

Charting the Future of Persistent Memory: Insights from Optane PMem

VIKAS GAJANAN KALE, SHIFIN MOHAMMED, SAI NISHCHAL GAMINI, SIDHARTH MENATH, JOBIN ROY, and RUTURAJ DILIP GAWADE, Otto von Guericke University, Magdeburg, Germany

Intel Optane DC Persistent Memory (PMem) introduced a new tier in the memory-storage hierarchy by combining byte-addressability, non-volatility, and large capacity. This survey analyzes 261 research papers published between 2019 and 2025 that employed Optane PMem to redesign system components such as file systems, databases, and data structures with the goal of improving performance and persistence. We group the system redesigns presented in the literature and examine how researchers have evaluated them, focusing on benchmark tools such as YCSB (Yahoo! Cloud Serving Benchmark), Filebench, and TPC (Transaction Processing Performance Council). Across these studies, performance gains are often reported in the range of $2\times$ to $10\times$. However, we also notice several recurring issues. Reproducibility is a common concern, endurance testing is rarely thorough, and many papers provide only limited evaluation across realistic deployment settings. In addition to summarizing past work, we highlight directions that are beginning to gain traction, including NUMA-aware (Non Uniform Memory Access) architectures, engines built specifically for PMem, and remote or disaggregated deployments. Finally, we reflect on these developments in light of Intel's discontinuation of Optane and consider how future systems might adapt to CXL-based persistent memory (Compute Express Link). The goal of this survey is not only to summarize what has been done, but also to provide guidance for designing and evaluating the next generation of systems using byte-addressable, non-volatile memory.

Additional Key Words and Phrases: Intel Optane DC Persistent Memory, Non-volatile Memory (NVM), Byte-addressable, File Systems, Databases, HPC (High-Performance Computing) Applications, Software Architecture, AppDirect Mode, Persistent Data Structures, Performance Benchmarking, Endurance Management.

ACM Reference Format:

Vikas Gajanan Kale, Shifin Mohammed, Sai Nishchal Gamini, Sidharth Menath, Jobin Roy, and Ruturaj Dilip Gawade. 2025. Charting the Future of Persistent Memory: Insights from Optane PMem. 1, 1 (July 2025), 53 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

The emergence of Intel Optane DC Persistent Memory (PMem) introduced a new tier in the memory-storage hierarchy, offering large, affordable, byte-addressable memory that retains data across power loss [1]. Optane PMem modules (128–512 GB each) reside on the memory bus, operating either in Memory Mode as volatile expansion of DRAM or App Direct Mode as persistent, direct-load/store memory [1]. In theory, this technology combines the low-latency random access of memory with the persistence of storage. Early evaluations showed that Optane PMem's performance sits between DRAM (dynamic random access memory) and SSD (Solid State Drive), with latencies in the hundreds of nanoseconds and bandwidth lower than DRAM but far higher than NAND SSD [1, 2]. These characteristics have motivated a wave of systems research to exploit PMem to improve

Authors' address: Vikas Gajanan Kale; Shifin Mohammed; Sai Nishchal Gamini; Sidharth Menath; Jobin Roy; Ruturaj Dilip Gawade, Otto von Guericke University, Magdeburg, Germany.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM XXXX-XXXX/2025/7-ART

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

storage and memory management.

The influence of Optane PMem has been pervasive across the system layers. Researchers have redesigned file systems, database engines, and data structures to take advantage of fast persistent loads and stores, aiming to eliminate software overheads such as kernel indirection and log delays. At the same time, they face new issues such as ensuring crash consistency with flushes, handling asymmetric write performance, and limited write endurance. In order to assess how Optane PMem has shaped system design and what lessons have been learned, we perform a systematic literature review. We collected 261 papers (2019–2025) that specifically evaluate or utilize Intel Optane DC PMem in computer systems. We focus on three research questions:

- How has Optane PMem influenced the architectural design and performance of databases, file systems, and data structures?
- What methodologies and benchmarks are used to evaluate Optane PMem systems, and what are their limitations?
- What are the current and potential application scenarios for byte-addressable Optane PMem?

In this survey, we synthesize the findings across these works. We find that Optane PMem deployments frequently achieve substantial performance gains, which were as high as 3 - 5x in file storage systems compared to traditional storage systems[3], or persistent indexes exceeding DRAM-backed ones by 4 - 8x [4] due to reduction of software overheads and direct persistence. At the same time, we identify consistent challenges such as PMem's higher latency and lower bandwidth relative to DRAM [5], the need for explicit cache line flushing for durability, and software modifications required to exploit persistence. We also observe that research efforts skew toward lower-level data structures (approximately 58% of studies) and file systems, indicating a need for more intensive system integration. By presenting these insights, our work informs the future of persistent memory, including how next-generation technologies like CXL-attached memory and new NVM devices can build on Optane's legacy. The remainder of this paper is structured as follows - we provide background on Optane PMem, review related work, detail our methodology, then present our findings on architectural insights, evaluation methods, application scenarios, challenges and gaps, and implications for the future, before concluding.

2 BACKGROUND AND TECHNICAL OVERVIEW

Optane PMem is based on 3D XPoint phase-change memory technology, packaged as DIMMs (Dual In-line Memory Module) that plug into standard memory slots. Architecturally, it introduces a persistent memory tier between DRAM and SSDs, providing larger memory capacity per socket at lower cost/GB than DRAM [6]. Each module offers hundreds of gigabytes of capacity [1] and can be used in two modes. In Memory Mode, the PMem acts as volatile main memory expanded by a DRAM cache; this requires no application changes but provides no persistence i.e., data vanishes on power loss. In App Direct Mode, the PMem is used as true persistent memory, where applications and file systems can perform direct load/store access to persistent addresses via CPU instructions along with OS support for memory mapping. App Direct mode enables data to survive crashes and reboots, but entails new instructions such as cache line flush and memory fence, to explicitly persist updates from CPU caches to the PMem hardware.

In terms of performance, Optane PMem has significantly lower latency than SSDs but higher latency than DRAM. Reported read latencies are on the order of 300–350 ns, with 64-byte writes around 0.5–0.7 μ s [4]. Bandwidth per module is also lower than DDR4 DRAM, approximately 2–6 GB/s vs approximately 10–20 GB/s for DRAM, and writes are slower than reads, exhibiting bandwidth asymmetry. These traits mean that naive usage of PMem may suffer from write-induced stalls and not fully match DRAM performance. A study also noted that

RocksDB running on Optane PMem initially achieved only $0.5\times$ the write throughput of an SSD-based setup due to software bottlenecks [7]. Consequently, system designers must consider PMem's non-uniform performance gains, optimizing for small, random accesses where it excels, and reducing its slower bulk writes and access parallelism limits.

In App Direct mode, Optane PMem implements an epoch-based persistence domain. Stores to PMem persist at the latest when the data is flushed from the CPU's memory caches to the DIMM media. For reliability, developers use instructions like CLWB (Cache Line Write Back) or CLFLUSHOPT (Flush Cache Line Optimized), and memory fences like SFENCE (Store Fence) to ensure ordering, thereby achieving crash consistency. Absent additional hardware support, flush and fence overheads become part of the software cost of using PMem. The second-generation Optane (200 series, Circa 2021) introduced an enhancement called eADR (enhanced Asynchronous DRAM Refresh), which extends the persistence domain into the CPU caches (on power loss, dirty cache lines are flushed to PMem via capacitor power). With eADR, software can in principle skip explicit flushes for durability. Some research prototypes have started to leverage eADR to simplify persistence, for example, using hardware transactional memory on PMem without flushes [4], but the majority of systems were built assuming first-generation Optane, thus carefully managing cache flush operations for atomic durability.

Optane PMem's large capacity is a major appeal. Servers configured with multiple PMem DIMMs can reach multi-terabyte memory sizes. For instance, a system with 12×128 GB PMem DIMMs provides approximately 1.5 TB of persistent memory on a single machine [8], and research testbeds have even used clusters of machines each with greater than 6 TB of PMem installed [9]. These high capacities enable new use cases in-memory databases with datasets larger than DRAM, or single-node analytics on huge graphs. However, using PMem at scale also stresses aspects like NUMA effects and necessitates careful allocation to avoid remote PMem accesses, which are slower. In summary, Optane PMem offers a unique combination of features, large byte-addressable persistent memory with performance close to memory, but requires reconsideration of system design to fully harness its potential while coping with its peculiarities.

3 RELATED WORK

The release of Optane DC Persistent Memory led to a wave of studies that aimed to understand its behavior and potential. Before and alongside the works we surveyed, several efforts focused on characterizing the hardware itself. For instance, one study looked at the basic performance of Optane PMem and confirmed the clear gap in latency and bandwidth compared to DRAM and SSD. It also pointed out practical tips for how to align accesses and make use of concurrency effectively [5]. Other papers explored how Memory Mode works by treating DRAM as a cache for PMem. These studies showed how access patterns could affect hit and miss rates, which in turn influence overall performance in systems that mix DRAM and PMem [10].

At the same time, some researchers worked on benchmarking tools and models tailored for persistent memory. A good example is PerMA-Bench, which was created to test PMem access patterns and persistence primitives in a consistent way [11]. It includes microbenchmarks that measure things like latency, throughput, and CPU overhead, helping researchers compare systems more fairly. Others evaluated software libraries like PMDK to pinpoint performance bottlenecks and suggest improvements [8]. These contributions don't just provide raw data—they help shape better evaluation practices that match PMem's unique characteristics, and they go hand-in-hand with the architectural studies we reviewed.

Moreover, beyond individual system papers, there have been broader perspectives and conceptual works. For

instance, some researchers discuss persistent memory in the context of universal memory abstractions, where a single storage/memory pool could simplify programming models [8]. Others considered the theoretical implications of durable memory for concurrent algorithms (so-called “universal constructions” in data structures [8]). While these do not present new system prototypes, they provide context for the significance of persistent memory in computing’s evolution.

Our work draws on ideas from several earlier surveys that looked at persistent memory. One of them [12] offers a clear and well-structured overview of how PMem has been integrated into operating systems and file systems, especially at the kernel level. Although their focus is on low-level software, their approach to categorizing system changes helped shape how we built our own architectural taxonomy. Another survey [13] takes a wider perspective, reviewing a decade of NVM research. What stood out to us there was their emphasis on system-wide adaptations, which aligns with our own focus on changes in file systems, databases, and data structures. A third study [14] looks specifically at NVM file systems, but it lacks a structured or repeatable methodology. Compared to these, our survey narrows the scope to research centered on Optane PMem and aims to provide a more systematic classification of redesign patterns, backed by cross-domain comparisons.

Our work differs from these related efforts by offering a holistic review of a large body of Optane PMem-based system research. Where prior hardware studies isolated performance traits, and benchmarking papers provided tools, we synthesize how those traits have been exploited or mitigated in actual system designs (databases, file systems, etc.). Likewise, while conceptual discussions outline potential futures, our survey grounds the discussion in empirical results and concrete use cases from real implementations. In the next sections, we detail our methodology for collecting and analyzing the literature, then delve into the key insights regarding architectures, evaluation practices, application domains, and open challenges for persistent memory.

4 METHODOLOGY

This study adopts a systematic literature review (SLR) approach to examine the research landscape surrounding Intel Optane DC Persistent Memory (DCPMem). Our goal was to collect, organize, and analyze relevant works in a structured and reproducible manner. To ensure thorough coverage, we defined clear research questions, applied explicit inclusion and exclusion criteria, and synthesized findings across multiple system layers and application domains.

4.1 Research Questions

We defined three primary research questions (RQs) to guide the review, as mentioned in the Introduction:

RQ1 (Architecture): How has Intel Optane DCPMem influenced the architectural design and performance of databases, file systems, and data structures? This includes sub-questions about how file systems have been redesigned for PMem (and what gains achieved), how databases and in-memory systems have been re-architected for PMem, how Optane’s microarchitectural characteristics such as write latency, bandwidth asymmetry influence data structure design, and which PMem operational modes (App Direct, Memory, Mixed) are preferred in these systems and why.

RQ2 (Evaluation): What are the common methodologies and benchmark suites used to evaluate Optane PMem systems, and how do they ensure fair and reproducible results? What limitations exist in these evaluation approaches?

RQ3 (Applications): What are the existing and potential application scenarios for byte-addressable persistent memory (especially Intel Optane)? Essentially, in what domains or use-cases has Optane PMem been deployed or studied, and what benefits or challenges were observed in those contexts?

By structuring the review around these questions, we aim to capture the impact of PMem on system design, the practices of measuring that impact, and the real-world contexts in which PMem is making a difference.

4.2 Search Strategy and Data Sources

We used Google Scholar to search for papers, always including “Optane” and adding a second keyword depending on the topic we were focusing on. The exact combinations are listed in Table 1. For each search, we went through the top 300 results. That number wasn’t perfect, but it was a good trade-off between being thorough and not spending too much time digging through less relevant material. Since Google Scholar ranks results by relevance and citations, we assumed the important papers would mostly appear early anyway.

Table 1. Search-term combinations used for literature retrieval

Primary Keyword	Combine	Secondary Keyword
“Optane”	×	File System
		Database
		Data Structure
		Index Structure
		Storage Engine
		High Performance Computing (HPC)

Because of the large number of papers to go through, we created a Python script to automatically extract key information like titles, authors, and URLs. This helped us gather and organize the data more efficiently while also making sure the process could be repeated reliably.

Our search focused on top-tier systems, architecture, and database venues, including OSDI, SOSP, USENIX FAST, ATC, SC, as well as niche outlets such as ACM Transactions on Storage and HotStorage. We also included a small number of high-impact preprints from platforms like arXiv, particularly when they were frequently cited and focused on application-level redesigns. This process yielded 1,196 non-duplicate papers, which were later screened for relevance and analyzed in detail.

4.3 Inclusion and Exclusion Criteria

From the gathered set, we applied inclusion/exclusion criteria to select studies most relevant to our research questions:

- We included papers that explicitly used and evaluated Intel Optane DC Persistent Memory. This meant the work either employed real Optane hardware in experiments or, in some cases, an emulator or simulator configured to model Optane-like NVM. We focused on system-level optimizations (software or hardware designs) that leverage PMem’s capabilities, namely persistence and byte-addressability. Our goal was to include exclusively experimental papers (e.g., building a new file system) and to exclude papers that focused solely on evaluations.

- We excluded papers that discussed non-volatile memory in general but did not specifically target Optane or did not provide new insights for Optane PMem usage. For example, older works using only battery-backed NVDIMMs or simulations of phase-change memory without reference to Optane were not included, to keep our review focused on the Optane era. We also excluded patents, editorial articles, and very short workshop position papers, unless they contained unique data or ideas relevant to our RQs.
- We required that the works be in English and accessible through our institution or open access. We gave preference to peer-reviewed venues, but as noted, we included a few preprints that were influential and later published. All the works other than research papers like books, thesis, posters, presentations were excluded.

After applying the criterias, we arrived at a final set of 261 papers that form the basis of this survey. These span the years 2019 to early 2025 and representing domains such as filesystems, data structures and databases.

4.4 Data Extraction Process

We systematically extracted data from each included paper using a predefined form to ensure consistency. For each paper, we extracted data for multiple sections as shown in Table 2.

Table 2. Data Extraction Schema Used for PMem Paper Analysis

Section	Description
Bibliographic Info	Title, publication venue/year, and DOI/URL.
Domain and Context	Classification of the paper’s domain (e.g., File-System, Database, Data-Structure, HPC, etc.), and specific context when applicable (e.g., “LSM-tree KV store”, “Graph processing”, “Distributed file system”).
Paper’s Focus	The primary focus (main contribution) and any secondary focus. For example, a paper might focus on a “Persistent B-tree index” with a secondary emphasis on “concurrency mechanisms”.
Optane Usage Mode	Whether real Optane hardware was used, and in what mode (App Direct, Memory, or Mixed). Also includes whether simulation or emulation was used, and any rationale given for mode selection.
System/Tool Chain Details	Key aspects of the implementation stack, such as OS or kernel modifications, libraries (e.g., Intel PMDK), custom allocators, or any specialized hardware/software infrastructure.
Design Aspects	Architectural or algorithmic innovations introduced to optimize for PMem. Captures redesign types (e.g., new file system, persistent index), consistency model (e.g., journaling, copy-on-write), and endurance techniques (e.g., wear leveling, write reduction).
Evaluation	Workloads or benchmarks used (e.g., YCSB, TPC-C, Filebench), reported metrics (e.g., throughput, latency, CPU usage), baselines (e.g., DRAM, SSD, other PMem systems), and type of evaluation (e.g., experimental, formal verification, fault injection).
Key Results	Main quantitative findings, such as “Throughput improved 3× over ext4-DAX” or “Detected 15 new crash-consistency bugs”.
Challenges & Limitations	Challenges faced during deployment, integration, scalability, or other limitations explicitly acknowledged by the authors.
Future Work & Gaps	Explicit future work suggestions, unresolved issues, or open research questions mentioned by the authors.
Effect of Using Optane	Describes how Optane influenced the design, evaluation results, or limitations in the paper’s context. Highlights performance improvements, new bottlenecks, or architectural dependencies on PMem characteristics.

