed as a self-contained, loadable module to ensure system safety and simplify adoption.

Figure 11 takes a hash table application as an example. The application should allocate the table using the ReCoW allocation API, enabling the underlying data blocks to be managed within ReCoW's library. Upon updating a key, it passes entry-ptr to call the ReCoW Tx-Read API to find the real entry-ptr and utilizes the real entry-ptr to check the entry's key. During this process, ReCoW queries the metadata lookup table in DRAM to locate the latest virtual address of the PM data. Once the key matches, the hash table uses the Tx-Write API to update the value by CoW and ReCoW will update the CoW mapping in DRAM metadata. Upon a successful transaction commit, the Metadata Scale Controller checks if the accumulated metadata has reached its size threshold. If it has, the controller triggers a background remapping process to consolidate the updates and reclaim metadata space.

	Hash Table	ReCoW Code & Operation
1	`table_ptr= alloc(table_size)`	`table_ptr = ReCoW.alloc(table_size)`
2	`entry_ptr = table_ptr + hash(key)`	use *obj_map* to alloc a continuous VMA
3	*tx_begin*	`ReCoW_CTX = ReCoW.new_tx_ctx()`
4	`value = entry_ptr->key == key ? entry_ptr->value : 0` *//find the match entry*	`entry_ptr = ReCoW_CTX.read(entry_ptr)`
		query metadata to locate entry_ptr and check entry value
5	`entry_ptr->value = new_value` *//update value*	`entry_ptr= ReCoW_CTX.write(entry_ptr, new_value)//CoW and update metadata`
		`ReCoW_CTX.commit()`
6	*tx_end*	`if(ReCoW_CTX.needs_remap()){Remap_triggered();}`

Fig. 11: Recow-based hash table

VII. Evaluation

To demonstrate the effectiveness of ReCoW, we evaluate and compare it against four existing PM transaction mechanisms:

- **PMDK**: Persistent Memory Development Kit, a toolkit designed by Intel to efficiently develop applications in persistent memory environments. It offers an undo logging-based transaction model for crash consistency. We use a read-write lock to provide thread isolation.
- **SpecPMT**: SpecPMT [56] is a state-of-the-art logging-based transaction. It advances log writes to the previous transaction, reducing the extra fence of the write-ahead log.
- **ArchTM**: ArchTM [52] employs a flexible variable-sized block (denoted as objects in the original paper) CoW strategy. It allocates and copies blocks in PM but overlooks the CoW metadata problem.
- **ReCoW-Bitmap**: We also compare ReCoW with ReCoW-Bitmap, a ReCoW implementation without hybrid metadata tracking. ReCoW-Bitmap consistently records the latest data location using a bitmap.

To showcase ReCoW's performance, we design a set of benchmark and real-world workload tests. We run Yahoo Cloud Serving Benchmark (YCSB) to model real-world application workloads under different average request sizes and varying numbers of working threads. We employ the standard transactional benchmark, STAMP [42], to evaluate the performance of different designs on real-world transaction processing applications.

A. Experimental Setup

Hardware Platform. All experiments are conducted on a dual-socket server featuring two Intel(R) Xeon(R) Gold 6230R CPUs, creating a Non-Uniform Memory Access (NUMA) architecture. Each CPU has 26 cores running at a base frequency of 2.10GHz, with 1.625MB L1, 13MB L2, and 36MB L3