This extraction was performed by reading the papers in detail. We cross-validated critical data points by consulting tables and figures in the papers to ensure accuracy for instance, matching a stated improvement to the

correct baseline value. The data was tabulated as a spreadsheet to allow sorting and grouping in later analysis. We also maintained traceability: for each data item, we kept a note of where in the paper (section or page) it was derived, in case clarification was needed later.

4.5 Review Quality Control

To maintain quality and consistency in this review, we employed several control measures:

- *Pilot Extraction*: Initially, we had multiple reviewers independently extract data from a small sample of papers covering each major domain: one file system paper, one database paper, and one data structure paper. We then compared the extracted information and harmonized our understanding of each field. This helped refine the extraction form and definitions (e.g., what counts as a “Redesign Type” or what level of detail to record for “Metrics Used”). We then distributed the whole corpus of papers among the team members and performed the data extraction.
- *Consensus & Cross-Checking*: For each paper, at least two team members discussed the key findings and any ambiguous points. In cases where a paper’s description was unclear (for example, if the operation mode was not explicitly stated), we checked multiple parts of the paper and even looked at any supplemental material or source code if available to infer the most likely interpretation. Disagreements were resolved through discussion.
- *Data Consistency*: After populating the database of extracted data, we performed automated checks for consistency. For instance, we flagged if any entry had an impossible combination (such as claiming “Real Optane” usage but an Operation Mode of “Not applicable”), or if a key field was accidentally left blank. We also standardized terminology post-hoc (e.g., some entries said “NVM” vs. “PMem”; we chose a uniform term in our analysis).
- *Representativeness*: We verified that our final selection of papers indeed covered the breadth of our RQs. Given the fast-evolving nature of PMem research, we wanted to ensure no significant subtopic was missed. We cross-checked against the programs of major conferences each year to see if any prominent Optane-related work was absent from our list, and if so, we investigated why.

Through these measures, we aimed to minimize bias and errors, ensuring that the insights we draw are well-supported by the source material.

4.6 Synthesis Process

With the data extraction complete, we synthesized findings for each RQ by aggregating and distilling information across papers:

- For RQ1, we started by sorting the papers into three main areas: file systems, databases, and data structures. Within each group, we looked for recurring ideas and standout design choices. We kept track of reported performance improvements and how they were achieved. Along the way, we noticed trends—like which operation modes were used more often (for instance, App Direct vs Memory Mode) and the reasons authors gave for those choices. Our findings combined numbers, such as the percentage of file systems using log-structured designs, with descriptive insights into how different systems addressed issues like crash consistency.

- For RQ2 (Evaluation Methodology), we compiled statistics on the usage of benchmarks and metrics. We identified the most popular benchmarks (for example, YCSB for key-value stores appeared in over 100 papers) and the typical evaluation setup (most works compare to DRAM and SSD baselines, measure throughput and latency). We also collected anecdotal notes on any shortcomings or limitations mentioned like reproducibility issues, or if certain benchmarks were insufficient. This helped us summarize the state of evaluation practices and what might be improved.
- For RQ3 (Application Scenarios), we categorized the papers by application domain (enterprise storage, HPC, AI/ML, etc.). We created a mapping of use cases to representative papers/solutions. By doing so, we could see which areas have seen significant attention (e.g., key-value stores, OLTP databases, file systems for OS kernels, in-memory analytics, etc.) and what unique benefits PMem brought in those scenarios. We also noted any emerging use cases that are still exploratory.
- Cross-cutting themes: In addition to the RQ-specific syntheses, we looked for cross-cutting patterns. For example, independent of domain, many papers mentioned the challenge of NUMA latency or the need for asynchronous durability techniques. We extracted such themes to discuss in the Discussions section. We also correlated features- e.g. whether using real hardware vs simulation affected reported performance, or whether second-generation Optane (with eADR) was associated with particular improvements.

Throughout the synthesis, we aimed to directly tie statements to evidence from the papers. The following sections present the results of this synthesis, with in-text citations [numbers] referring to the specific papers supporting each claim or example.

5 ARCHITECTURAL INSIGHTS

Research Question 1 explores the insights gained from system-level redesigns aimed at making the most of Optane DC Persistent Memory’s unique features. This includes changes made in databases, data structures, and file systems. The introduction of Intel Optane PMem has had a major influence on system architecture, especially in how file systems are designed, how databases are managed, and how persistent data structures are implemented. In this section, we examine each of these areas in detail, focusing on the key architectural shifts and the reported performance outcomes.

5.1 File Systems Redesigns

Our systematic analysis of PMem file system research shows a field that is still facing major evaluation challenges, even as it moves toward more specialized design solutions. The data highlights several important insights that question commonly held beliefs about how progressive and well-directed this area of research really is.

One of the clearest takeaways comes from comparing how much research exists in each area versus how well that research is evaluated. For instance, concurrency-aware redesigns are the most heavily studied, with 39 papers. Yet, they receive the same low benchmark rating, just two stars—as areas that have only one or two papers. This raises a red flag: more papers don’t necessarily mean better quality or more careful evaluation.

This inverse relationship indicates that popular research areas may be suffering from the researcher’s inclinations to pursue incremental variations on established themes rather than addressing fundamental evaluation weaknesses. The field appears to be optimizing for publication volume rather than methodological advancement.

Table 3. File Systems-Redesign Type Summary

Redesign Type	Member Research Paper(s)	Count
Concurrency Aware	[3, 8, 10, 15–50]	39
Metadata-Optimized	[10, 19–22, 27, 29–32, 34–40, 42, 44, 45, 48–64]	37
Crash-Consistency	[5, 10, 19, 20, 24, 25, 28, 30, 35–37, 39, 40, 42, 43, 45, 47, 48, 50, 51, 56–58, 60, 61, 63–72]	35
Log-Structured	[10, 20, 24, 25, 30, 31, 35–38, 40, 42, 47, 58, 60, 63, 65, 67, 73–78]	24
Latency-Optimized	[10, 18–22, 26–29, 31, 35, 40, 44, 51, 52, 79–83]	21
Endurance-Aware	[5, 10, 19, 22, 27, 29, 31, 33, 37, 48, 54–57, 68–71]	18
Tiered-Storage	[5, 18, 28, 30–32, 35, 45, 47, 56, 57, 61, 70, 71, 75, 84, 85]	17
Allocator-Aware	[15, 19, 21, 22, 24, 30, 33, 35, 38, 42, 78, 86–89]	15
Object Based Redesign	[3, 10, 16, 17, 19, 24, 29, 32, 34, 36, 40, 44, 48, 61, 63]	15
Cache-Optimized	[34–36, 51, 52, 54, 55, 71, 87, 90–92]	12
Copy-on-Write	[19, 37, 38, 57, 58, 60, 61, 63, 68, 69, 72, 73]	12
Transactional	[5, 32, 45, 49, 57, 61, 70]	7
Bandwidth-Asymmetric	[18, 19, 21, 25, 28, 32, 34, 35, 40, 54, 55]	6
Fault-Tolerant	[25, 32, 34, 54, 55]	5
Deduplication-Aware	[64, 65]	2
Security-Aware	[39]	1

Table 4. **File Systems - Redesign Types with benchmark Ratings and Coverage Gaps.** *Legend:* Benchmark rating (0–5): Stars (★) indicate overall benchmark quality and rigor. Higher rating reflects more realistic workloads, multi-node or real-world scale evaluations, endurance testing, and availability of public artifacts.

Redesign Type	Benchmark Rating	Research Gaps
Concurrency Aware	★★☆☆☆	Lack of multi-node and cross-cluster validation. Limited testing under realistic deployment scales and contention-heavy workloads.
Metadata-Optimized	★★☆☆☆	Most studies rely on synthetic metadata workloads, lacking evaluation under real-world, mixed I/O access patterns.
Crash-Consistency	★★☆☆☆	Limited use of realistic, multi-tenant workloads; most work fails to evaluate recovery behavior at scale.
Log-Structured	★★☆☆☆	Evaluations often exclude real-world, mixed workloads and do not assess long-term fragmentation or aging effects.
Latency-Optimized	★★☆☆☆	Benchmarks lack workload realism and diversity; limited exploration of tail-latency behaviors in complex deployment environments.

Continued on next page

Table 4 – continued from previous page

Redesign Type	Benchmark Rating	Research Gaps
Endurance-Aware	★★☆☆☆	Sparse validation using real-world, mixed workloads; durability optimizations rarely tested under stress or device variability.
Tiered-Storage	★★☆☆☆	Evaluations rely heavily on synthetic or microbenchmarks; underexplored impact of tier placement policies in production-like settings.
Allocator-Aware	★★☆☆☆	Allocator behavior and NUMA effects are poorly studied; cross-node coordination overhead is rarely measured.
Object Based Redesign	★★☆☆☆	Focused mainly on KVS layering; limited insights on how redesigns affect consistency, garbage collection, and storage compaction.
Cache-Optimized	★★★☆☆	Synthetic workloads dominate evaluations; cache interactions with concurrency and persistence are often untested.
Copy-on-Write	★★☆☆☆	Most papers adopt CoW as a mechanism, not a central focus, they underexplored space reclamation, metadata tracking, and tail latency effects of CoW Very few discuss integration with mmap(), multi threaded commit ordering, or wear balancing
Transactional	★★☆☆☆	No public workloads or endurance tests in any paper All lack multi node transactional stress testing. Few explore failure recovery time under real contention.
Bandwidth-Asymmetric	★★☆☆☆	Most papers use synthetic or semi-realistic workloads; few measure NVM-SSD bandwidth mismatch at system scale. Lack of power failure and write pacing experiments aware bandwidth balancing underexplored
Fault-Tolerant	★★★☆☆	Most systems focus on deduplication or metadata protection but lack multi node crash recovery validation. Few explore replica consistency under power failure.
Deduplication-Aware	★★☆☆☆	No paper evaluates on real-world deduplication workloads (backup traces, VM image datasets). Endurance impact of deduplication logic not analyzed node tests or public artifacts available
Security-Aware	★★★☆☆	Lack of realistic adversarial models; evaluations often ignore performance-security trade-offs under real-world conditions.

The low benchmark scores across all categories point to a shared problem in how PMem file system research is evaluated. But what stands out even more is that nearly every study leans heavily on synthetic workloads. This isn't just a matter of convenience, it shows how hard it is for the field to move from controlled setups to

meaningful, real-world testing. That all 16 redesign categories rely on the same limited methods suggests this is not about individual choices, but about deeper structural issues in the research process.

One issue that keeps coming up is the lack of multi-node validation. Most of the studies mentions it, but very few actually tackle it. This is worrying, because real-world storage systems are distributed by default. Sticking to single-node experiments makes it difficult to produce results that hold up in actual deployment.

Even in areas that seem more progressive, like endurance-aware designs, the same problem appears. There are 18 papers on the topic, but none really engage with how these systems wear out over time in real conditions. Researchers are trying to design for hardware limits that they haven't properly measured. This gap needs to be addressed if the field wants to stay relevant.

Table 5. File System Redesigns with the core architectural definitions

Redesign Pattern	Core Architectural Idea
Bandwidth-Asymmetric	Optimizes data paths when read and write throughput differ significantly on Optane PMEM.
Latency-Optimized	Reduces end-to-end access latency via critical-path trimming, prefetching, and lightweight code paths.
Cache-Optimized	Aligns structures for CPU-cache friendliness, minimizing misses when accessing PMEM-resident data.
Allocator-Aware	Places threads and allocations NUMA-locally and uses PMEM-oriented allocators to curb fragmentation.
Endurance-Aware	Mitigates write-amplification and wear hot-spots to extend Optane's write-cycle lifetime.
Crash-Consistency	Ensures on-PMEM images remain consistent after crashes through journaling, CoW roots, or flush ordering.
Transactional	Offers ACID-style atomic multi-write durability directly in the storage layer on PMEM.
Copy-on-Write	Uses CoW snapshotting so updates clone only modified pages, enabling rapid rollback and consistency.
Log-Structured	Writes are appended sequentially in a log, simplifying recovery and minimizing random updates on PMEM.
Tiered-Storage	Hot data stays on PMEM while colder data migrates to SSD/HDD/cloud, balancing cost/performance.
Concurrency Aware	Shards locks or uses lock-free and RDMA methods so PMEM's low latency scales with cores/nodes.
Fault-Tolerant	Replicates or erasure-codes data across nodes/devices to survive power loss or PMEM failures.
Security-Aware	Encrypts and integrity-checks PMEM contents to prevent unauthorized post-reboot data access.

Continued on next page

Table 5 – continued from previous page

Redesign Pattern	Core Architectural Idea
Deduplication-Aware	Inline or offline dedupe removes redundant data blocks/objects, saving capacity and writes.
Metadata-Optimized	Re-designs directory/inode structures for microsecond-level metadata operations in PMEM.
Object-Based Redesign	Provides immutable object or key-value APIs instead of byte-stream files, fitting random-access workloads.

The architectural patterns reveal three critical insights not immediately obvious from the raw categorization: Multiple seemingly distinct approaches, namely Latency-Optimized, Cache-Optimized, Bandwidth-Asymmetric are actually tackling the same underlying problem: software overhead elimination. This convergence suggests the field has identified the core challenge but is fragmenting solutions across artificial categories. The existence of distinct Crash-Consistency, Transactional, and Copy-on-Write categories indicates the field hasn't resolved the fundamental trade-off between performance and durability guarantees. Unlike traditional storage where this trade-off was well-understood, PMem has reopened this question without clear resolution.

Patterns like "Hardware Transactional Memory integration" and "eADR exploitation" signal a shift toward hardware-dependent solutions. This creates a concerning trend toward vendor lock-in and reduced portability.

Table 6. Redesign type abbreviations used throughout the File System Section.

Abbreviation	Redesign Type
SC	Scalable/Concurrency Aware
MO	Metadata-Optimized
CC	Crash-Consistency
LS	Log-Structured
LO	Latency-Optimized
EA	Endurance-Aware
TS	Tiered-Storage
AA	Allocator-Aware
OR	Object based redesign
CO	Cache-Optimized
CW	Copy-on-Write
TR	Transactional
BA	Bandwidth-Asymmetric
FT	Fault-Tolerant
DA	Deduplication-Aware
SA	Security-Aware

The correlation matrix illustrated in Figure 1 exposes several non-obvious relationships that challenge traditional categorization: Strong correlations between seemingly unrelated categories (e.g., Security-Aware and Endurance-Aware) suggest that practical deployment concerns are more interconnected than academic categorization implies. Security mechanisms may inadvertently impact wear patterns, or wear-leveling may create

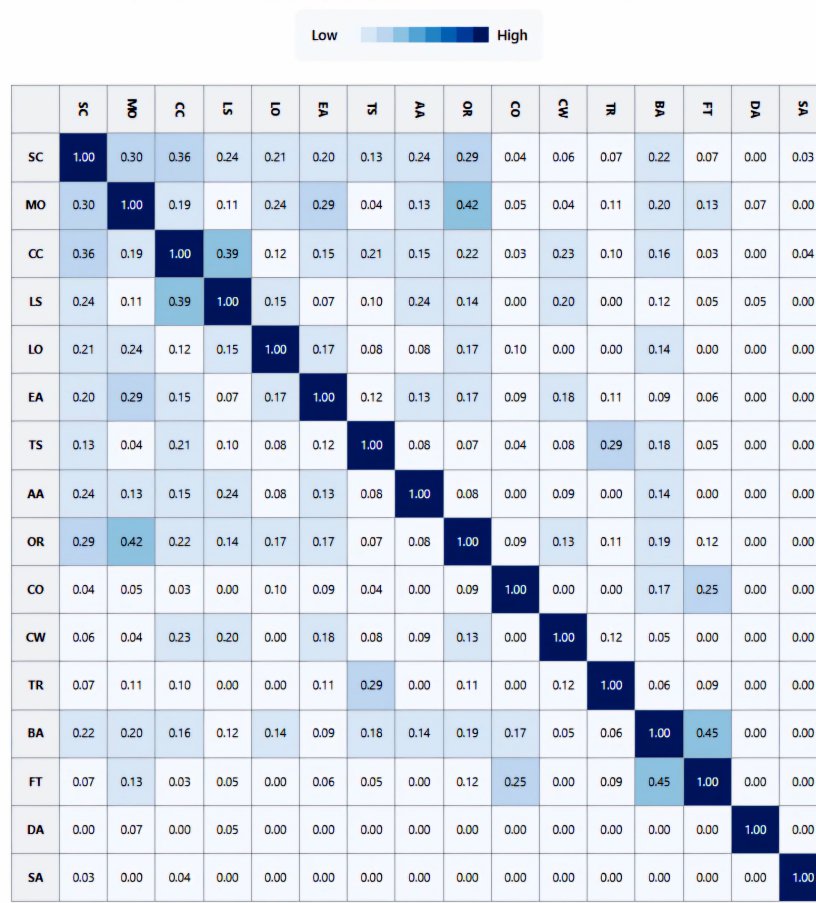


Fig. 1. Co-similarity matrix showing Jaccard Similarity for 16 File system redesign types

security vulnerabilities. The lack of strong correlation between Crash-Consistency and Transactional approaches indicates these are being pursued as competing rather than complementary strategies. This fragmentation may be preventing optimal solutions that combine both approaches.

The matrix reveals three distinct research clusters:

- (1) Performance-Focused Cluster: Latency-Optimized, Cache-Optimized, Bandwidth-Asymmetric
- (2) Reliability-Focused Cluster: Crash-Consistency, Fault-Tolerant, Endurance-Aware
- (3) Architecture-Focused Cluster: Allocator-Aware, Tiered-Storage, Object-Based

This clustering suggests the field is naturally organizing around fundamental system properties rather than the artificial categories often used in academic literature.

App Direct mode appears in more than 85 percent of the studies we reviewed. This is not a coincidence. It reflects a clear judgment by researchers: treating PMem like conventional memory through Memory Mode is not worth the trade-offs. Transparent use might seem attractive in theory, but in practice, the loss of control and performance

Table 7. File-system redesign types and their common benchmark suites

Redesign Type	Benchmark Suites
Concurrency-Aware	FIO, Filebench, YCSB, TPC-C, IO500, IOR, mdtest
Metadata-Optimized	Filebench, FIO, YCSB, FxMark, TPC-C
Crash-Consistency	FIO, YCSB, Filebench, TPC-C
Log-Structured	Filebench, FIO, YCSB, TPC-C
Latency-Optimized	YCSB, TPC-C, Filebench, FIO
Endurance-Aware	YCSB, TPC-C, IOR, mdtest
Tiered-Storage	FIO, Filebench, YCSB, TPC-C, IOR
Allocator-Aware	FIO, Filebench, IO500, IOR, mdtest
Object-Based Redesign	IO500, IOR, mdtest, YCSB, Filebench
Cache-Optimized	FIO, Filebench, YCSB, TPC-C, IOR, mdtest
Copy-on-Write	Filebench, TPC-C, FIO, YCSB
Transactional	YCSB, FIO, Filebench
Bandwidth-Asymmetric	TPC-C, FIO, Filebench
Fault-Tolerant	FIO, YCSB, Filebench
Deduplication-Aware	FIO
Security-Aware	FIO, Filebench, FxMark

is too high. What we see instead is a move toward models that accept PMem for what it is—something that sits between memory and storage and requires explicit handling. This shift has real consequences. It complicates adoption in legacy systems but opens the door to designs that can take full advantage of PMem’s capabilities. The consensus around App Direct is not just about benchmarks. It shows that the field is redefining the role of memory in system architecture.

This analysis makes it clear that PMem file system research is at a turning point. The community has done well to surface the main technical hurdles, but the bigger issue now is how these systems are being evaluated. Unless the quality of evaluation improves, even the most promising architectural ideas may struggle to make a real-world impact. What stands out from the data is that instead of branching further into niche directions, the field might benefit more from coming together around the three clusters that emerged in the correlation study. At the same time, there needs to be a deliberate push to fix the evaluation problems that are holding back progress across the board.

5.2 Databases and Storage Engines Redesigns

Relational and key-value databases form the transactional building blocks of nearly every online service. Originally designed to work in a two-tiered world of fast, volatile DRAM and slow, block-oriented disks, database engines gained significant advantages after the emergence of Optane DC PMem, which provided persistence and byte-addressable memory while operating at just $\sim 3\times$ slower than DRAM [73]. However, it demands 256-byte aligned, flush-ordered writes while keeping endurance in check [92, 93]. In this subsection, we catalogue 60 papers from the entire corpus that have either retrofitted or newly designed database systems to better utilize the unique characteristics of Intel Optane DC persistent-memory modules. We gather the principal design axes, performance gains along with their evidence strengths, and existing gaps, which are further elucidated in Tables 8, 9, 11, and 12.

Table 8. Database-domain papers that redesign architectures for Intel Optane PMEM.

Redesign Type	Member Papers	Count
Multi-Tier Memory Hierarchy Mgmt	[54, 67, 68, 70, 77, 88, 91–99]	15
Persistent Index	[64, 67, 68, 92, 94, 96, 97, 99–103]	12
Dynamic Data Placement	[54, 61, 66, 67, 69, 84, 95, 102, 104, 105]	10
Persistence Batching & Flush Optimisation	[40, 62, 72, 73, 92, 106, 107]	7
Endurance-Aware Compaction	[5, 78, 94, 98, 108, 109]	6
Persistent Logging	[63, 66, 72, 91, 106, 110]	6
Remote PMEM	[18, 56, 78, 85, 111]	5
PMEM-Native Engine	[59, 61, 89, 110]	4
Instant Recovery	[57, 76, 85, 105]	4
PMEM-Adapted LSM-Tree Design	[62, 70, 92, 103]	4
NUMA Optimisation	[66, 89, 112, 113]	4
Near-Data Compute	[56, 58, 60, 114]	4
Hierarchical Buffer Management	[60, 69, 71, 115]	4
Concurrency Control	[55, 116, 117]	3
Persistent Replication	[65, 83, 111]	3
PMM-Aware Allocator	[57, 116, 118]	3

We summarize the insights gathered from this analysis into five recurring design patterns, each cross-cut by methodological observations on evidence strength and open problems.

5.2.1 Tier-Conscious Storage Layouts: Almost 36% (25/60) of the domain-specific papers (see Table 8, rows “Multi-Tier Memory Hierarchy” and “Dynamic Data Placement”) adopt a three-tier hierarchy where hot metadata and indexes remain in DRAM, bulk pages migrate to PMem, and archival data sinks to SSD. Table 11 reports winsorized geometric mean speed-ups of 7.2× and 6.1× respectively, for Dynamic Data Placement and Multi-Tier Memory Hierarchy, though these findings are relative to the benchmarks and evaluation methods used for the implementations. An interesting pattern shown in Figure 2 is that both these individual types form an average of 0.48 Jaccard overlap with the Persistent Index optimization, as 12 papers co-occur between the redesign types. A key insight from this observation is that the tier split only pays off when paired with PMem-aligned data structures that exploit 256-byte write blocks and low-latency persistence barriers. For instance, systems like FlyDB [94] and NStore [66] combine hybrid placement with mmap- or PMDK-based index backends to achieve high throughput and efficient crash recovery.

5.2.2 PMem-Optimized Index Structures: Twenty percent of the database domain corpus (12/60) (see Table 8) focuses on redesigning B⁺ trees, radix tries, or hash tables, ensuring that every node is padded to Optane’s 256B write block and flushed with CLWB+SFENCE ordering [67, 68, 100]. The relevance of PMem-optimized index structures to database redesigns lies in how 256-byte aligned, byte-addressable writes translate into faster queries, immediate durability at commit, and lower-cost execution plans, leading to an impact spanning the entire engine rather than just the index code. Table 11 shows a winsorized geometric mean 7.1× lookup speed gain relative to the specific benchmark suites and evaluation methods adopted by the experiments. As mentioned in the previous subsection, Figure 2 reveals a strong co-occurrence ($J \approx 0.48$) with the two tiering families, underlining the

Table 9. Database domain redesign types and definition

Redesign pattern	Core architectural definition
Multi-tier memory hierarchy	Hot-cold placement across DRAM + PMM (optionally SSD); keep only metadata in DRAM.
Persistent index	Crash-consistent B ⁺ /Bw-trees, tries or hash tables stored directly in PMM.
Dynamic data placement (hot migration)	Runtime promotion/demotion between tiers based on access hotness and NUMA locality.
Persistence batching and flush opt.	Defer, coalesce or reorder cache-line flushes; exploit clwb/sfence.
Endurance-aware compaction	LSM tweaks—KV separation, small-value filters—to curb write amplification on PMM.
PMM-native engine	Ground-up row/column store that bypasses disk-era layers; operates directly on PMM.
Persistent logging	Append logs in PMM with byte-granular atomicity; drop fsync latency.
Remote PMM (disaggregation)	Access pooled PMM over RDMA; persistence becomes a rack-scale service.
Instant recovery	Keep working state in-place on PMM; restart by remapping pointers, no log replay.
PMM-adapted LSM-tree	Flatten levels or bypass L0 flushes to match PMM’s latency/endurance profile.
NUMA optimisation	Socket-local allocators/partitions to avoid cross-socket PMM traffic.
Near-data compute	Push compaction, filtering or logging into the PMM device / controller.
Concurrency control	Lock-free or optimistic protocols leveraging atomic PMM writes instead of heavy logging.
Hierarchical buffer management	Explicit DRAM to PMM to SSD migration with tier-aware eviction and prefetch.
Persistent replication	Mirror PMM-resident logs/data via RDMA or erasure-coding for μ s-scale durable commit.
PMM-aware allocator (space mgmt)	Crash-safe, low-fragmentation persistent heaps; log-friendly free-list updates.

correlation between the optimizations.

A key takeaway from the representative papers is that persistent indexing achieves high throughput and near-instant recovery by aligning index nodes with PMem’s native 256B block size and avoiding volatile-only metadata rebuilds. Systems such as PACTree [100], Viper [68], and TrieKV [67] report point lookup speed-ups of up to 15 \times and sub-second recovery at the 10GB scale by aligning index nodes to 256B blocks and persisting structure metadata, contributing to a median performance gain of 8.6 \times across all papers under this optimization (Table 11). At the same time, nine of the twelve articles nonetheless reconstruct a volatile DRAM search layer upon restart—an approach that exhibits poor scalability when the dataset exceeds several hundred gigabytes.

5.2.3 Write-Path Refactoring: From Level-0 Splits to Flush Coalescing: PMem’s asymmetric latency profile penalizes fine-grain, synchronous flushes, and we have observed two distinct optimization patterns that can mitigate this constraint. Redesigning for Persistence Batching/Flush Optimization and PMem-Adapted LSMTrees makes up around 18% of the domain-specific corpus (11/60) (see Table 8). These works rethink logging, memtable flushing, and level-0 persistence to eliminate write stalls and reduce persistence overhead. Representative systems include FlatLSM [62] and SplitDB [70], which redesign the flush and compaction mechanisms of LSMtrees to align writes with PMem’s 256B blocks, and NF-Log [106] along with the Persistent Memory I/O Primitives framework [92], which emphasize atomic, 256B-aligned page propagation and coalesced CLWB+SFENCE batching.

The key takeaway is that append-aligned designs optimized for PMem’s persistence domains—such as storing immutable level-0 tables directly on PMem or batching small writes into block-aligned units—achieve a winsorized geometric-mean speed-up of $5.9\times$ (Table 11). These strategies significantly reduce write amplification and tail latency. However, performance gains tend to level off beyond ~ 16 – 24 threads due to PMem bandwidth saturation, and most evaluations are limited to microbenchmarks or synthetic tests, with real-world and multi-node scalability still largely unexamined [62].



Fig. 2. Co-similarity matrix showing Jaccard Similarity for 16 Database redesign types

5.2.4 Locality-Aware Resource Pooling: PMem deployed across NUMA sockets or provided as a remote service via RDMA gives rise to locality-aware resource pooling, which covers almost 15% of the corpus (9/60) (see Table 8). It consists of the redesign themes NUMA Optimization and Remote PMem, which share a focus on binding data, threads, and persistence to physical locality to avoid costly cross-socket or network hops. The representative papers elucidate approaches such as partitioning logs and data structures per NUMA node and co-scheduling threads to eliminate inter-socket PMem traffic (NStore [66] and HybridKV [112]) and using one-sided RDMA writes and per-thread logs to build remote PMem pools (OpenEmbedding [105] and Replicating PMemKVS [111]).

Table 10. Redesign type abbreviations used in correlation matrix.

Abbreviation	Redesign Type
MT	Multi-Tier Memory Hierarchy Management
PI	Persistent Index
DD	Dynamic Data Placement
PB	Persistence Batching & Flush Optimization
CO	Endurance-Aware Compaction
PL	PMM Native Engine
RP	Persistent Logging
PN	Remote PMM
IR	Instant Recovery
PA	PMM-Adapted LSM-Tree Design
NO	NUMA Optimization
ND	Near Data Compute
HB	Concurrency Control
CC	Hierarchical Buffer Management
PR	Persistent Replication
PA	PMM Aware Allocators

The key insight gathered from this study is that making PMem locality explicit yields winsorized geometric mean gains of $6.2\times$ for NUMA-aware designs and $11.1\times$ for RDMA-based disaggregation (Table 11), relative to the benchmarks and evaluation methods used. However, throughput breaks once the socket or network bandwidth is saturated. Cold start times can exceed 50s for multi-TB pools [111], and most evaluations remain confined to synthetic or microbenchmarks—real-world, multi-node validation is still scarce. Another interesting insight is that multiple papers, such as “Persistent Memory Disaggregation for Cloud-Native Relational Databases” [78], call out CXL memory pooling as their future adaptation scenario.

5.2.5 PMem-Native Engine: This approach stands out by integrating OptaneDC PMem as a core design element rather than layering onto traditional engine architecture. Although the papers span less than 10% of the studies (4/60 papers; Table 8), this cluster delivers a winsorized geometric mean speed-up of $10.64\times$. Instead of incremental tweaks, PMem-native engine designs overhaul core data paths and buffer managers for the complete utilization of PMem’s byte-addressability, persistence, and NUMA-aware characteristics.

The representative systems have achieved substantial performance gains by integrating PMem into core engine data pathways. HiEngine [59] utilizes Partitioned Indirection Arrays over PMem to attain a $7.5\times$ speed-up in TPC-C, Hockey [89] incorporates an on-PMem delta store within query pipelines for a $3.2\times$ improvement in TPC-H, and Zen+ [61] employs a socket-local PMem buffer manager to achieve a $10.1\times$ increase in YCSB/TPC-C throughput. A prototype PM-native SQL engine [110] that rewrites storage layers reports up to a $50\times$ enhancement in microbenchmark performance—all contributing to a $10.64\times$ winsorized geometric-mean speed-up for this redesign.

The key takeaway from this redesign is that although these reconstructions for optimal utilization of PMem yield uniformly high gains without the usual thread-saturation issues seen in previous patterns, they incur integration and maintenance costs, and their portability across diverse DBMS kernels and workloads remains untested.

Table 11. **Per-redesign-type synopsis.** *Legend:* Benchmark rigor (0–5): one ★ each for real Optane, real workload/trace, 2 baselines, open artefact, endurance/scale (first three flags shown). Filled (★) and hollow (☆). Impact bucket (High, Medium, Low) combines winsorized G-Mean and rigor:

- High if G-Mean $\geq 3\times$ and rigor ≥ 3 (otherwise);
- Medium if G-Mean ≥ 1.5 (otherwise);
- Low if G-Mean < 1.5 or rigor < 2 ★.

Evidence cells show m/N matching papers, ✓ when $m = N$, ✗ when $m = 0$.

Winsorized G-Mean: winsorized geometric mean of per-paper gains (5% tails capped).

Redesign type	Evidence flags			Avg. Baselines	Win. G-Mean (\times)	Rigor	Impact Strength
	RealOpt	RealWrk	Reproducible				
Multi-Tier Memory Hierarchy Management	14/15	1/15	2/15	3.4	7.2	★★☆☆☆	Medium
Persistent Index	✓	1/12	4/12	4.0	7.1	★★★★☆	High
Dynamic Data Placement	✓	2/10	1/10	3.3	6.1	★★★★☆	High
Persistence Batching & Flush Opt.	✓	1/7	✗	1.9	5.9	★★☆☆☆	Medium
Endurance-Aware Compaction	✓	3/6	✗	3.0	2.8	★★★★☆	Medium
PMM Native Engine	✓	✗	✗	1.8	10.6	★★☆☆☆	Medium
Persistent Logging	✓	1/6	1/6	2.0	12.6	★★☆☆☆	Medium
Remote PMM	4/5	2/5	✗	2.6	11.1	★★★★☆	High
Instant Recovery	3/4	1/4	✗	2.5	10.7	★★★★☆	High
PMM-Adapted LSM-Tree	✓	1/4	1/4	2.5	2.8	★★★★☆	Medium
NUMA Optimization	✓	✗	✗	3.5	6.2	★★☆☆☆	Medium
Near Data Compute	3/4	2/4	✗	3.5	6.2	★★★★☆	High
Hierarchical Buffer Management	✓	✗	✗	4.3	13.0	★★★★☆	High
Concurrency Control	✓	✗	✗	2.0	3.9	★★☆☆☆	Medium
Persistent Replication	✓	1/3	✗	4.0	3.1	★★★★☆	High
PMM Aware Allocators	✓	✗	✗	1.7	1.9	★★☆☆☆	Low

Table 12. PMEM redesigned for Database engines: representative papers, benchmark practice and remaining gaps.