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

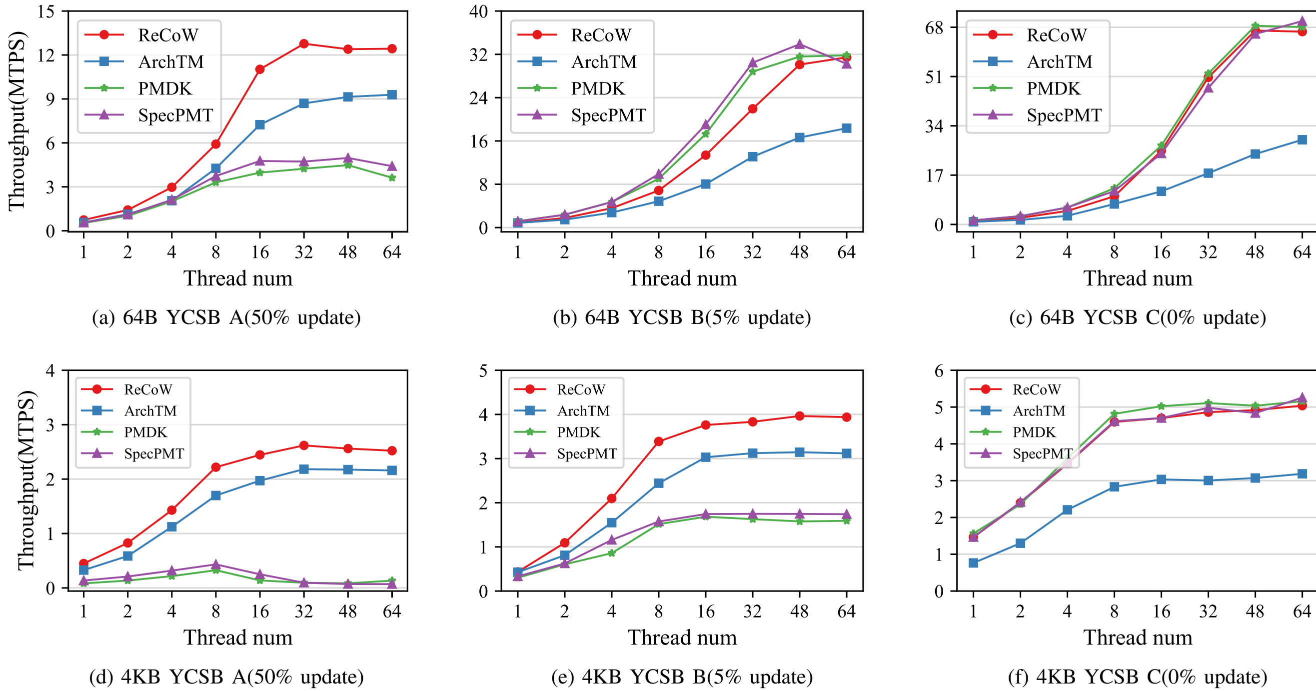


Fig. 12: Throughput under YCSB workloads.(Higher is better)

cache per socket. The system is populated with 96 GB of DRAM (6x 16GB DIMMs) and 1 TB of Intel Optane DC Persistent Memory (4x 256GB DIMMs). The Optane PM is configured in App Direct mode. For specific experiments involving memory expansion, we utilize a CXL 1.1 FPGA development board connected to the host, which provides an additional (6x 16) GB of DRAM.

Software Environment. The server runs Ubuntu 22.04 LTS. We use a custom-compiled Linux kernel (v6.5) to integrate our proposed system call modifications. The persistent memory is managed by an ext4 file system mounted with the Direct Access (dax) option, which allows memory-mapping files directly into the application's address space. All programs are compiled using GCC 11.4.0 with the -O2 optimization level.

Hardware Timestamp To ensure a consistent ordering of transactions across multiple cores, we rely on the hardware Time Stamp Counter (TSC). Our platform's CPUs feature an invariant TSC synchronized across all cores using ORDO [26]. This guarantees that timestamps are monotonic and consistent system-wide, which is crucial for correctly ordering concurrent operations. We use the 'rdtscp' instruction to read the TSC, which prevents instruction reordering and provides a reliable timestamp for determining the persistence order of transactions.

B. YCSB Benchmark

To evaluate the performance of ReCoW in a multi-thread environment, we conduct standardized tests using the YCSB benchmark and compare the throughput with those of ArchTM, PMDK, and SpecPMT. The tests comprised three update ratios, designated as A (50% update ratio, an update-intensive workload), B (5% update ratio, a read-intensive workload), and C (0% update ratio, a read-only workload). All these tests follow the Zipfian distribution and we choose the hash table as the basic key-value index engine. At the same time, we set the average request size to 64B and 4KB to emphasize hybrid metadata tracking and understand the effectiveness of kernel remapping. Figure 12 (a)-(f) respectively shows the transaction throughput curves of each scheme, as the number of threads changes under different levels of request granularity and various YCSB workloads.

In the 64B YCSB A test, ReCoW exhibits a 33% improvement over ArchTM, attributed to its reduced metadata overhead. Moreover, it surpasses PMDK by 3.43x and SpecPMT by 2.82x, highlighting its enhanced write performance. In the 64B YCSB B test, ReCoW achieves a 1.92x speedup over ArchTM and 83.2%, 76.0% throughput of PMDK, SpecPMT at 32 threads, respectively. This indicates that the CoW metadata overheads reduce the overall throughput of read-dominant workloads compared to the log-based strategies, which update in place. When the number of threads increases to 64, ReCoW achieves performance comparable to PMDK and SpecPMT due to its avoidance of double writes. In the 64B YCSB C read-only workload, ReCoW achieves 94.8% and 97.7% throughput of PMDK and SpecPMT, respectively. Since we expect an empty CoW metadata table under read-only workloads, the results are collected after initialization, followed by manually triggered GC. The throughput of ArchTM is only 44% of ReCoW, due to the metadata overhead introduced by out-of-place updates during the initialization phase. In summary, under the 64B YCSB test, ReCoW achieves throughput comparable to log-based transactions in read-intensive workloads, while delivering the best performance among state-of art works in write-intensive workloads.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