Redesign type	Representative paper	Common benchmarks	Coverage gaps
Multi-tier memory hierarchy mgmt	<i>Y. Zhang et al. [98]</i>	YCSB, DB_Bench, TPC-H, YCSB-TS, Meituan	Tuning complexity, crash-consistency limits, scalability bottlenecks
Persistent index	<i>H. Kumar et al. [119]</i>	YCSB, DB_Bench, production trace	Incomplete recovery paths; cost-performance knobs under-explored
Dynamic data placement	<i>C. Chen et al. [105]</i>	YCSB, DB_Bench, TPC-C, YCSB-TS, DLRM	Migration overheads, NUMA placement, recovery-path design gaps

Continued on next page

Table 12 continued from previous page

Redesign type	Representative paper	Common benchmarks	Coverage gaps
Persistence batching & flush opt.	<i>A. van Renen et al. [92]</i>	YCSB, DB_Bench, TPC-C, YCSB-TS, DLRM	Flush-granularity tuning and thread sensitivity; limited recovery eval.
Endurance-aware compaction	<i>J. Wang et al. [109]</i>	YCSB, TPC-C	Integration hurdles; CPU/tail-latency trade-offs unresolved
PMM-native engine	<i>Y. Ma et al. [59]</i>	TPC-C, TPC-H, YCSB	Scalability implications and recovery support still vague
Persistent logging	<i>B. Zheng et al. [63]</i>	YCSB, TPC-C, YCSB-TS	Sparse crash-recovery detail; limited large-scale evaluation
Remote PMM	<i>J. Sun et al. [56]</i>	YCSB, TPC-C, TPC-H	Cost-performance and DDIO trade-offs; portability and recovery scale
Instant recovery	<i>A. Baumstark et al. [76]</i>	YCSB, TPC-C, TPC-H, DLRM	No GC tuning; lacks evaluation on mixed OLTP/BI workloads
PMM-adapted LSM-tree	<i>M. Cai et al. [70]</i>	YCSB, TPC-C, DB_Bench	Profiling and recovery-cost modelling unclear
NUMA optimisation	<i>Z. Wang et al. [66]</i>	YCSB, TPC-H, DB_Bench, FIO	NUMA scalability thin; crash-consistency resilience lacking
Near-data compute	<i>Y. Seneviratne et al. [114]</i>	YCSB, TPC-C	Non-instant recovery; CPU contention and cost-perf trade-offs
Concurrency control	<i>Y. Chen et al. [55]</i>	YCSB, TPC-C, PARSEC	NUMA constraints; relies on manual timestamp/memory tuning
Hierarchical buffer mgmt	<i>X. Zhou et al. [60]</i>	YCSB	Random-read penalties; GC/lock contention under skewed loads
Persistent replication	<i>Q. Wang et al. [111]</i>	YCSB, TPC-C	Cold-start latency; DDIO penalties; limited fault-tolerance study
PMM-aware allocators	<i>X. Qi et al. [118]</i>	YCSB, TPC-C, PARSEC	Inefficient GC at high concurrency; hardware-specific tuning burden

5.2.6 Cross-Cut Observations: While the five redesign patterns capture concrete engine changes, three overarching trends span the entire corpus, revealing both strengths and blind spots in current OptaneDCPMM integration research.

First, benchmark rigor remains unsatisfactory. Although many papers boast dramatic speed-ups, only seven papers [56, 78, 101–103, 105, 111] earn four-star ratings for benchmark quality, and of those, only four papers [56, 92, 101, 103] combine high rigor with greater throughput gains. As these seven papers span a variety of redesign themes, not a single redesign theme has an average benchmark rating climbing over three stars (Table 11), and under 15% of studies exercise real-world workloads. Consequently, even winsorized geometric-mean gains should be interpreted as directional signals rather than definitive performance guarantees.

Second, endurance is largely ignored. Despite OptaneDC’s advertised 200PBW(Petabytes Written) endurance, only two papers actually measure wear—Perach et al. [85] and Wang et al. [111]—leaving a critical reliability dimension of persistent memory underexplored.

Finally, CXL readiness is already on the horizon. Several papers across the RemotePMM/Disaggregation, Endurance-Aware Compaction, and PMem-Adapted LSMTree clusters cite “adapt to CXL” in their future work

sections [78, 98], signaling the transcendence of Optane DCPMem lessons to next-generation CXL-attached byte-addressable memory platforms.

5.3 Data Structures Redesigns

Persistent memory technologies have fundamentally challenged the assumptions underlying traditional data structure design, forcing researchers to reconsider decades of optimization strategies built for volatile memory systems. Our comprehensive survey identifies over 30 distinct redesign patterns across 160 research papers, revealing a field in active transition. This section examines the architectural shifts that have emerged in response to persistent memory’s unique characteristics, analyzes the patterns of research activity, and highlights the persistent challenges that limit practical adoption.

5.3.1 From DRAM-Optimized to PMem-Native Design: The transition from volatile DRAM to persistent memory has required researchers to fundamentally reconsider data structure design principles that have remained largely unchanged for decades. Traditional structures, optimized for DRAM’s uniform access patterns and volatility assumptions, prove inadequate when confronted with PMem’s distinctive characteristics: byte-addressable persistence, asymmetric read/write performance, and finite write endurance. Through our systematic analysis, three dominant design patterns have emerged as solutions to these constraints: Write-Efficient Architectures represent the most widespread adaptation, with researchers developing techniques to minimize write amplification. The Write-Combining Buffer (WC) pattern, documented across 37 papers, exemplifies this approach by batching small updates in volatile memory before committing them as single, cache-line-aligned writes to persistent storage. Similarly, Ground-Up Copy-On-Write (GU) designs avoid costly in-place modifications by allocating new versions and atomically updating root pointers. These strategies acknowledge that PMem’s write characteristics fundamentally alter the cost model that guided traditional data structure optimization.

Crash-Consistent by Design approaches embed durability guarantees directly into data structure architectures rather than relying on external mechanisms. The Cache-Line Transaction (CL) pattern, appearing in 40 papers, demonstrates this philosophy by incorporating version numbers and checksums within individual cache lines, enabling recovery through hardware primitives without separate logging infrastructure. This integration of consistency mechanisms represents a departure from the traditional separation of data organization and durability concerns.

Tiered Memory Awareness acknowledges that uniform treatment of data in persistent memory may be suboptimal. The Hot-Cold Page Migrator pattern, the most extensively studied approach with 75 associated papers, implements dynamic data placement strategies that maintain frequently accessed data in DRAM while utilizing PMem for less active content. These hybrid approaches attempt to preserve the performance characteristics of volatile memory while gaining the capacity and persistence benefits of PMem.

Table 13. Data Structure Redesign Patterns with the Core Architectural Idea

Redesign Pattern	Core Architectural Idea
Write-Combining Buffer	Accumulate many small (256 B) updates in DRAM and flush them as single 256 B writes to PMem to cut write amplification and fence overhead.
Segment-Range Lock	Replace fine-grained inode locks with coarse byte-range locks on PMem segments to allow concurrent appends without cache-line bouncing.

Continued on next page

Table 13 (continued)

Redesign Pattern	Core Architectural Idea
Cache-Line Transaction	Embed version and checksum metadata inside each 64 B cache line to get atomic durability via clwb+sfence alone—no external log needed.
Epoch-Buffered Persistence	Buffer dirty lines and rely on eADR to flush an entire epoch on failure, avoiding per-write fences (or use epoch barriers/checkpoints otherwise).
Flat Namespace BR-Tree	Use a hash-keyed, bounded-range radix/B-tree so a single PMem pointer chain replaces multi-level directory traversals.
Learned Node Index	Embed a tiny regression model and overflow buffer in each 256 B PMem node to shrink tree depth and pointer chasing.
Compact-Flush Hash	Organize metadata-free extendible-hash segments written once, and batch atomic directory doublings into aligned 256 B flushes.
Ground-Up Copy-On-Write	Design the entire data structure around copy-on-write by allocating new versions and atomically swapping a root pointer—no log needed.
Persistent Cuckoo Tier	Guard a PMem B*-tree with a DRAM cuckoo filter so that more than 90% of negative lookups avoid costly PMem reads.
Sparse Pointer Compression	Replace 64-bit pointers with 32-bit offsets in PMem blocks to halve pointer traffic and reduce bandwidth use.
Vector-Pointer Cluster Graph	Store the CSR index in PMem and keep vector clusters in DRAM, streaming PMem edges sequentially to hide latency.
Three-Tier Anti-Cache	Maintain an append-only hierarchy of DRAM cache → PMem spill → SSD backend to keep hot data fast and cold data cheap.
Hot-Cold Page Migrator	Use an online ski-rental algorithm to move hot pages to DRAM and cold pages to PMem, hiding Optane's higher access latency.
Profile-Guided Object Hoist	Use static or dynamic profiling to tag hot objects and hoist them to DRAM on the next launch, ensuring latency-critical data stays fast.
Hybrid Slab-Variable Allocator	Keep small fixed-size slabs in DRAM and allocate larger (256 B) blocks in PMem with single-flush metadata for alignment.
Per-Thread Bump Pool	Allocate from thread-local bump-pointer arenas in PMem, reclaimed via epoch GC, to eliminate allocator locks at microsecond scales.
Wear-Levelling Bitmap Map	Track write counts globally in a bitmap and migrate hot blocks to cooler DIMMs to even out wear and extend Optane's lifetime.
Dual-Logger Undo-Redo	Use per-thread 64 B redo entries plus a global 16 B undo link to achieve constant-time commit/abort with just one cache-line flush.
Shadow-Replay Execution	Log high-level semantic operations (e.g., "insert key X") and replay them on recovery to minimize PMem writes and simplify logic.
Selective WAL Buffer	Buffer the redo log in DRAM and flush lazily, reconstructing metadata from payload headers after a crash to reduce sync PMem writes.
Persistent State-Machine Log	Log semantic operations and rebuild a shadow state on recovery—avoiding persistence of intermediate memory states.
DMA Hash Consolidator	Offload hash-bucket writes to NIC/DPU DMA in large stripes, overlapping Optane's write latency with network transfers.
RDMA Hitchhiked Lock	Embed the lock bit in a one-sided RDMA write to eliminate an extra network round-trip for mutual exclusion.
SmartNIC Metadata Cache	Cache inode log-tail pointers in NIC SRAM, only touching PMem when overflows occur to cut frequent metadata reads.
Hash Bucket DPU Cache	Use a two-tier hash table with a DPU SRAM cache and PMem buckets that are step-merged to keep hot buckets in-core.

Continued on next page

Table 13 (continued)

Redesign Pattern	Core Architectural Idea
Overlap-Aware Compaction	During LSM compaction, only merge overlapping key ranges and skip clean blocks to minimize PMem write traffic.
Compressed Level-Zero	Keep Level-0 SSTables compressed in PMem and only inflate them during compaction to reduce both read and write volume.
Hot-Path Learned Prefetch	Use a tiny learned model to predict and software-prefetch the next PMem node, turning random reads into near-sequential ones.
Tensor Phase Migrator	Bucket tensors by reuse distance, pin active ones in DRAM and checkpoint cold ones in PMem to balance DL performance and capacity.
Epoch Buffer Queue	Persist POSIX wait-queues as PMem circular buffers so user threads can poll without costly kernel crossings.

5.3.2 Quantification of Design Patterns: The distribution of research effort across different redesign patterns provides insight into community priorities and reveals underlying technical challenges. Analysis of co-occurrence relationships among the ten most studied redesign types, presented in 3, identifies three coherent research clusters that have emerged organically within the field:

The Hardware-Optimization Cluster encompasses Write-Combining Buffer (WC), Cache-Line Transaction (CL), and Segment-Range Lock (SR) approaches, which exhibit strong co-occurrence patterns in the literature. This cluster, representing approximately 30% of surveyed papers, focuses on exploiting low-level hardware characteristics of persistent memory devices. Researchers working in this area typically combine multiple hardware-aware optimizations, suggesting that effective PMem utilization requires coordinated manipulation of cache line alignment, atomic write guarantees, and memory ordering constraints.

The System-Architecture Cluster groups Hot-Cold Page Migrator (HC), Profile-Guided Object Hoist (PG), and Hybrid Slab-Variable Allocator (HS) techniques around system-level memory management concerns. The prevalence of HC research (75 papers) within this cluster indicates that managing PMem’s latency characteristics through intelligent data placement remains a central challenge. The clustering suggests that effective memory management in persistent memory systems requires coordination across allocation strategies, access pattern prediction, and dynamic data migration policies.

The Consistency-Recovery Cluster consolidates Ground-Up Copy-On-Write (GU), Dual-Logger Undo-Redo (DL), and Selective WAL Buffer (SW) approaches around crash consistency mechanisms. This grouping reflects the fundamental challenge of maintaining data integrity in systems where memory contents survive system failures, requiring novel approaches to transaction processing and recovery protocols.

The quantitative similarity analysis presented in 4 reveals that apparently distinct techniques often share substantial implementation strategies. The high Jaccard similarity coefficient between WC and CL approaches (exceeding 0.6) indicates convergent evolution toward hardware-aware, write-efficient designs, suggesting that successful PMem data structures must address multiple constraints simultaneously rather than optimizing for individual characteristics.

5.3.3 Evaluation Quality and Impact Assessment. A systematic evaluation of benchmark practices and impact metrics across the surveyed literature reveals concerning patterns in the progress of persistent-memory data-structure

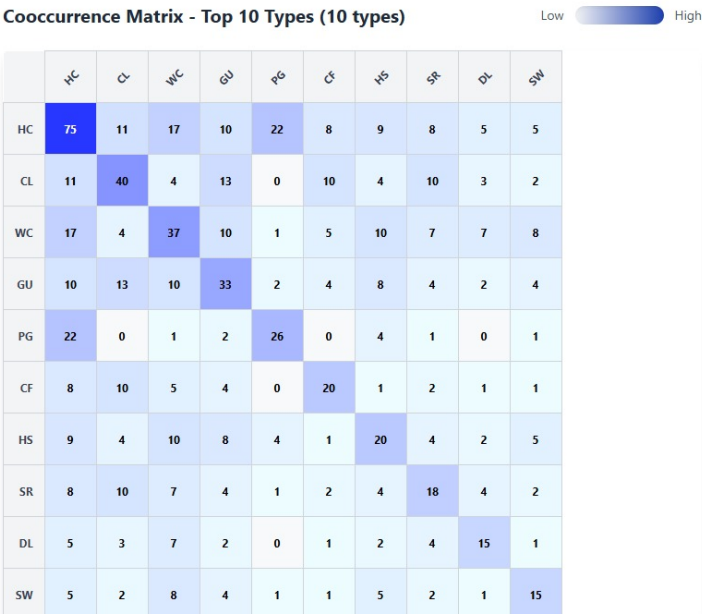


Fig. 3. Co-occurrence matrix for top 10 Data Structures redesign types



Fig. 4. Co-similarity matrix showing Jaccard Similarity for top 10 Data Structures redesign types

Table 14. Redesign type abbreviations used throughout the Data Structures Section.

Abbreviation	Redesign Type
HC	Hot-Cold Page Migrator
CL	Cache-Line Transaction
WC	Write-Combining Buffer
GU	Ground-Up Copy-On-Write
PG	Profile-Guided Object Hoist
CF	Compact-Flush Hash
HS	Hybrid Slab-Variable Allocator
SR	Segment-Range Lock
DL	Dual-Logger Undo-Redo
SW	Selective WAL Buffer

research. Our assessment, summarized in Table 15, applies a standardized five-point rigor scale to evaluate both experimental methodology and practical impact; we find that most redesigns achieve only moderate benchmark rigor while delivering incremental performance improvements.

Methodological Limitations in Evaluation Practices. The predominant reliance on YCSB as an evaluation benchmark—used in 25 of the 30 redesign categories—highlights a critical limitation in current research methodology. While YCSB provides standardized key-value access patterns that enable cross-study comparisons, its synthetic workloads fail to capture the complexity and diversity of real-world applications. This methodological homogeneity may yield optimization strategies that perform well under controlled conditions but do not generalize to production environments. More concerning is the complete absence of benchmark evaluation for six redesign approaches—including promising techniques such as Persistent Cuckoo Tier and Vector-Pointer Cluster Graph—limiting our understanding of their practical utility.

Impact Assessment and Performance Gains. The narrow distribution of impact scores across all redesigns (high, medium, low in our scale) suggests that, despite substantial research investment, transformational performance improvements remain elusive. Network and acceleration optimizations demonstrate slightly higher impact scores; for example, RDMA Hitchhiked Lock achieves a rating of 3.0, indicating potential for cross-layer strategies that extend beyond traditional memory-hierarchy concerns. However, the clustering of most approaches around 2.0–2.5 indicates that current research has produced incremental rather than breakthrough gains.

Hardware Dependency and Practical Constraints. Redesigns with the highest benchmark-rigor scores tend to rely on specialized hardware platforms such as SmartNICs and Data Processing Units (DPUs). Although these approaches demonstrate technical sophistication, their dependence on non-commodity hardware creates adoption barriers that may limit practical significance. This pattern reveals a trade-off between technical innovation and deployability that the research community has yet to resolve effectively.