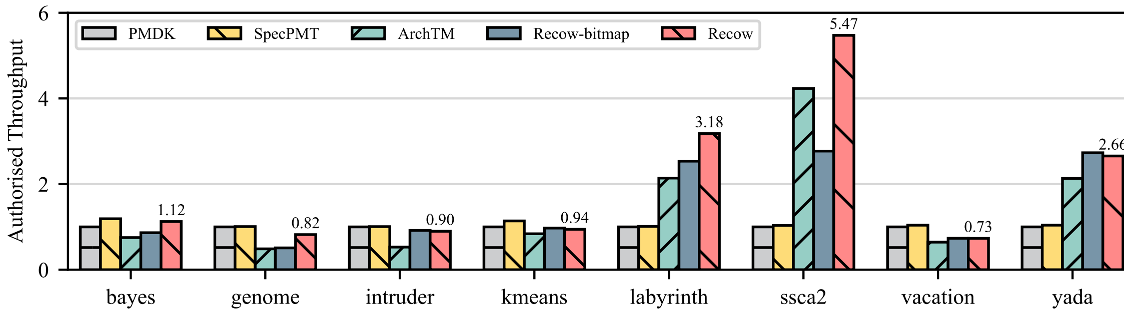


Fig. 13: Speedup over PMDK under different STAMP applications.(Higher is better)

In the 4KB YCSB A test, ReCoW achieves the highest throughput, demonstrating a performance improvement of 1.17x, 5.87x, and 7.88x over ArchTM, SpecPMT, and PMDK. It is noteworthy that when the thread count exceeds 8, the bottleneck shifts to PM because ReCoW hits its' maximum write bandwidth. Additionally, while 4KB requests are large and sequential, an increase in the number of threads implies an escalation in the intensity of concurrent writes, which slightly reduces the effectiveness of PM's write-combining buffer (XPBuffer) [11], [47], [49]. In the 4KB YCSB B test, ReCoW attains a bandwidth of 1.26x that of ArchTM. ReCoW achieves throughput that is 2.26x and 2.47x higher than SpecPMT and PMDK, respectively, where write performance is the primary bottleneck.

C. STAMP Benchmark

Application	Tx Num	Update Ratio	Remapping Ratio
bayes	10,240	0.37	0.02
genome	3,841,536	0.08	0.81
intruder	2,222,592	0.15	0.91
kmeans	699,072	0.48	0.33
labyrinth	40,448	0.49	0.17
ssca2	548,736	0.66	0.41
vacation	1,697,792	0.04	0
yada	6,049,280	0.37	0.01

TABLE I: Scale, update ratio and remapping ratio of Stamp applications, remapping ratio means the ratio of metadata is handled by the *obj_remap* syscall.

We employ the STAMP benchmark [42] to evaluate ReCoW under transaction applications. All evaluations spawn 32 threads. Figure 13 shows the speedup of different strategies over PMDK. Furthermore, Table I shows the configurations of STAMP applications and the ratio of metadata entries processed by *obj_remap* syscall applications.

ReCoW considerably outperforms log-based transactions when updating a large amount of data, because of the elimination of double writes in logs and the optimization with sequential write merging. For example, ReCoW demonstrates a substantial performance improvement, ranging from 2.66x to 5.47x, in write-intensive scenarios. And in read-intensive workloads such as kmeans, intruder and genome, ReCoW can achieve throughput comparable to that of log-based transactions and is better than ArchTM. In vacation, which performs many small updates scattered over a wide range of memory, ReCoW achieves 0.73x and 0.70x throughput of PMDK and SpecPMT respectively.

Compared to ArchTM, ReCoW achieves superior overall performance by adopting *obj_remap* syscall, effectively reducing the scale of the metadata table. Table I reveals that the metadata table entries cleaned by *obj_remap* syscall are nearly negligible in bayes, vacation, and yada. In these three applications, the number of metadata entries in ReCoW and ArchTM is similar. Consequently, the throughputs of ReCoW and ArchTM are comparable due to the similar indexing overhead in ReCoW and ArchTM at this time. In other applications, ReCoW achieves superior throughput to ArchTM, ranging from 1.48x to 1.72x. By employing a hybrid metadata tracking scheme, ReCoW effectively diminishes metadata scale, leading to lower write amplification. Moreover, it optimizes global metadata table queries through virtual space remapping, showcasing advantages over ArchTM in both read- and write-intensive workloads.

Furthermore, we conduct a comparison between ReCoW and ReCoW-Bitmap to assess the influence of the metadata tracking scheme. In read-intensive workloads, the differences between ReCoW and ReCoW-Bitmap are minimal, less than 3%. However, in random write workloads, ReCoW outperforms ReCoW-Bitmap by 1.67x and 1.97x in genome and ssca2, respectively, as ReCoW utilizes fine-grained page tracking to consolidate random writes into sequential writes.