Table 15. **Data Structure Redesigns in Persistent Memory Systems: Overview and Metrics** *Legend:* Benchmark rigor (0–5): one ★ each for real Optane, real workload/trace, 2 baselines, open artefact, endurance/scale (first three flags shown). Filled (★) and hollow (☆). Impact bucket (High, Medium, Low) combines impact and rigor:

- High if Impact $\geq 3\times$ and rigor ≥ 3 (otherwise);
- Medium if Impact ≥ 1.5 (otherwise);
- Low if Impact < 1.5 or rigor < 2 ★.

Redesign Types	Redesign Subtypes	Benchmark Rigor	Impact Strength
Hardware-Level Optimizations	Write-Combining Buffer	★★★★☆	Medium
	Segment-Range Lock	★★★★☆	Medium
	Cache-Line Transaction	★★★★☆	Medium
	Epoch-Buffered Persistence	★★★★☆	Medium
Data Structure Redesigns	Flat Namespace B ⁺ -Tree	★★★★☆	Medium
	Learned Node Index	★★★★☆	Medium
	Compact-Flush Hash	★★★★☆	Medium
	Ground-Up Copy-On-Write	★★★★☆	Medium
	Persistent Cuckoo Tier	★★★★★	Medium
	Sparse Pointer Compression	★★★★☆	Medium
	Vector-Pointer Cluster Graph	★★★★☆	Medium
System/Architecture-Level Designs	Three-Tier Anti-Cache	★★★★☆	Medium
	Hot-Cold Page Migrator	★★★★☆	Medium
	Profile-Guided Object Hoist	★★★★☆	Medium
	Hybrid Slab-Variable Allocator	★★★★☆	Medium
	Per-Thread Bump Pool	★★★★☆	Medium
	Wear-Levelling Bitmap Map	★★★★☆	Medium
Logging and Recovery Optimizations	Dual-Logger Undo-Redo	★★★★☆	Medium
	Shadow-Replay Execution	★★★★☆	Medium
	Selective WAL Buffer	★★★★☆	Medium
	Persistent State-Machine Log	★★★★☆	Medium
Network and Acceleration Optimizations	DMA Hash Consolidator	★★★★★	Medium
	RDMA Hitchhiked Lock	★★★★☆	High
	SmartNIC Metadata Cache	★★★★★	Medium
	Hash Bucket DPU Cache	★★★★★	Medium
Storage-Level and Indexing Optimizations	Overlap-Aware Compaction	★★★★☆	Medium
	Compressed Level-Zero	★★★★☆	Medium
Prefetch and Access Optimizations	Hot-Path Learned Prefetch	★★★★★	Medium
	Tensor Phase Migrator	★★★★★	Low
	Epoch Buffer Queue	★★★★★	Medium

5.3.4 Research Gaps and Systemic Challenges: The comprehensive gap analysis presented in Table 16 reveals systemic limitations that transcend individual redesign approaches and point to deeper structural challenges

within the field. These limitations can be organized into four critical categories that collectively constrain the practical adoption of persistent memory data structures:

Toolchain and Ecosystem Maturity: A pervasive challenge across multiple redesign categories is the absence of advanced compiler and runtime support for persistent memory programming constructs. Techniques such as Cache-Line Transaction and SmartNIC-based approaches require manual implementation of low-level memory management primitives, often necessitating inline assembly programming and specialized hardware interfaces. This toolchain gap creates a significant barrier between research prototypes and production deployments, as developers cannot rely on standard software development practices when implementing PMem-optimized data structures.

Concurrency and Scalability Limitations: Multi-threaded scalability remains poorly understood across the majority of surveyed redesigns. The documented scalability limitations of the Persistent State-Machine Log in concurrent environments exemplify a broader pattern where research has prioritized single-threaded performance optimization over the coordination challenges inherent in multi-threaded persistent memory systems. This focus has resulted in techniques that may perform well in controlled experimental settings but fail to maintain their advantages under realistic concurrent workloads.

Validation Methodology Deficiencies: The heavy reliance on synthetic benchmarks has created a validation gap where optimizations tuned for benchmark workloads may not translate to production performance gains. This limitation is particularly evident in approaches such as Hot-Path Learned Prefetch, which demonstrates promise under controlled access patterns but lacks validation against the irregular and unpredictable access patterns characteristic of real applications. The absence of standardized real-world workloads hampers the field’s ability to assess the practical significance of proposed optimizations.

Integration and Composition Challenges: While individual redesign techniques may demonstrate merit in isolation, their integration into complete systems remains largely unexplored. The complexity of coordinating multiple PMem optimizations—such as combining write-combining strategies with crash consistency mechanisms—can produce emergent behaviors that current research methodologies fail to address adequately. This limitation reflects a broader tendency toward point optimization rather than systems-level thinking in persistent memory research.

Table 16. Data Structure Redesign Types with Associated Benchmarks and Coverage Gaps

Redesign Type	Common Benchmarks	Coverage Gaps
Write-Combining Buffer	FILEBENCH, MLC, STREAM, TPC-C, YCSB	shallow exploration of tuning effects
Segment-Range Lock	YCSB	fine-grained locking overheads in concurrency
Cache-Line Transaction	STREAM, TPC-C, YCSB	formal semantics, limited compiler support, few tools
Epoch-Buffered Persistence	YCSB	multi-threaded coordination cost, coarse epoch granularity
Flat Namespace BR-Tree	FILEBENCH, MLC, TPC-C, YCSB	scalability for large metadata, weak concurrency guarantees
Learned Node Index	YCSB	poor scalability in update-heavy distributed workloads
Compact-Flush Hash	FILEBENCH, MLC, STREAM, TPC-C, YCSB	high write amplification, update complexity

Continued on next page

Table 16 (continued)

Redesign Type	Common Benchmarks	Coverage Gaps
Ground-Up Copy-On-Write	TPC-C, YCSB	metadata overhead
Persistent Cuckoo Tier	YCSB	high write amplification, poor concurrency under updates
Sparse Pointer Compression	YCSB	workload diversity, lacks real-world system validation
Vector-Pointer Cluster Graph	Micro-benchmarking	multi-node scalability gaps, dataset limitations
Three-Tier Anti-Cache	YCSB	tier rebalancing overhead, cross-tier consistency
Hot-Cold Page Migrator	FILEBENCH, STREAM, TPC-C, YCSB	Lacks NVM bandwidth management policy, no deployment/endurance results
Profile-Guided Object Hoist	STREAM, TPC-C, YCSB	lacks runtime adaptability
Hybrid Slab-Variable Allocator	STREAM, TPC-C, YCSB	fragmentation challenges
Per-Thread Bump Pool	YCSB	fragmentation, dynamic resizing, poor integration
Wear-Levelling Bitmap Map	TPC-C, YCSB	lacks support for spatial workloads, small-scale evaluation
Dual-Logger Undo-Redo	YCSB	contention recovery scalability, tiered memory support
Shadow-Replay Execution	YCSB	fine-grained replay overhead, coordination issues
Selective WAL Buffer	TPC-C, YCSB	crash-consistency vs wear-leveling tradeoffs, scalability
Persistent State-Machine Log	YCSB	scalability gaps in multi-threaded environments
DMA Hash Consolidator	FILEBENCH, FIO, TPC-C	DMA platform limitations, underexplored energy efficiency
RDMA Hitchhiked Lock	Filebench, TPC-C, YCSB	missing failure recovery, RDMA scalability concerns
SmartNIC Metadata Cache	TPC-C, YCSB	cache consistency, lack of SmartNIC tooling
Hash Bucket DPU Cache	YCSB	CXL crash recovery, real-world indexing gaps
Overlap-Aware Compaction	YCSB	limited generalization to non-LSM stores, no energy insights
Compressed Level-Zero	YCSB	limited scalability analysis beyond IoT, energy overheads
Hot-Path Learned Prefetch	STREAM, YCSB	lack of real-world benchmarks, untested adaptability
Tensor Phase Migrator	Filebench, TPC-C, YCSB	scalability limits, runtime adaptability missing
Epoch Buffer Queue	YCSB	limited HPC context, no fault-tolerance modeling

5.3.5 Research Momentum: The distribution of research attention across different redesign approaches, quantified in Table 15, provides insight into the field’s evolution and suggests directions for future investigation. The pronounced variation in paper counts—from 75 papers addressing Hot-Cold Page Migrators to just 2 papers examining Persistent Cuckoo Tiers—reveals both progressive research areas and underexplored opportunities.

Saturated Research Domains: The extensive coverage of Hot-Cold Page Migrator techniques (75 papers) and Cache-Line Transaction approaches (40 papers) suggests that these areas have reached a point of diminishing returns from incremental research contributions. The high paper count for Hot-Cold migration techniques reflects the fundamental challenge of managing PMem’s latency characteristics, but the persistence of this research focus may indicate that current approaches have not yet provided satisfactory solutions. Future work in these areas would benefit from more fundamental reconsiderations of the underlying assumptions rather than continued refinement of existing techniques.

Emerging Research Opportunities: Several techniques with limited current coverage represent potentially high-impact research directions. Learned Node Index approaches (8 papers) and Tensor Phase Migrator techniques demonstrate early-stage exploration of machine learning integration with persistent memory systems. These areas may benefit from increased research attention, particularly as the broader systems community develops more sophisticated approaches to learned system optimization.

Research Fragmentation and Consolidation Needs: The proliferation of distinct redesign patterns with 30 separate categories identified in our survey suggests that the field may be experiencing excessive fragmentation. While this diversity indicates healthy exploration of the design space, it also points to a need for consolidation frameworks that can integrate multiple optimization strategies coherently. The moderate co-occurrence values in Figure 2 suggest that most researchers are working within narrow technical domains rather than developing comprehensive approaches that address multiple PMem challenges simultaneously.

Table 17. Data Structure Redesign Types with Member Papers and Counts

Redesign Type	Member Papers	Count
Write-Combining Buffer	[2, 65, 67, 73, 88, 99, 111, 117, 120–147]	37
Segment-Range Lock	[15, 88, 101, 113, 120, 121, 124, 126, 132, 145, 148–154]	18
Cache-Line Transaction	[4, 9, 104, 113, 117, 126, 129, 133, 148–179]	40
Epoch-Buffered Persistence	[88, 135, 153, 174, 180–186]	11
Flat Namespace BR-Tree	[73, 99, 147]	3
Learned Node Index	[9, 67, 99, 104, 138, 157, 187, 188]	8
Compact-Flush Hash	[73, 102, 121, 129, 141, 147, 148, 165–167, 169, 170, 173, 175, 178, 187, 189–192]	20
Ground-Up Copy-On-Write	[9, 65, 66, 102, 104, 120, 133, 137, 142–146, 151, 154, 155, 159, 163–165, 168, 172, 173, 180, 183, 184, 189, 193–198]	33
Persistent Cuckoo Tier	[119, 136]	2
Sparse Pointer Compression	[100, 109, 152, 190, 191, 199, 200]	7
Vector-Pointer Cluster Graph	[100, 152, 201]	3
Three-Tier Anti-Cache	[1, 88, 202–205]	6
Hot-Cold Page Migrator	[1, 51, 65–67, 88, 101, 102, 120, 122, 123, 126, 127, 131–134, 136, 137, 140, 141, 148, 150, 152, 160, 164, 173, 175, 178, 182, 185, 191, 192, 194, 195, 201–203, 205–239]	75
Profile-Guided Object Hoist	[1, 51, 101, 123, 194, 195, 202, 209–212, 214–216, 218, 220–222, 234–237, 240–243]	26
Hybrid Slab-Variable Allocator	[67, 99, 101, 102, 113, 117, 120, 128, 131, 133, 142, 145, 146, 179, 197, 198, 204, 209, 237, 244]	20
Per-Thread Bump Pool	[117, 145, 146, 198, 239]	5
Wear-Levelling Bitmap Map	[104, 176]	2
Dual-Logger Undo-Redo	[5, 9, 88, 99, 111, 120, 121, 132, 135, 161, 162, 180, 181, 208, 213]	15
Shadow-Replay Execution	[67, 88, 142–144, 174, 176, 179, 183, 198, 237, 245]	12
Selective WAL Buffer	[2, 104, 117, 121, 128, 131, 135, 142, 145, 195, 205, 217, 239, 246, 247]	15
Persistent State-Machine Log	[100, 198, 208]	3
DMA Hash Consolidator	[102, 130, 186, 193, 248–250]	8
RDMA Hitchhiked Lock	[168, 186, 251, 252]	4
SmartNIC Metadata Cache	[130, 186]	2
Hash Bucket DPU Cache	[130]	1
Overlap-Aware Compaction	[2, 67, 94, 98, 109, 119–121, 144, 187, 237, 253]	12
Compressed Level-Zero	[98, 254]	2
Hot-Path Learned Prefetch	[102, 109, 122, 206, 217, 220, 230]	7
Tensor Phase Migrator	[202, 214]	2
Epoch Buffer Queue	[255]	1

5.4 Use of App Direct vs Memory Mode

Across databases, data structures, and file systems, the overwhelming majority of works utilize App Direct mode. Although most papers across all domains do not explicitly specify reasons for their choice, those provided typically share common themes. App Direct is chosen primarily for its fine-grained control, allowing direct, byte-addressable access that bypasses the kernel's page cache and block-I/O layers. Researchers frequently highlight the benefits of explicitly placing specific indexes or logs on PMem, using precise flush instructions to guarantee ordering and persistence, and minimizing overhead from system calls, DRAM-PMem data copies, and write amplification.

Only a small fraction of studies (approximately 3%) used Memory Mode, typically in scenarios demanding large volatile-memory expansion with minimal software modifications. These papers emphasize ease of adoption and transparency, especially beneficial for legacy applications or HPC workloads. For instance, one HPC study ran memory-intensive scientific computations with Optane in Memory Mode, aiming to leverage high capacity (3–4× memory size) without altering the application code [256]. They reported effective capacity scaling, though performance varied greatly depending on memory access patterns—sequential accesses remained fast due to effective DRAM caching, whereas random accesses suffered from DRAM cache thrashing and incurred PMem latency.

Additionally, Mixed Mode configurations, utilizing both App Direct and Memory Mode simultaneously, appeared in roughly 7% of studies. These were chosen mainly to achieve the benefits of both modes concurrently: providing fast, explicitly managed storage through App Direct for critical data structures or logs, while using Memory Mode to transparently expand general-purpose memory for less performance-critical portions.

The survey clearly indicates that App Direct mode dominates research (90%), highlighting a strong consensus that significant performance benefits arise when applications and systems are explicitly engineered for persistent memory semantics. Conversely, Memory Mode is limited to cases requiring immediate memory expansion without the possibility or willingness to rewrite applications, confirming a trade-off between ease-of-use and maximum performance gain through explicit management.

6 EVALUATION METHODOLOGY

RQ2 focuses on the benchmark suites and evaluation methodologies that provide fair evaluation of Optane DC Pmem and their limitations; this section will cover the said issue. To fairly evaluate systems built on Optane persistent memory, researchers have employed a variety of benchmarks, metrics, and experimental setups. We observed a strong emphasis on microbenchmarking and comparative evaluation to isolate PMem benefits, as well as efforts to test crash consistency and other unique aspects of persistence. Eight benchmark suites appear most commonly in PMem studies (see Table 18, with paper counts from our survey):

- **YCSB** (key-value / NoSQL workloads): configurable read-/write-heavy mixes across Workloads A–F (throughput ops/s, average tail latency). Used in 219 papers—chiefly to assess PMem-aware key-value stores, databases, and storage engines under realistic mixed-operation patterns.
- **TPC-H / TPC-DS** (OLAP): complex, ad-hoc SQL queries. Used in 4 papers to quantify analytical throughput and runtime (QphH, seconds), often to show that PMem can accelerate decision-support workloads.
- **Filebench** (file-system I/O): file-level create/read/write operations (IOPS, MB/s, latency). Used in 59 papers—mostly to exercise metadata and mixed I/O workloads on PMem-aware file systems.

- **IOzone / NVMFS-IOzone** (file I/O): sequential and random read/write and rewrite tests on 512B records (MB/s, latency). Used in 5 papers, primarily single-client, micro-level I/O characterizations without transactional semantics.
- **Sysbench-OLTP** (OLTP): record-scan and point read/write operations (ops/s, latency in ms). Used in 3 papers to compare PMem-backed transactional engines, though it does not model the full TPC-C mix.
- **db_bench** (key-value store): point reads and writes on 1KB records (throughput, latency). Used in 17 papers—almost exclusively with RocksDB—to microbenchmark KV workloads on PMem.
- **DBT-3 (OLTP)**: TPC-C-like transaction mix (throughput, latency). Used in 2 papers as an open-source TPC-C proxy to evaluate transactional performance on PMem.
- **BigDataBench** (big-data analytics): batch analytics, ML tasks, and cloud workload traces (runtime, throughput, efficiency). Used in 4 papers to study higher-level PMem usage in data-parallel frameworks, though its end-to-end workloads can obscure isolated PMem effects.
- **Lmbench** (system microbenchmark): instruction-level latency, syscall, and bandwidth tests (latency, bandwidth). Used in 4 papers for low-level memory and OS kernel measurements, but it provides only a micro-view rather than application-level insights.

The metrics most commonly reported across these studies are throughput (operations per second) and latency (average and tail, e.g. 95th/99th percentile). Other occasional metrics include CPU utilization, write amplification, energy or power consumption, and scalability (e.g. throughput vs. thread count or NUMA effects). Baseline comparisons almost always include: *DRAM baseline* (volatile upper bound), *SSD or disk baseline* (storage-mode comparison), and often *other PMem systems* (e.g. NOVA, PMFS, SplitFS).

Finally, while performance dominates, about 8% of papers also incorporate automated crash-consistency testing (e.g. via DURINN), and many detail their hardware setups (Xeon + Optane DCPMems, DAX, PMDK, NUMA pinning) to aid reproducibility.

Despite the areas covered, gaps remain, especially in standardized PMem-specific benchmarks, multi-node/distributed evaluations, and long-term endurance testing. Going forward, more unified, and comprehensive evaluation frameworks will be essential to ensure a fair and reproducible evaluation.

7 APPLICATION SCENARIOS

As mentioned before, RQ3 focuses on the existing and potential applications for byte-addressable PMem; this section will address said issue. Persistent memory has evolved from a niche trial to a core technology in various fields—real-time search, key-value stores, embedding servers, HPC workflows, and graph analytics [76, 105, 119, 120, 129]. All are connected by the requirement for byte-addressable, high-capacity, crash-consistent storage that offers DRAM-like access characteristics [52]. Researchers commonly keep frequently accessed metadata in DRAM while storing larger data sets on OptaneDC modules [97]. They employ log-structured hashing and lock-free designs to curb write amplification [121, 159], and they use RDMA with zero-copy checkpointing to reinforce persistence in distributed or GPU-accelerated environments [118, 150, 193]. Looking ahead, tighter integration of next-generation PMem with CXL fabrics and GPU memory could yield unified memory-storage layers that support large in-memory databases, low-latency inference, and rapid recovery [72, 78, 257]. Work on persistent learned indexes, near-memory computation, and programmable persistence frameworks aims to advance performance, energy efficiency, and capacity [87, 114, 157].

Persistent memory and hybrid memory technologies have also reshaped durability-versus-speed assumptions in database systems [257]. They now power cloud-native deployments [246], NoSQL stores [73], IoT databases and

Table 18. Benchmark Characteristics and Usage

Benchmark	Domain	Access Pattern	Granularity	Metrics	# Papers
Intel MLC	Memory	Random load probes	64 B cache line	Latency (ns)	6
STREAM	Memory	Sequential stream-ing	8–64 B elements	Bandwidth (GB/s)	24
FIO	Block I/O	Configurable rand/seq R/W	4 KB–1 MB blocks	IOPS, Latency	59
YCSB	Key-Value Store	Record-level R/W mixes	~ 1 KB record	Ops/s, Latency (ms)	219
TPC-C / TPROC-C	OLTP	Mixed transactional mix	Transaction	tpmC, response time	76
TPC-H / TPC-DS	OLAP	Complex SQL queries	Query	QphH, runtime (s)	4
Filebench	File-system I/O	File-level create/read/write	File ops	IOPS, MB/s, Latency	59
IOzone / NVMFS-IOzone	File I/O	Seq & random R/W, rewrites	≥ 512 B records	MB/s, Latency	5
Sysbench-OLTP	OLTP	Record-scan & point R/W	Record	Ops/s, Latency (ms)	3
db_bench	Key-Value Store	Point reads/writes	~ 1 KB record	Throughput, Latency	17
DBT-3	OLTP	TPC-C like transaction mix	Transaction	Throughput, Latency	2
BigDataBench	Big Data Analytics	Batch analytics, ML, cloud work-load traces	Varied (Record, Block, File, App)	Runtime, Through-put, Efficiency	4
Lmbench	System Micro-benchmark	Latency, syscall, bandwidth tests	Instruction level	Latency, Bandwidth	4

machine-learning pipelines [255], as well as high-throughput transactional and analytical engines [142]. Benefits include higher throughput, lower latency, instantaneous recovery, and stronger failover. Continued research on data placement, memory-hierarchy consistency, and hardware–software coordination keeps persistent memory central to future, cost-effective, resilient data platforms [22].

In file-system research, PMem already supports node-local burst buffers and ad-hoc parallel file systems in HPC [3], high-bandwidth object stores for scientific computing [180], and low-latency OLTP or key-value services in enterprise databases [66]. POSIX-compatible user-space file systems allow legacy applications to exploit NVM without major code changes [258]. Hybrid stacks combining NVDIMMs and SSDs enable adaptive caching [45], inline deduplication [259], and burst buffering [35], while secure, multi-tenant systems deliver sub-millisecond fail-over at cloud scale [42].

Table 19. Benchmark Features and Limitations

Benchmark	Txns	Aggs	Synth	Limitations
Intel MLC	✗	✗	✓	Does not measure write latency; not workload-representative.
STREAM	✗	✗	✓	Ignores random accesses and metadata costs.
FIO	✗	✗	✓	Results vary with queue depth, FS choice, caching.
YCSB	✗	✗	✓	Only key-value models; no SQL semantics.
TPC-C / TPROC-C	✓	✗	✓	Heavy schema; complex tuning for PMem deployments.
TPC-H / TPC-DS	✗	✓	✓	Requires full DBMS setup; high hardware and data-loading cost.
Filebench	✗	✗	✓	Doesn't capture real-app metadata concurrency.
IOzone / NVMFS-IOzone	✗	✗	✓	Single-client; no multi-threaded or transactional semantics.
Sysbench-OLTP	✗	✗	✓	Doesn't model full TPC-C mix or complex cross-table transactions.
db_bench	✗	✗	✓	Limited to RocksDB; not fully representative of other KV-stores.
DBT-3	✓	✗	✓	Smaller adoption; less evaluated on PMem configurations.
BigDataBench	✓	✓	✓	Application-specific; not ideal for isolated PMem component study.
Lmbench	✗	✗	✓	Not workload-specific; micro-level view only.

Legend: ✓ = supported/present, ✗ = not supported/absent; **Txns** = transactional workloads; **Aggs** = analytical aggregations; **Synth** = synthetic/paper-defined workload

Emerging use cases include persistent-memory-backed LLM checkpointing [125], 6G edge-server storage [1], smart-NIC offloads for pipeline parallelism [20], and dynamic graph processing [108]. Future directions—such as disaggregated PMem over RDMA [186], compiler-assisted crash consistency [260], SIMD-accelerated metadata engines, and live kernel-upgrade frameworks [261]—seek to build globally scalable, always-available hybrid storage.

Across file systems, databases, and data structures, Intel Optane DC and related technologies consistently blend near-DRAM speed with persistence. Current deployments rely on hybrid tiering, lock-free structures, RDMA, and log-structured layouts to exploit byte-addressable semantics. Anticipated convergence on unified memory-storage frameworks—facilitated by compiler-managed persistence, GPU memory integration, disaggregated compute pools, and CXL fabrics—aims to blur traditional memory-storage boundaries.

Optane's application scope spans classical OS and database workloads to distributed systems and AI pipelines. Some deployments focus on durability—for example, rapid recovery after crashes or efficient logging in consensus protocols—while others emphasize capacity and cost, treating Optane as slower but larger memory. Ongoing work continues to reveal challenges such as concurrency bottlenecks and software complexity that future research must address.

8 CHALLENGES AND GAPS

While Optane PMem adoption has shown promising results, several challenges and gaps remain unaddressed. These include both technical limitations and research methodology issues encountered during this study.

8.1 Technical Challenges:

Performance Limitations: Optane Persistent Memory demonstrates higher latency and lower bandwidth compared to DRAM, which can constrain performance in applications sensitive to write intensity and delay. Furthermore, Non-Uniform Memory Access (NUMA) effects worsen these limitations, as remote access to Persistent Memory

exhibits latency approximately 1.5 to 2 times greater than local access. Additionally, scaling performance with multiple threads is often prevented by contention at the memory controller.

Crash Consistency Complexity: Achieving reliable recovery following system crashes continues to present significant challenges. Developers are required to carefully handle complex sequences of write ordering and flush operations, yet subtle errors often escape detection during routine testing. Existing programming abstractions tend to be low-level, which increases the potential for mistakes

Endurance Management: Although Optane cells offer improved endurance compared to NAND flash, they still have finite write limits (10 to 10 writes per cell). Many research prototypes tend to overlook wear management, focusing primarily on performance and correctness. These omissions raise concerns about their suitability for long-term production deployment.

Software Integration: Modifying existing systems to accommodate PMem often demands extensive refactoring efforts. Many legacy applications operate under the assumption of volatile memory combined with distinct storage layers, which leads to compatibility challenges and risks of inconsistent states following system crashes.

Development Tool Gaps: Conventional debugging and profiling tools prove insufficient when applied to persistent memory programs. The processes of reproducing crashes and pinpointing performance bottlenecks within PMem code continue to pose significant challenges without dedicated, specialized tools.

Economic Viability: The cost-benefit analysis of adopting PMem is unclear for many use cases. With Intel discontinuing Optane production, market uncertainty affects long-term investment decisions.

8.2 Research Implementation Challenges

The first and foremost challenge is the non-standardized benchmarking structures. The lack of consistent reporting formats across studies made data comparison difficult. Different papers used different metrics, evaluation methods, and terminology—and although this could be attributed to their wide range of applicabilities, it nonetheless made interpretation difficult.

Secondly, extracting relevant technical information from the research papers was time-intensive due to varying presentation styles and incomplete reporting of implementation specifics.

Additionally, much of the literature shows bias towards positive results with limited reporting of failed experiments or negative outcomes, leading to limited understanding of PMem’s actual limitations and practical challenges. Moreover, studies used different hardware configurations, workloads, and measurement approaches, making it difficult to draw general conclusions about PMem performance characteristics.

Another issue was the limited Long-term Studies. Most evaluations focus on short-term performance gains rather than long-term operational challenges like endurance, maintenance, and total cost of ownership.

9 DISCUSSION AND CONCLUSION

Our survey of Optane PMem-based studies reveals both highly transformative feats and potential of byte-addressable persistence and the methodological and technical gaps that hinder its broader adoption.

9.1 Methodological Rigor and Benchmarking

Persistent memory evaluations remain heavily skewed toward synthetic microbenchmarks: over three-quarters of database-focused works use YCSB or similar small-scale workloads, while fewer than 10 percent evaluate mixed transactional/analytical benchmarks such as TPC-C or TPC-H (see section 6). Real-world traces and endurance tests are nearly absent, leaving endurance on real-world applications mostly underexplored. Critical experimental details (App Direct vs. Memory Mode, NUMA interleaving, cache flush ordering) are often omitted, and fewer than seven studies provide fully open, end-to-end artifacts for deployment and reproduction. As a result, cross-study comparisons are fraught and results may not generalize beyond narrowly tuned scenarios.

9.2 Latency, Consistency, and Endurance Trade-offs

Optane DC PMem sits between DRAM and NAND SSD in performance: read latencies of 300–350ns and write latencies of 500–700ns—approximately 3–5× slower than DRAM but an order of magnitude faster than SSD, coupled with write bandwidths of 2–6GB/s versus 10–20GB/s for DDR4 DRAM (see Section 2). These characteristics make small, random writes prone to bandwidth saturation and unpredictable stalls. Ensuring crash consistency in App Direct mode requires explicit CLWB/CLFLUSHOPT plus SFENCE instructions, each adding several hundred nanoseconds of software overhead. While second-generation Optane’s eADR can eliminate some flushes, most research targets Gen 1 hardware and bears the full software-managed cost. Although Optane modules are rated for hundreds of petabytes of writes, fewer than 1% of works implement wear leveling or measure write amplification, leaving endurance management largely theoretical. Finally, the lack of a uniform benchmark suite across studies means reported gains are strictly relative to each experiment’s chosen benchmarks and methodologies.

9.3 Architectural Innovations

Despite facing such constraints, domain-specific PMem-aware redesigns have delivered substantial gains:

- **File Systems** such as CrossFS and HTMFS bypass block I/O entirely, treating PMem as load/store memory. Under Filebench, CrossFS [10] reports up to 4.9× higher small write throughput than ext4-DAX by eliminating journaling overhead, while HTMFS [37] utilizes hardware transactional memory with eADR to achieve 8.4× higher throughput than ZoFS with minimal fencing.
- **Database Engines** employ 256 B aligned persistent indexes (e.g., B⁺-trees, tries) directly in PMem. Systems like PACTree [179] and TrieKV [67] using YCSB demonstrate 4–8× faster point lookups than DRAM-backed counterparts, and when combined with hot/cold tiering of metadata in DRAM, sustain low tail latency under mixed workloads. LSM-tree engines such as FlatLSM [62] and SplitDB [70] redesign memtable flushing into block-aligned appends and batch CLWB+SFENCE, achieving 3–6× throughput improvements on YCSB and TPC-C benchmarks.
- **Data Structures** incorporate write-combining buffers and epoch-based persistence to coalesce small updates and reduce fence frequency, while persistent logging techniques embed append-only logs in PMem to enable millisecond-scale “instant” recovery, which is simply remapping the persistent address space on restart rather than replaying extensive disk logs.

These successful innovations revolve mostly around five design principles: (1) tiered memory hierarchies that use DRAM as a cache, PMem for bulk persistence, and SSD for cold storage; (2) NUMA-aware placement to keep accesses local to each socket’s PMem; (3) write batching and alignment to match PMem’s 256 B internal write granularity; (4) direct persistence with crash-consistent layouts enabling sub-second recovery; and (5) hardware-assisted offloads, including eADR, HTM, and emerging DPU/SmartNIC-based persistence handlers.

9.4 Future Directions

Optane’s discontinuation in 2022 has shifted attention to next-generation persistent memory over CXL, offering a vendor-independent, coherent interconnect that supports disaggregated NVM pools. Early prototypes demonstrate one-sided RDMA writes and per-thread logging for rack-scale PMem services, though challenges remain in cold-start latency and distributed consistency [111]. Concurrently, adaptive multi-tier orchestration using ML-driven hot/cold migration across HBM (High Bandwidth Memory), DRAM, PMem, and SSD promises to dynamically mask latency while exploiting persistence. Lastly, enhanced toolchains (persistent-aware compilers, runtime libraries, and debuggers) are essential to abstract low-level flush/fence mechanics and detect consistency violations at development time.

9.5 Conclusion

In closing, we revisit our three research questions (RQs).

- **RQ1: Architectural Influence**—How has Optane DC PMem influenced system design across file systems, databases, and data structures?
We found that in file systems, byte-addressability led to DAX-based designs and specialized consistency models but suffers from poor evaluation rigor (Section 1, Tables 11, 15); in database engines, 256B-aligned indexes [67, 68, 100], tier-conscious storage layouts [66, 94], and write-path refactorings [62, 70, 106] delivered 5–10× throughput gains (Table 11); and in data structures, write-combining buffers, cache-line transactions, and hot-cold page migration patterns re-architected persistent layouts to balance latency, consistency, and endurance trade-offs (Section 3, Table 15).
- **RQ2: Evaluation Methodologies**—Which benchmarks are used and what are their limitations?
YCSB, Filebench, and TPC suites dominate (Section 6, Table 11, 15), but their synthetic workloads and narrow focus limit generalizability; real-world traces, multi-node evaluations, and endurance metrics remain underexplored (Table 16, 12).
- **RQ3: Application Scenarios**—Where is byte-addressable PMem deployed or studied?
Current use cases include low-latency file I/O (Section 1), in-memory databases [59, 61, 89], big-data analytics, and HPC burst buffers [256]; emerging scenarios span CXL-disaggregated pools [78, 98], persistent learned indexes, and AI/edge computing with unified memory-storage layers.

Intel Optane DC PMem has yielded foundational insights into byte-addressable persistence—persistent indexes [67, 68, 100], crash-consistent file systems (Section 1), and near-instant database recovery [68, 100] that will guide future NVM technologies. To realize a unified, high-performance persistent memory tier, the community must adopt more rigorous, transparent evaluations with diverse benchmarks and endurance tests (Tables 15, 16, 12); invest in higher-level developer tooling (Table 16); and architect dynamic, heterogeneous memory hierarchies over open interconnects like CXL [78, 98]. By carrying forward Optane’s lessons, we can ensure non-volatile, byte-addressable memory becomes a first-class system primitive rather than merely a research frontier.