D. Sensitivity Study

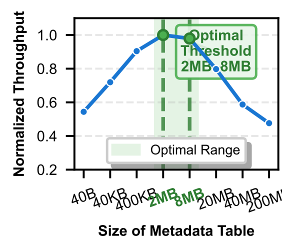


Fig. 14: Throughput with Different Global Metadata Sizes

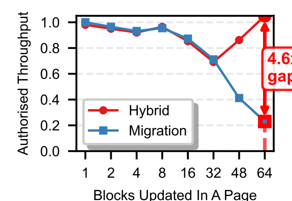


Fig. 15: Influence of Hybrid GC and Migration-Based GC on Throughput

Threshold Setting ReCoW employs a lazy-merge strategy to reduce the overhead of frequent merge syscalls. A key question is how to determine the optimal merge threshold, as this directly impacts when the background merge thread is

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

triggered and affects overall system performance. To explore this, we run experiments on the YCSB A test to assess performance sensitivity to varying threshold values. As shown in Figure 14, we set the merge threshold between 1 and 5 million (40B to 200MB) and compare throughput in the hash table. With a threshold of 1 (40B), throughput is lowest due to frequent system calls. As the threshold increases, throughput improves up to 400KB, thanks to the lazy-merge scheme reducing remapping overhead. However, beyond 40MB, performance drops as remapping becomes rare and cache misses increase during lookup. Thus, the merge threshold significantly affects ReCoW's transaction performance, influenced by the workload and L3 cache size. For optimal performance, keeping ReCoW's metadata under 4MB is crucial.

Migration and Remapping: To compare the impact of hybrid-based GC used by ReCoW and migration-based GC on the performance, we conduct an experiment with a write-intensive workload on 48 threads. We control the number of data blocks (64B) updated per page varying the count from 1 to 64. As shown in Figure 15, when the number of data blocks updated within a single page is less than 32, ReCoW employs a user-space migration GC strategy, resulting in performance that is consistent with that of the migration GC. However, when the number of updated data blocks exceeds 32, ReCoW leverages remapping which can migrate less data. This approach reduces the write amount to PM by the background GC threads. Consequently, hybrid-based GC can achieve a throughput that is 4.6x that of the migration-based GC.

Memory Usage Analysis: ReCoW employs a strategic trade-off to conserve scarce DRAM resources: it accepts a modest, transient peak PM overhead ($\approx$13%, due to momentarily retaining old data versions for crash consistency) to achieve an order-of-magnitude reduction in DRAM footprint. Our experiments with 100 million records (YCSB) demonstrate this efficiency: while ArchTM consumes up to 2.2 GB of DRAM metadata, ReCoW slashes this requirement to just 0.5% (YCSB-A) and 0.06% (YCSB-C) of ArchTM's usage. This validates ReCoW's design philosophy of leveraging abundant PM capacity to free up critical DRAM in tiered-memory environments.

Latency Breakdown: To better understand the performance improvements brought by defragmentation in the virtual address space, we compare the breakdown of transaction latency between ReCoW and ArchTM in a YCSB B benchmark with 5% updates, as shown in Figure 16. Due to defragmentation, ReCoW reduces the size of the metadata, which in turn lowers the overhead associated with metadata querying and updating. As a result, metadata overhead accounts for only 23% of the transaction latency under read-intensive workloads. In contrast, ArchTM, with its large metadata footprint, sees metadata overhead constituting 57% of its latency.

Hybrid Metadata setting: To configure the optimal granularity for our hybrid metadata, we conducted a sensitivity analysis on STAMP Bayes test for the metadata change threshold setting. The figure shows that normalized throughput peaks at 4 cachelines (CL). This trend is explained by a trade-off: at smaller change setting (e.g., 1-2 CL), the aggregation effect is insufficient to smooth random writes, failing to effectively amortize per-operation overheads. Conversely, as the change setting increases beyond 4 CL, the performance declines sharply due to the high conversion overhead required to process larger, complex metadata blocks. Consequently, 4 CL provides the best balance, and we adopt it as the default setting for our hybrid metadata operations.

Metadata format Change Over time: Figure 20 illustrates the dynamic metadata distribution under the YCSB-A workload. While initially dominated by Page Metadata, the share of fine-grained Cacheline Metadata steadily increases to become the dominant form, driven by the workload's high volume of random writes. The periodic fluctuations in the Cacheline Metadata proportion reveal a background remap process. This mechanism consolidates accumulated Cacheline Metadata into more compact Page Metadata, causing a temporary dip in its share. Following consolidation, Cacheline Metadata begins to re-accumulate as writes continue. This cyclical behavior demonstrates our system's strategy to balance the immediate performance of fine-grained writes with the long-term space efficiency of coarse-grained storage.

Grain Test: We also test ReCoW's performance at granularity levels other than 64B and 4KB. Specifically, under 32 threads, we evaluate the performance at the 512B granularity using the YCSB A benchmark. As shown in Figure 17, ReCoW achieves throughput that is 1.42x, 6.46x, and 5.91x higher than ArchTM, PMDK, and SpecPMT, respectively. As the data granularity increases and data copying gradually becomes the primary bottleneck, ReCoW continues to maintain excellent performance.