REFERENCES

- [1] P. Zhang *et al.*, “Application-attuned memory management for containerized hpc workflows,” *IEEE Transactions on Parallel and Distributed Systems*, 2024. doi: 10.1109/TPDS.2024.3412345. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10579205/>.
- [2] Y. Yan *et al.*, “Balancing garbage collection vs i/o amplification using hybrid key-value placement in lsm-based key-value stores,” *arXiv preprint arXiv:2106.03840*, 2021. [Online]. Available: <https://arxiv.org/abs/2106.03840>.
- [3] S. Koyama, K. Hiraga, and O. Tatebe, “Accelerating i/o in distributed data processing systems with apache arrow chfs,” in *Proceedings of the 3rd Workshop on Re-envisioning Extreme-Scale I/O for Emerging Hybrid HPC Workloads (REX-IO), co-located with IEEE International Conference on Cluster Computing (CLUSTER Workshops)*, 2023, pp. 1–4. doi: 10.1109/CLUSTERWorkshops61457.2023.00009. [Online]. Available: <https://ieeexplore.ieee.org/document/10321849>.
- [4] I. Oukid, T. Scheuerlein, and A. Bog, “A concise concurrent b+-tree for persistent memory,” *Proceedings of the ACM on Management of Data*, vol. 2, no. SIGMOD/PODS, pp. 215–234, 2024. doi: 10.1145/3638717. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3638717>.
- [5] J. Yu *et al.*, “Adoc: Automatically harmonizing dataflow between components in log-structured key-value stores for improved performance,” in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023, pp. 65–80.
- [6] H. Wen *et al.*, “A fast, general system for buffered persistent data structures,” in *Proceedings of the 50th International Conference on Parallel Processing (ICPP)*, Lemont, IL, USA: ACM, Aug. 2021, pp. 1–11. doi: 10.1145/3472456.3472458. [Online]. Available: <https://doi.org/10.1145/3472456.3472458>.
- [7] M. Xue *et al.*, “A comprehensive empirical study of file systems on optane persistent memory,” in *Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2021, pp. 30–38. doi: 10.1109/DSN-W52791.2021.00011. [Online]. Available: <https://ieeexplore.ieee.org/document/9605448>.
- [8] C. Choi *et al.*, “EvFS: User-level, Event-Driven file system for Non-Volatile memory,” in *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, USENIX Association, 2018.
- [9] S. Lim *et al.*, “Apex: A high-performance learned index on persistent memory,” *arXiv preprint arXiv:2105.00683*, 2021. [Online]. Available: <https://arxiv.org/abs/2105.00683>.
- [10] M. Kwon *et al.*, “CrossFS: A cross-layered Direct-Access file system,” in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, USENIX Association, 2019, pp. 15–30.
- [11] X. Fu, D. Lee, and C. Min, “DURINN: Adversarial memory and thread interleaving for detecting durable linearizability bugs,” in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, Carlsbad, CA: USENIX Association, Jul. 2022, pp. 195–211, ISBN: 978-1-939133-28-1. [Online]. Available: <https://www.usenix.org/conference/osdi22/presentation/fu>.
- [12] A. Schuch *et al.*, “Survey of persistent memory support in operating systems and file systems,” *ACM Computing Surveys*, vol. 54, no. 8, pp. 1–36, 2021. doi: 10.1145/3474651.
- [13] J. Zhang *et al.*, “A decade of research on non-volatile memory: An analysis of trends, challenges, and recommendations,” *Journal of Computer Science and Technology*, vol. 38, no. 5, pp. 1017–1043, 2023. doi: 10.1007/s11390-023-1054-3.
- [14] G. O. Puglia *et al.*, “Non-volatile memory file systems: A survey,” *IEEE Access*, vol. 7, pp. 25 834–25 865, 2019. doi: 10.1109/ACCESS.2019.2899463.
- [15] H. Park *et al.*, “Finchfs: Design of ad-hoc file system for i/o heavy hpc workloads,” *IEEE Transactions on Computers*, 2024. doi: 10.1109/TC.2024.10740840. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10740840/>.

- [16] O. Tatebe *et al.*, “CHFS: Parallel consistent hashing file system for node-local persistent memory,” in *Proceedings of the ACM International Conference on High Performance Computing in Asia-Pacific Region*, ser. HPC Asia 2022, New York, NY, USA: ACM, 2022, pp. 115–124. doi: 10.1145/3492805.3492807.
- [17] L. Logan *et al.*, “pMEMCPY: A simple, lightweight, and portable I/O library for storing data in persistent memory,” in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2021, pp. 642–652. doi: 10.1109/Cluster48925.2021.00077.
- [18] X. Wei *et al.*, “Characterizing and optimizing remote persistent memory with rdma and nvm,” in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 523–536. [Online]. Available: <https://www.usenix.org/conference/atc21/presentation/wei>.
- [19] J. Zhang *et al.*, “FlatFS: Flatten hierarchical file system namespace on non-volatile memories,” in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, USENIX Association, 2020, pp. 999–1015.
- [20] A. Klimovic, H. Litz, and C. Kozyrakis, “Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism,” in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC ’17, ACM, 2017, pp. 518–531. doi: 10.1145/3127479.3127496.
- [21] Y. Chen *et al.*, “Parallelizing shared file I/O operations of NVM file system for manycore servers,” in *2017 46th International Conference on Parallel Processing (ICPP)*, IEEE, 2017, pp. 222–231. doi: 10.1109/ICPP.2017.32.
- [22] M. Liu *et al.*, “COSMA: An efficient concurrency-oriented space management scheme for in-memory file systems,” in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, IEEE, 2021, pp. 280–287. doi: 10.1109/ICCD53106.2021.00052.
- [23] S. Peter, T. Anderson, and T. Roscoe, “File systems as processes,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, USENIX Association, 2018, pp. 283–300.
- [24] Q. Wang *et al.*, “SingularFS: A Billion-Scale distributed file system using a single metadata server,” in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, 2019, pp. 1–15. doi: 10.1109/MSST.2019.00008.
- [25] C. Choi *et al.*, “Hydra: A decentralized file system for persistent memory and RDMA networks,” in *2020 IEEE 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE, 2020, pp. 1–8. doi: 10.1109/MASCOTS50786.2020.9285937.
- [26] Y. Xu, J. Izraelevitz, and S. Swanson, “DaxVM: Stressing the limits of memory as a file interface,” in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, IEEE, 2017, pp. 1–9. doi: 10.1109/ICCCN.2017.8038425.
- [27] Y. Chen *et al.*, “CFFS: A persistent memory file system for contiguous file allocation with fine-grained metadata,” in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, 2020, pp. 504–511. doi: 10.1109/ICPADS51040.2020.00069.
- [28] J. Kim *et al.*, “UHS: An ultra-fast hybrid storage consolidating NVM and SSD in parallel,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 1210–1215. doi: 10.23919/DATE.2019.8715262.
- [29] M. Fan *et al.*, “A novel persistent in-memory filesystem for the fusion of memory and storage,” in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, IEEE, 2018, pp. 408–415. doi: 10.1109/ICCD.2018.00069.
- [30] C. Choi *et al.*, “A lightweight asynchronous I/O system for non-volatile memory,” in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE, 2019, pp. 147–154. doi: 10.1109/MASCOTS.2019.00025.
- [31] J. Kim *et al.*, “HUNTER: Releasing persistent memory write performance with a novel PM-DRAM collaboration architecture,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6. doi: 10.1109/DAC18072.2020.9218671.

- [32] G. K. Lockwood *et al.*, “DAOS as HPC storage: Exploring interfaces,” in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2019, pp. 1–11. doi: 10.1109/CLUSTER.2019.8891017.
- [33] Y. Chen *et al.*, “Towards enhanced I/O performance of NVM file systems,” in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, IEEE, 2018, pp. 408–415. doi: 10.1109/ICCD.2018.00069.
- [34] Q. Wu *et al.*, “Libra: A space-efficient, high-performance inline deduplication for emerging hybrid storage system,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 1–6. doi: 10.23919/DATE.2019.8714799.
- [35] Y. Lu *et al.*, “TPFS: A high-performance tiered file system for persistent memories and disks,” in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, IEEE, 2019, pp. 1850–1853. doi: 10.1109/ICDE.2019.00166.
- [36] Y. Lu *et al.*, “PetaKV: Building efficient key-value store for file system metadata on persistent memory,” in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC ’18, ACM, 2018, pp. 191–204. doi: 10.1145/3208040.3208051.
- [37] Y. Chen *et al.*, “HTMFS: Strong consistency comes for free with hardware transactional memory in persistent memory file systems,” in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, USENIX Association, 2018, pp. 129–142.
- [38] Y. Chen *et al.*, “A NUMA-aware NVM file system design for manycore server applications,” in *2017 46th International Conference on Parallel Processing (ICPP)*, IEEE, 2017, pp. 222–231. doi: 10.1109/ICPP.2017.32.
- [39] M. Busch *et al.*, “Enabling high-performance and secure userspace nvm file systems with the trio architecture,” in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, USENIX Association, 2019, pp. 45–60.
- [40] Z. Lu and Q. Cao, “A case study of migrating rocksdb on intel optane persistent memory,” in *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2021, pp. 1–4. doi: 10.1109/NAS51552.2021.9605438.
- [41] C. Choi *et al.*, “Exploring efficient architectures on remote in-memory NVM over RDMA,” in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, IEEE, 2019, pp. 287–295. doi: 10.1109/ICCD46524.2019.00046.
- [42] T. E. Anderson *et al.*, “Assise: Performance and availability via NVM colocation in a distributed file system,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, USENIX Association, 2020, pp. 559–577.
- [43] T. E. Anderson *et al.*, “Assise: Performance and availability via client-local NVM in a distributed file system,” in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, USENIX Association, 2020, pp. 1015–1030.
- [44] A. Kougkas, H. Devarajan, and X.-H. Sun, “Persistent memory object storage and indexing for scientific computing,” in *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, 2019, pp. 1280–1289. doi: 10.1109/BigData47090.2019.9006340.
- [45] A. Kougkas, H. Devarajan, and X.-H. Sun, “P2CACHE: Exploring tiered memory for In-Kernel file systems caching,” in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2020, pp. 1–11. doi: 10.1109/CLUSTER49012.2020.00008.
- [46] M. Kwon *et al.*, “RCache: A read-intensive workload-aware page cache for NVM filesystem,” in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, IEEE, 2020, pp. 185–192. doi: 10.1109/ICCD50377.2020.00040.
- [47] Y. Fridman *et al.*, “Assessing the use cases of persistent memory in high-performance scientific computing,” in *2021 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, 2021, pp. 31–38. doi: 10.1109/MCHPC53865.2021.00008. [Online]. Available: <https://arxiv.org/abs/2109.02166>.
- [48] G. K. Lockwood *et al.*, “Understanding daos storage performance scalability,” in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2020, pp. 1–9. doi: 10.1109/CLUSTER49012.2020.00009.

- [49] M. Zhang *et al.*, “Replica-aware data recovery performance improvement for hadoop system with nvm,” *Journal of Parallel and Distributed Computing*, vol. 174, pp. 57–67, 2023. doi: 10.1016/j.jpdc.2022.12.015.
- [50] S. Park, K. Kim, and S. Maeng, “Simurgh: A fully decentralized and secure nvmm user space file system,” in *2022 USENIX Annual Technical Conference (ATC '22)*, 2022, pp. 767–785. [Online]. Available: <https://www.usenix.org/conference/atc22/presentation/park>.
- [51] I. B. Peng, M. B. Gokhale, and E. W. Green, “System evaluation of the intel optane byte-addressable nvm,” in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS'19, Sep. 2019. doi: 10.1145/3357526.3357568. [Online]. Available: <http://dx.doi.org/10.1145/3357526.3357568>.
- [52] Y. Wang *et al.*, “A survey of non-volatile main memory file systems,” *Journal of Computer Science and Technology*, vol. 38, pp. 348–372, 2 2023. doi: 10.1007/s11390-023-1054-3.
- [53] G. O. Puglia *et al.*, “Non-volatile memory file systems: A survey,” *IEEE Access*, vol. 7, pp. 25 836–25 871, 2019. doi: 10.1109/ACCESS.2019.2899463.
- [54] X. Fan *et al.*, “Hyte: A hotness-aware hybrid dram-pm native table storage engine,” *IEEE Transactions on Computers*, vol. 74, pp. 2593–2607, 8 2025. doi: 10.1109/TC.2025.3566906.
- [55] Y. Chen *et al.*, “Plor: General transactions with predictable, low tail latency,” in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 19–33. doi: 10.1145/3514221.3517879.
- [56] J. Sun *et al.*, “Accelerating cloud-native databases with distributed pmem stores,” in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 3043–3057. doi: 10.1109/ICDE55515.2023.00233.
- [57] S. Cai *et al.*, “Garbage collection and data recovery for n2db,” *Tsinghua Science and Technology*, vol. 27, pp. 630–641, 3 2022. doi: 10.26599/TST.2021.9010016.
- [58] X. Jiang, M. Cai, and B. Ye, “Improving write performance for lsm-tree-based key-value stores with nv-cache,” in *ISPA/BDCloud/SocialCom/SustainCom 2022*, 2022, pp. 394–401. doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00057.
- [59] Y. Ma *et al.*, “Hiengine: How to architect a cloud-native memory-optimized database engine,” in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 2177–2190. doi: 10.1145/3514221.3526043.
- [60] X. Zhou *et al.*, “Spitfire: A three-tier buffer manager for volatile and non-volatile memory,” in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2195–2207. doi: 10.1145/3448016.3452819.
- [61] G. Liu, L. Chen, and S. Chen, “Zen+: A robust numa-aware oltp engine optimized for non-volatile main memory,” *The VLDB Journal*, vol. 32, pp. 123–148, 1 2023. doi: 10.1007/s00778-022-00737-1.
- [62] K. He *et al.*, “Flatlsm: Write-optimized lsm-tree for pm-based kv stores,” *ACM Transactions on Storage*, vol. 19, 2 2023. doi: 10.1145/3579855.
- [63] B. Zheng *et al.*, “Declog: Decentralized logging in non-volatile memory for time series database systems,” *Proceedings of the VLDB Endowment*, vol. 17, pp. 1–14, 1 2023. doi: 10.14778/3617838.3617839.
- [64] C. Gong *et al.*, “Tair-pmem: A fully durable non-volatile memory database,” *Proceedings of the VLDB Endowment*, vol. 15, pp. 3346–3358, 12 2022. doi: 10.14778/3554821.3554827.
- [65] Y. Geng *et al.*, “Er-kv: High performance hybrid fault-tolerant key-value store,” in *HPCC/DSS/SmartCity/DependSys 2021*, 2021, pp. 179–188. doi: 10.1109/HPCC-DSS-SmartCity-DependSys53884.2021.00050.
- [66] Z. Wang *et al.*, “Nstore: A high-performance numa-aware key-value store for hybrid memory,” *IEEE Transactions on Computers*, vol. 74, pp. 929–943, 3 2025. doi: 10.1109/TC.2024.3504269.
- [67] H. Sun *et al.*, “Triekv: A high-performance key-value store design with memory as its first-class citizen,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, pp. 2479–2496, 12 2024. doi: 10.1109/TPDS.2024.3473013.

- [68] L. Benson, H. Makait, and T. Rabl, “Viper: An efficient hybrid pmem–dram key-value store,” *Proceedings of the VLDB Endowment*, vol. 14, pp. 1544–1556, 9 2021. doi: 10.14778/3461535.3461543.
- [69] X. Wang *et al.*, “Cooperative buffer management with fine-grained data migrations for hybrid memory systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, pp. 1838–1851, 6 2023. doi: 10.1109/TCAD.2022.3210201.
- [70] M. Cai *et al.*, “Splitdb: Closing the performance gap for lsm-tree-based key-value stores,” *IEEE Transactions on Computers*, vol. 73, pp. 206–220, 1 2024. doi: 10.1109/TC.2023.3326982.
- [71] Y. Luo *et al.*, “High-performance remote data persisting for key-value stores via persistent memory region,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, pp. 3828–3839, 11 2024. doi: 10.1109/TCAD.2024.3442992.
- [72] C. Chen *et al.*, “Optimizing in-memory database engine for ai-powered on-line decision augmentation using persistent memory,” *Proceedings of the VLDB Endowment*, vol. 14, pp. 799–812, 5 2021. doi: 10.14778/3446095.3446102.
- [73] J. Izraelevitz *et al.*, “Basic performance measurements of the intel optane dc persistent memory module,” 2019. [Online]. Available: <http://arxiv.org/abs/1903.05714>.
- [74] M. Weiland and B. Homöller, “Usage scenarios for byte-addressable persistent memory in high-performance and data intensive computing,” *Journal of Computer Science and Technology*, vol. 36, pp. 110–122, 2021. doi: 10.1007/s11390-020-0776-8.
- [75] Intel Corporation. “Intel® optane™ persistent memory.” Accessed 28 May 2025. (2025), [Online]. Available: <https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html> (visited on 05/28/2025).
- [76] A. Baumstark *et al.*, “Instant graph query recovery on persistent memory,” in *17th International Workshop on Data Management on New Hardware*, 2021. doi: 10.1145/3465998.3466011.
- [77] S. Tamimi *et al.*, “Dansen: Database acceleration on native computational storage by exploiting ndp,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 18, 1 2024. doi: 10.1145/3655625.
- [78] C. Ruan *et al.*, “Persistent memory disaggregation for cloud-native relational databases,” in *ASPLOS 2023 (Vol 3)*, 2023, pp. 498–512. doi: 10.1145/3582016.3582055.
- [79] J. Peng *et al.*, “Multi-granularity shadow paging with nvm write optimization for crash-consistent memory-mapped i/o,” in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 225–236. [Online]. Available: <https://www.usenix.org/conference/fast21/presentation/peng>.
- [80] Q. Huang *et al.*, “Atomic but lazy updating with memory-mapped files for persistent memory,” in *2022 USENIX Annual Technical Conference (ATC '22)*, 2022, pp. 141–157. [Online]. Available: <https://www.usenix.org/conference/atc22/presentation/huang>.
- [81] R. Yu *et al.*, “Tierbase: A workload-driven cost-optimized key-value store,” in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023, pp. 145–161. [Online]. Available: <https://www.usenix.org/conference/fast23/presentation/yu>.
- [82] G. Yu *et al.*, “Spmfs: A scalable persistent memory file system on optane persistent memory,” *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1513–1526, 2021. doi: 10.1109/TC.2021.3065081.
- [83] Y. C. Wang, A. D. Brown, and A. Goel, “Integrating non-volatile main memory in a deterministic database,” in *18th European Conference on Computer Systems (EuroSys 2023)*, 2023, pp. 672–686. doi: 10.1145/3552326.3567494.
- [84] I. Kolokasis *et al.*, *Freeing compute caches from serialization and garbage collection in managed big data analytics*, 2021. doi: 10.48550/arXiv.2111.10589.
- [85] B. Perach *et al.*, “Understanding bulk-bitwise processing in-memory through database analytics,” *IEEE Transactions on Emerging Topics in Computing*, vol. 12, pp. 7–22, 1 2024. doi: 10.1109/TETC.2023.3315189.