E. CXL-attached memory test

To evaluate ReCoW beyond Optane PM, we conducted a 64B YCSB A test under 32 threads on CXL 1.1 expanded memory. Here, PMDK represents the standard undo-log transaction, where data flushes are enforced using `clwb` and `fence` instructions. The results of these tests are illustrated in Figure 18. Due to the superior latency and bandwidth performance of CXL memory, ReCoW outperforms PMDK, SpecPMT, and ArchTM by 35%, 31%, and 39%, respectively. In B test, ReCoW is 20% lower than logging-based transaction. The reason lies in the fact that ReCoW still necessitates accessing the metadata table, while log transactions can read directly. In CXL memory, CoW transactions maintain their write performance advantage over logging-based transactions, and the metadata overhead issue remains a significant challenge even in CXL expanded memory.

F. Checkpoint and Recovery Time

The checkpoint recovery time depends on the frequency of checkpoint operations. However, frequent checkpoints will decrease ReCoW's overall throughput. We set different metadata thresholds for the frequency to compare the impact of checkpoint frequency on throughput. For the workload, we choose the YCSB A test, and we find that even with the frequent 1MB metadata threshold, the performance degradation remains largely negligible, at around 2%. The performance degradation after adding checkpoints is not significant because

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

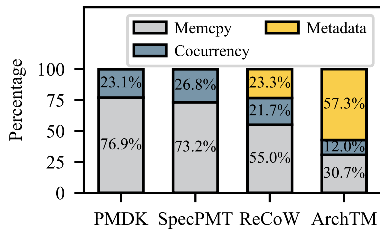


Fig. 16: Latency Breakdown

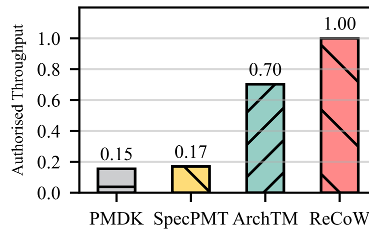


Fig. 17: 512B YCSB A Test

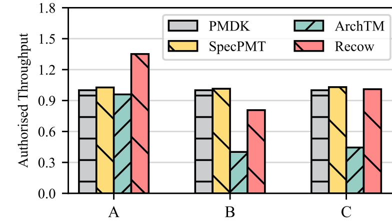


Fig. 18: CXL YCSB Test

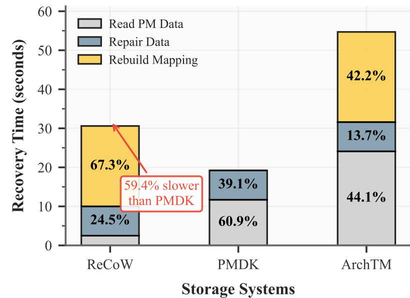


Fig. 19: Recovery Breakdown Test

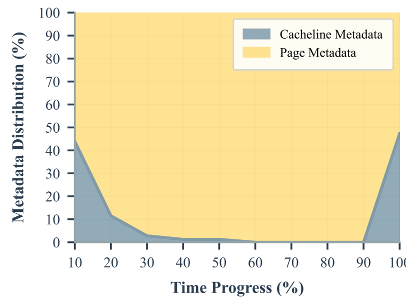


Fig. 20: Metadata Format Test

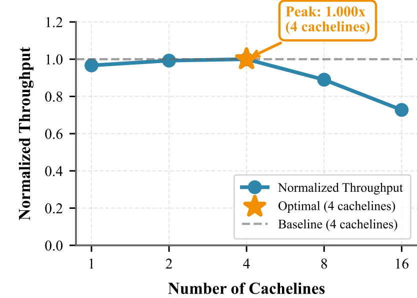


Fig. 21: Hybrid Metadata Setting Test

we design the metadata with reduced write amplification and we ensure that the metadata is written sequentially to PM. As shown in Figure 19, we also conducted the recovery test for ReCoW. We use a 40GB YCSB A workload and record the recovery time. For the frequent checkpoint setting, ReCoW can recover the system in about 30 seconds. Compared to PMDK, ReCoW's crash recovery is 59.4% slower, primarily because it needs to process mapping information from DRAM metadata to invalidate illegal updates after a crash.

VIII. Related Work

Software-based transaction: Software-based transactions are implemented entirely in software. Logging-based transactions, including undo [10], [39], [45] and redo [56] variants, are a popular approach but fundamentally suffer from the double-write problem. Optimizations like operational logging in TimeStone [31] reduce log volume but do not eliminate this issue. In contrast, copy-on-write (CoW) transactions avoid double writes but introduce significant metadata overhead. For instance, ArchTM [52] supports variable-sized objects, which improves flexibility but complicates metadata management. ReCoW addresses this by enforcing fixed (fine or page-aligned) granularity. This design choice allows reusing hardware MMU for efficient virtual-to-physical remapping, dramatically simplifying metadata management and eliminating the metadata bottleneck.

Hardware-involved transaction: Hardware-involved transactions leverage hardware features to improve performance. One category uses Hardware Transactional Memory (HTM) to accelerate logging (e.g., DudeTM [37], SPHT [9]). However, HTM suffers from hardware-specific side-channel vulnerabilities—such as Prime+Abort [15], KASLR breaks [23], and TAA [7], [21]—that have led to its disablement in mainstream production environments (e.g., Linux) [44]. In contrast, ReCoW does not rely on TSX-like hardware features and thus does not inherit these HTM-specific vulnerabilities, while remaining compatible with existing side-channel mitigations. Another category modifies memory controllers [4], [5], [43] to manage CoW, but these are inflexible and typically limited to coarse granularities inefficient for fine-grained workloads.

IX. Conclusion

This paper proposes ReCoW, a kernel-user collaborative CoW transaction that leverages the memory management unit (MMU) to accelerate data referencing. ReCoW implements hybrid-granularity metadata tracking to consolidate small updates and make the metadata compatible with remapping. It also features a cost-effective metadata concurrent control mechanism between the kernel and user spaces, ensuring that foreground transactions are not blocked. Additionally, ReCoW employs a lazy merge scheme to aggregate remap syscalls, reducing the kernel page table updating overhead. Extensive evaluation indicates that ReCoW effectively handles CoW-based transactions for large objects, avoiding the dual-write problem found in log-based transactions. It achieves this with minimal metadata overhead, maximizing PM throughput without significantly increasing the metadata size.

References

[1] Gal Assa, Andreia Correia, Pedro Ramalhete, Valerio Schiavoni, and Pascal Felber. Tl4x: buffered durable transactions on disk as fast as in memory. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 245–259, 2023.

[2] Alexandro Baldassin, Rafael Murari, João PL de Carvalho, Guido Araujo, Daniel Castro, João Barreto, and Paolo Romano. Nv-phtm: An efficient phase-based transactional system for non-volatile memory. In *Euro-Par 2020: Parallel Processing: 26th International Conference on Parallel and Distributed Computing, Warsaw, Poland, August 24–28, 2020, Proceedings 26*, pages 477–492. Springer, 2020.

[3] Philip A Bernstein, Vassos Hadzilacos, Nathan Goodman, et al. *Concurrency control and recovery in database systems*, volume 370. Addison-wesley Reading, 1987.

[4] Abhishek Bhattacharyya, Abhijith Somashekhar, and Joshua San Miguel. Nvmr: non-volatile memory renaming for intermittent computing. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 1–13, 2022.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

[5] Miao Cai, Chance C. Coats, and Jian Huang. Hoop: Efficient hardware-assisted out-of-place update for non-volatile memory. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 584–596, 2020.

[6] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. Efficient distributed memory management with rdma and caching. *Proceedings of the VLDB Endowment*, 11(11):1604–1617, 2018.

[7] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. Fallout: Leaking data on meltdown-immune cpus. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 769–784, 2019.

[8] Zhichao Cao, Siying Dong, Sagar Vemuri, and David HC Du. Characterizing, modeling, and benchmarking {RocksDB}{Key-Value} workloads at facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 209–223, 2020.

[9] Daniel Castro, Alexandro Baldassin, João Barreto, and Paolo Romano. {SPHT}: Scalable persistent hardware transactions. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 155–169, 2021.

[10] Dhruva R Chakrabarti, Hans-J Boehm, and Kumud Bhandari. Atlas: Leveraging locks for non-volatile memory consistency. *ACM SIGPLAN Notices*, 49(10):433–452, 2014.

[11] Youmin Chen, Youyou Lu, Fan Yang, Qing Wang, Yang Wang, and Jiwu Shu. Flatstore: An efficient log-structured key-value storage engine for persistent memory. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 1077–1091, New York, NY, USA, 2020. Association for Computing Machinery.

[12] Felix Cloutier. Rdtscp — x86 instruction reference, 2023.

[13] Andreia Correia, Pascal Felber, and Pedro Ramalhete. Romulus: Efficient algorithms for persistent transactional memory. In Christian Scheideler and Jeremy T. Fineman, editors, *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 271–282. ACM, 2018.

[14] CXL Consortium. Compute express link (cxl), 2025.

[15] Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean Tullsen. Prime+abort: A timer-free high-precision l3 cache attack using intel tsx. In *Proceedings of the 26th USENIX Security Symposium*, pages 51–67, 2017.

[16] Yehonatan Fridman, Suprasad Mutalik Desai, Navneet Singh, Thomas Willhalm, and Gal Oren. Cxl memory as persistent memory for disaggregated hpc: A practical approach. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 983–994, 2023.

[17] Kaan Genç, Michael D Bond, and Guoqing Harry Xu. Crafty: Efficient, htm-compatible persistent transactions. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 59–74, 2020.

[18] Jinyu Gu, Qianqian Yu, Xiayang Wang, Zhaoguo Wang, Binyu Zang, Haibing Guan, and Haibo Chen. Pisces: A scalable and efficient persistent transactional memory. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 913–928, 2019.

[19] Frank T Hady, Annie Foong, Bryan Veal, and Dan Williams. Platform storage performance with 3d xpoint technology. *Proceedings of the IEEE*, 105(9):1822–1833, 2017.

[20] Maurice Herlihy and J Eliot B Moss. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th annual international symposium on Computer architecture*, pages 289–300, 1993.

[21] Intel Corporation. Intel® tsx asynchronous abort advisory (intel-sa-00270). https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00270.html, November 2019.

[22] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amir Saman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. Basic performance measurements of the intel optane DC persistent memory module. *CoRR*, abs/1903.05714, 2019.

[23] Yeongjin Jang, Sangho Lee, and Taesoo Kim. Breaking kernel address space layout randomization with intel tsx. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 387–398, 2016.

[24] Myoungsoo Jung. Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd). In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, pages 45–51, 2022.

[25] Rohan Kadekodi, Saurabh Kadekodi, Soujanya Ponnapalli, Harshad Shirwadkar, Gregory R Ganger, Aasheesh Kolli, and Vijay Chidambaram. Winefs: a hugepage-aware file system for persistent memory that ages gracefully. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 804–818, 2021.

[26] Sanidhya Kashyap, Changwoo Min, Kangnyeon Kim, and Taesoo Kim. A scalable ordering primitive for multicore machines. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018.

[27] Takayuki Kawahara. Scalable spin-transfer torque ram technology for normally-off computing. *IEEE Design & Test of Computers*, 28(01):52–63, 2011.

[28] Patrick Kennedy. Samsung memory-semantic cxl ssd at fms 2022 powered by amd xilinx, 2022.

[29] Linux kernel. Xfs reflink.

[30] Inc. Kioxia America. Xl-flash memory, 2024.

[31] R Madhava Krishnan, Jaeho Kim, Ajit Mathew, Xinwei Fu, Anthony Demeri, Changwoo Min, and Sudarsun Kannan. Durable transactional memory can scale with timestone. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 335–349, 2020.

[32] Hsiang-Tsung Kung and John T Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2):213–226, 1981.

[33] Miryeong Kwon, Sangwon Lee, and Myoungsoo Jung. Cache in hand: Expander-driven cxl prefetcher for next generation cxl-ssd. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*, HotStorage '23, page 24–30, New York, NY, USA, 2023. Association for Computing Machinery.

[34] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, 2009.

[35] Ruibin Li, Xiang Ren, Xu Zhao, Siwei He, Michael Stumm, and Ding Yuan. ctfs: Replacing file indexing with hardware memory translation through contiguous file allocation for persistent memory. *ACM Transactions on Storage*, 18(4):1–24, 2022.

[36] 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 Yunji Chen, Olivier Temam, and John Carter, editors, *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, pages 329–343. ACM, 2017.

[37] Mengxing Liu, Mingxing Zhang, Kang Chen, Xuehai Qian, Yongwei Wu, Weimin Zheng, and Jinglei Ren. Dudetm: Building durable transactions with decoupling for persistent memory. *ACM SIGPLAN Notices*, 52(4):329–343, 2017.

[38] Sihang Liu, Suraaj Kanniwadi, Martin Schwarzl, Andreas Kogler, Daniel Gruss, and Samira Khan. Side-channel attacks on optane persistent memory. In *32nd USENIX Security Symposium 2023*. USENIX Association, 2023.

[39] Leonardo Marmol, Mohammad Chowdhury, and Raju Rangaswami. Libpm: Simplifying application usage of persistent memory. *ACM Transactions on Storage (TOS)*, 14(4):1–18, 2018.

[40] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33. Citeseer, 2007.

[41] Thomas Mikolajick, Christine Dehm, Walter Hartner, Ivan Kasko, MJ Kastner, Nicolas Nagel, Manfred Moert, and Carlos Mazure. Feram technology for high density applications. *Microelectronics Reliability*, 41(7):947–950, 2001.

[42] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. Stamp: Stanford transactional applications for multi-processing. In *2008 IEEE International Symposium on Workload Characterization*, pages 35–46. IEEE, 2008.

[43] Yuanjiang Ni, Jishen Zhao, Heiner Litz, Daniel Bittman, and Ethan L Miller. Ssp: Eliminating redundant writes in failure-atomic nvrams via shadow sub-paging. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 836–848, 2019.

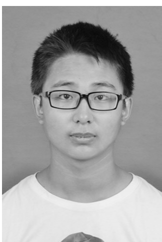
[44] Red Hat, Inc. RHEL 8.3 release notes — TSX is now disabled by default. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/8.3_release_notes/rhel-8-3-0-release, 2020. Accessed: 2025-12-10.

[45] Steve Scargall. *Programming persistent memory: A comprehensive guide for developers*. Springer Nature, 2020.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

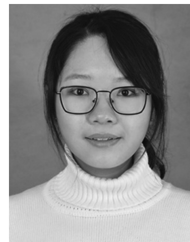
This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3644290

[46] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.
[47] Qing Wang, Youyou Lu, Jing Wang, and Jiwu Shu. Replicating persistent memory Key-Value stores with efficient RDMA abstraction. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 441–459, Boston, MA, July 2023. USENIX Association.
[48] Qing Wang, Youyou Lu, Jing Wang, and Jiwu Shu. Replicating persistent memory {Key-Value} stores with efficient {RDMA} abstraction. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 441–459, 2023.
[49] Rui Wang, Shuibing He, Weixu Zong, Yongkun Li, and Yinlong Xu. Xpgraph: Xpline-friendly persistent memory graph stores for large-scale evolving graphs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1308–1325, 2022.
[50] Zixuan Wang, Xiao Liu, Jian Yang, Theodore Michailidis, Steven Swanson, and Jishen Zhao. Characterizing and modeling non-volatile memory systems. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 496–508, 2020.
[51] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, 2010.
[52] Kai Wu, Jie Ren, Ivy Bo Peng, and Dong Li. Archtm: Architecture-aware, high performance transaction for persistent memory. In *FAST*, volume 21, pages 141–153, 2021.
[53] Lingfeng Xiang, Xingsheng Zhao, Jia Rao, Song Jiang, and Hong Jiang. Characterizing the performance of intel optane persistent memory: A close look at its on-dimm buffering. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 488–505, New York, NY, USA, 2022. Association for Computing Machinery.
[54] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 169–182, 2020.
[55] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies*, FAST'20, pages 169–182, USA, 2020. USENIX Association.
[56] Chencheng Ye, Yuanchao Xu, Xipeng Shen, Yan Sha, Xiaofei Liao, Hai Jin, and Yan Solihin. Specpmt: Speculative logging for resolving crash consistency overhead of persistent memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 762–777, 2023.
[57] Hanbin Yoon, Justin Meza, Naveen Muralimanohar, Norman P Jouppi, and Onur Mutlu. Efficient data mapping and buffering techniques for multilevel cell phase-change memories. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(4):1–25, 2014.
[58] Ming Zhang and Yu Hua. Silo: Speculative hardware logging for atomic durability in persistent memory. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 651–663. IEEE, 2023.
[59] Ming Zhang, Yu Hua, and Zhijun Yang. Motor: Enabling {Multi-Versioning} for distributed transactions on disaggregated memory. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 801–819, 2024.



Aoxin Wei received his B.S. degree from Huazhong University of Science and Technology in 2021. He is currently pursuing his Ph.D. degree at Huazhong University of Science and Technology. His research interests include persistent memory systems, database systems, and storage systems.
Affiliation: Huazhong University of Science and Technology.

Jian Zhou received the Ph.D. degree in computer engineering from the University of Central Florida, Orlando, FL, USA, in 2018. He joined the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China, as an Associate Professor in 2021. He worked as a Postdoctoral Fellow with the University of Central Florida from 2018 to 2020. His research interests include persistent memory and solid-state storage drive.
Affiliation: Huazhong University of Science and Technology.



Shuhan Bai received her B.S. degree from Huazhong University of Science and Technology in 2017. She received her Ph.D. degree from Huazhong University of Science and Technology in 2024. Her research interests include storage systems and key-value stores.
Affiliation: Huazhong University of Science and Technology.

Yufan Jia received his B.S. degree from Huazhong University of Science and Technology in 2022. He is currently pursuing his Master's degree at Huazhong University of Science and Technology. His research interests include persistent memory and file systems.
Affiliation: Huazhong University of Science and Technology.

Jintian Wu received his B.S. degree from Huazhong University of Science and Technology in 2022. He is currently pursuing his Master's degree at Huazhong University of Science and Technology. His research interests include storage systems and database systems.
Affiliation: Huazhong University of Science and Technology.

Hong Jiang (Fellow, IEEE) is a Nedderman Distinguished Professor and Chair of the Computer Science and Engineering Department at the University of Texas at Arlington. He was a Professor at the University of Nebraska-Lincoln. His research interests include computer architecture, storage systems, and high-performance computing. He is an IEEE Fellow and a member of the ACM.
Affiliation: The University of Texas at Arlington.

Fei Wu Fei Wu (Member, IEEE) received the B.E. and M.E. degrees in electrical automation, control theory, and control engineering from Wuhan Industrial University, Wuhan, China, in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, in 2005. She is currently a Professor with the Wuhan National Laboratory for Optoelectronics, HUST. Her research interests include computer architecture, nonvolatile memory, and high-performance and high-reliability storage systems.
Affiliation: Huazhong University of Science and Technology.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on December 17,2025 at 16:08:28 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.