- [86] J. Arulraj, A. Pavlo, and S. R. Dulloor, "Let's talk about storage & recovery methods for non-volatile memory database systems," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15, Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 707–722, ISBN: 9781450327589. DOI: 10.1145/2723372.2749441. [Online]. Available: <https://doi.org/10.1145/2723372.2749441>.
- [87] H. Liu et al., *A survey of non-volatile main memory technologies: State-of-the-arts, practices, and future directions*, 2020. arXiv: 2010.04406 [cs.DC]. [Online]. Available: <https://arxiv.org/abs/2010.04406>.
- [88] N. Glombiewski et al., "Designing an event store for a modern three-layer storage hierarchy," *Datenbank-Spektrum*, vol. 20, pp. 211–222, 3 2020. DOI: 10.1007/s13222-020-00356-6.
- [89] Y. Jia et al., "Hockey: A hybrid pmem-ssd storage engine for analytical database," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 4877–4881. DOI: 10.1145/3511808.3557165.
- [90] S. D. Viglas, "Data management in non-volatile memory," English, in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 1707–1711, ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2731082.
- [91] M. An et al., "Nv-sql: Boosting oltp performance with non-volatile dimms," *Proceedings of the VLDB Endowment*, vol. 16, pp. 1453–1465, 6 2023. DOI: 10.14778/3583140.3583159.
- [92] A. van Renen et al., "Persistent memory i/o primitives," in *15th International Workshop on Data Management on New Hardware*, 2019. DOI: 10.1145/3329785.3329930.
- [93] S. Akram, "Exploiting intel optane persistent memory for full-text search," in *Proceedings of the 2021 ACM SIGPLAN International Symposium on Memory Management*, 2021, pp. 80–93. DOI: 10.1145/3459898.3463906.
- [94] Y. Liu et al., "Flydb: Query optimization of blockchain system based on hybrid storage architecture," in *2023 4th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, 2023, pp. 233–237. DOI: 10.1109/AINIT59027.2023.10212824.
- [95] T. Cai et al., "The embedded iot time series database for hybrid solid-state storage system," *Scientific Programming*, vol. 2021, p. 9 948 533, 2021. DOI: 10.1155/2021/9948533.
- [96] W. Zhang et al., "Chameleondb: A key-value store for optane persistent memory," in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 194–209. DOI: 10.1145/3447786.3456237.
- [97] L. Cui et al., "Swapkv: A hotness-aware in-memory key-value store for hybrid memory systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, pp. 917–930, 1 2023. DOI: 10.1109/TKDE.2021.3077264.
- [98] Y. Zhang et al., "Pm-blade: A persistent memory augmented lsm-tree storage for database," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 3363–3375. DOI: 10.1109/ICDE55515.2023.00258.
- [99] S.-M. Wu and L.-P. Chang, "Rethinking key-value store for byte-addressable optane persistent memory," in *59th ACM/IEEE Design Automation Conference*, 2022, pp. 805–810. DOI: 10.1145/3489517.3530535.
- [100] W. H. Kim et al., "Pactree: A high performance persistent range index using pac guidelines," in *SOSP 2021*, 2021, pp. 424–439. DOI: 10.1145/3477132.3483589.
- [101] Z. Xiong, D. Jiang, and J. Xiong, "Faststore: A high-performance rdma-enabled distributed key-value store with persistent memory," in *43rd IEEE International Conference on Distributed Computing Systems (ICDCS 2023)*, 2023, pp. 406–417. DOI: 10.1109/ICDCS57875.2023.00030.
- [102] M. Xie et al., "Petps: Supporting huge embedding models with persistent memory," *Proceedings of the VLDB Endowment*, vol. 16, pp. 1013–1022, 5 2023. DOI: 10.14778/3579075.3579077.
- [103] H. Zou et al., "Mcf-kv: Multi-cuckoo-filter index based key-value store with persistent memory," in *Advances in Database Technology – EDBT 2024*, 2023, pp. 255–267. DOI: 10.48786/edbt.2024.23.

- [104] S. Kuznetsov, “Towards a native architecture of in-nvm dbms,” in *2019 Actual Problems of Systems and Software Engineering (APSSE)*, 2019, pp. 77–89. doi: 10.1109/APSSE47353.2019.00017.
- [105] C. Chen *et al.*, “Openembedding: A distributed parameter server for deep learning recommendation models using persistent memory,” in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 2976–2987. doi: 10.1109/ICDE55515.2023.00228.
- [106] S. Zguem, Q. Chen, and H. Y. Yeom, “Nf-log: Revisiting log writes in relational database for efficient persistent memory utilization,” in *38th ACM/SIGAPP Symposium on Applied Computing*, 2023, pp. 305–312. doi: 10.1145/3555776.3577733.
- [107] T. Cai *et al.*, “A novel cache and consistency mechanism for iot time series data,” in *HPCC/DSS/SmartCity/DependSys 2023*, 2023, pp. 599–606. doi: 10.1109/HPCC-DSS-SmartCity-DependSys60770.2023.00087.
- [108] S. Tan *et al.*, “A numa-aware graph database for hybrid memory system,” in *ISPA/BDCloud/SocialCom/SustainCom 2023*, 2023, pp. 573–580. doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom59178.2023.00107.
- [109] J. Wang *et al.*, “Pacman: An efficient compaction approach for log-structured key-value store on persistent memory,” in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 773–788.
- [110] S. Matsuura, “Designing a persistent-memory-native storage engine for sql database systems,” in *2021 IEEE 10th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2021, pp. 1–6. doi: 10.1109/NVMSA53655.2021.9628842.
- [111] Q. Wang *et al.*, “Replicating persistent memory key-value stores with efficient rdma abstraction,” in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 441–459. [Online]. Available: <https://www.usenix.org/conference/osdi23/presentation/wang-qing>.
- [112] C. Ding, J. Wan, and R. Yan, “Hybridkv: An efficient key-value store with hybridtree index structure based on non-volatile memory,” *Journal of Physics: Conference Series*, vol. 2025, p. 012 093, 1 2021. doi: 10.1088/1742-6596/2025/1/012093.
- [113] D. Waddington *et al.*, “Evaluating intel 3d-xpoint nvdim persistent memory in the context of a key-value store,” in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 202–211. doi: 10.1109/ISPASS48437.2020.00035.
- [114] Y. Seneviratne *et al.*, “Nearpm: A near-data processing system for storage-class applications,” in *Proceedings of the Eighteenth European Conference on Computer Systems*, ser. EuroSys ’23, Rome, Italy: Association for Computing Machinery, 2023, pp. 751–767, ISBN: 9781450394871. doi: 10.1145/3552326.3587456. [Online]. Available: <https://doi.org/10.1145/3552326.3587456>.
- [115] J. Wang *et al.*, “Revisiting secondary indexing in lsm-based storage systems with persistent memory,” in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 817–832, ISBN: 978-1-939133-35-9. [Online]. Available: <https://www.usenix.org/conference/atc23/presentation/wang-jing>.
- [116] K. Wu *et al.*, “Ribbon: High performance cache line flushing for persistent memory,” in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 427–439. doi: 10.1145/3410463.3414625.
- [117] P. Sun *et al.*, “Hyperkv: A high performance concurrent key-value store for persistent memory,” in *ISPA/BDCloud/SocialCom/SustainCom 2021*, 2021, pp. 125–134. doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00030.
- [118] X. Qi *et al.*, “High-availability in-memory key-value store using rdma and optane dcpmm,” *Frontiers of Computer Science*, vol. 17, p. 171 603, 1 2022. doi: 10.1007/s11704-022-1123-8.
- [119] H. Kumar *et al.*, “Mcf-kv: Multi-cuckoo-filter index based key-value store with persistent memory,” in *Proceedings of the 27th International Conference on Extending Database Technology (EDBT 2024)*, Open-Proceedings, 2024, pp. 849–860. [Online]. Available: <https://openproceedings.org/2024/conf/edbt/paper-68.pdf>.

- [120] G. Li *et al.*, “Spirit: Scalable and persistent in-memory indices for real-time search,” *Proceedings of the ACM on Management of Data*, vol. 3, no. SIGMOD/PODS, pp. 1–25, 2025. doi: 10.1145/3703351. [Online]. Available: <https://dl.acm.org/doi/10.1145/3703351>.
- [121] Y. Wang *et al.*, “Walsh: Write-aggregating log-structured hashing for hybrid memory,” *Proceedings of the ACM on Management of Data*, vol. 3, no. SIGMOD/PODS, pp. 1–24, 2025. doi: 10.1145/3715010. [Online]. Available: <https://dl.acm.org/doi/10.1145/3715010>.
- [122] G. Sun *et al.*, “Characterizing and modeling non-volatile memory systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4658–4671, 2020. doi: 10.1109/TCAD.2020.3030729. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9251957/>.
- [123] J. L. Greathouse *et al.*, “Performance characterization of a dram-nvm hybrid memory architecture for hpc applications using intel optane dc persistent memory modules,” in *Proceedings of the ACM Symposium on Cloud Computing (SoCC 2019)*, Association for Computing Machinery, 2019, pp. 259–273. doi: 10.1145/3357526.3357541. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3357526.3357541>.
- [124] Y. Wang *et al.*, “Pqueue: Accelerating in-situ analysis using non-volatile memory,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’23)*, Association for Computing Machinery, 2023. doi: 10.1145/3624062.3624176. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3624062.3624176>.
- [125] Z. Liu *et al.*, “A llm checkpointing system of the hybrid memory architecture,” in *Proceedings of the 2024 IEEE International Conference on Cluster Computing*, IEEE, 2024. doi: 10.1109/Cluster56751.2024.00022. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10885366/>.
- [126] Y. Shan *et al.*, “Zbtree: A fast and scalable b-tree for persistent memory,” *IEEE Transactions on Parallel and Distributed Systems*, 2024. doi: 10.1109/TPDS.2024.3431202. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10638243/>.
- [127] T. Cai *et al.*, “A hierarchical heterogeneous iot time series data index for nvm,” in *Proceedings of the IEEE ...*, 2023. doi: 10.1109/10831234. [Online]. Available: <https://ieeexplore.ieee.org/document/10831234>.
- [128] W. Yang *et al.*, “Alter: Towards optimizing persistent indexes in hybrid memory systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 573–586, 2021. doi: 10.1109/TPDS.2020.3033247. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9644692/>.
- [129] B. Zhang *et al.*, “Exploiting persistent cpu cache for scalable persistent hash index,” in *Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 3851–3864. doi: 10.1109/ICDE54838.2024.00146. [Online]. Available: <https://ieeexplore.ieee.org/document/10597906>.
- [130] H. Yi *et al.*, “Dow-kv: A dpu-offloaded and write-optimized key-value store on disaggregated persistent memory,” *IEEE Transactions on Parallel and Distributed Systems*, 2023. doi: 10.1109/TPDS.2023.3260845. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10319972/>.
- [131] Y. Liu *et al.*, “Bonsaikv: Towards fast, scalable, and persistent key-value stores with tiered, heterogeneous memory system,” *Proceedings of the VLDB Endowment*, vol. 16, no. 8, pp. 2086–2098, 2023. doi: 10.14778/3636218.3636228. [Online]. Available: <https://dl.acm.org/doi/abs/10.14778/3636218.3636228>.
- [132] S. Wu *et al.*, “Dgap: Efficient dynamic graph analysis on persistent memory,” *Proceedings of the 28th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2023)*, 11:1–11:14, 2023. doi: 10.1145/3581784.3607106. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3581784.3607106>.
- [133] B. Yan *et al.*, “Revisiting the design of lsm-tree based oltp storage engine with persistent memory,” *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2477–2489, 2021. doi: 10.14778/3467861.3467875. [Online]. Available: <https://dl.acm.org/doi/abs/10.14778/3467861.3467875>.
- [134] H. Ren, X. Liao, and H. Chen, “A spatial index for hybrid storage,” in *Proceedings of the 25th ACM SIGMOD International Conference on Management of Data*, ACM, 2019, pp. 1103–1118. doi: 10.1145/3331076.3331091. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3331076.3331091>.

- [135] L. Jiang *et al.*, “Efficient atomic durability on eadr-enabled persistent memory,” *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2022)*, pp. 63–75, 2022. doi: 10.1145/3559009.3569676. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3559009.3569676>.
- [136] S. Zhang *et al.*, “A numa-aware key-value store for hybrid memory architecture,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1337–1348, 2022. doi: 10.1109/TPDS.2021.3099327. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9798134/>.
- [137] K. Wu, H. Chen, and B. Zang, “Archtm: Architecture-aware, high-performance transaction for persistent memory,” in *19th USENIX Conference on File and Storage Technologies (FAST 2021)*, USENIX Association, 2021, pp. 207–220. [Online]. Available: <https://www.usenix.org/conference/fast21/presentation/wu-kai>.
- [138] R. Zhao *et al.*, “Parallel external sorting of ascii records using learned models,” *arXiv preprint arXiv:2305.05671*, 2023. [Online]. Available: <https://arxiv.org/abs/2305.05671>.
- [139] Q. Liu *et al.*, “Efficient and atomic-durable persistent memory through in-pm hybrid logging,” *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’22)*, pp. 68–81, 2022. doi: 10.1109/ASPLOS.2022.00016. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9898581/>.
- [140] A. Sadiq, S. Islam, and M. Zulkernine, “Performance evaluation of intel optane memory for managed workloads,” *Proceedings of the 2021 ACM Symposium on Applied Computing*, 2021. doi: 10.1145/3451342. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3451342>.
- [141] R. Zhao, H. Chen, and B. Zang, “Plush: A write-optimized persistent log-structured hash-table,” *Proceedings of the VLDB Endowment*, vol. 15, no. 5, pp. 820–832, 2022. doi: 10.14778/3551793.3551839. [Online]. Available: <https://dl.acm.org/doi/abs/10.14778/3551793.3551839>.
- [142] X. Yang *et al.*, “Asymnvm: An efficient framework for implementing persistent data structures on asymmetric nvm architecture,” *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 185–200, 2020. doi: 10.1145/3373376.3378511. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3373376.3378511>.
- [143] W. Chen *et al.*, “Buffered hash table: Leveraging dram to enhance hash indexes in persistent memory,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 820–832, 2022. doi: 10.1109/TPDS.2021.3075113. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9898543/>.
- [144] X. Zhu *et al.*, “Revisiting log-structured merging for kv stores in hybrid memory systems,” in *Proceedings of the 2023 ACM SIGMOD International Conference on Management of Data*, ACM, 2023, pp. 1992–2006. doi: 10.1145/3575693.3575715. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3575693.3575715>.
- [145] R. Zhao, H. Chen, and B. Zang, “Halo: A hybrid pmem-dram persistent hash index with fast recovery,” in *Proceedings of the ACM on Management of Data (SIGMOD 2022)*, ACM, 2022, pp. 1597–1611. doi: 10.1145/3514221.3517884. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3514221.3517884>.
- [146] Y. Shindo *et al.*, “Metall: A persistent memory allocator for data-centric analytics,” *Parallel Computing*, vol. 111, p. 102 941, 2022. doi: 10.1016/j.parco.2022.102941. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167819122000114>.
- [147] C. Sun *et al.*, “An simd-accelerated metadata management scheme for persistent memory file systems,” *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1777–1790, 2022. doi: 10.1109/TC.2022.3167793. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9826066/>.
- [148] R. Zhao *et al.*, “Rewo-hash: A read-efficient and write-optimized hash table for intel optane dc persistent memory,” *Future Generation Computer Systems*, 2024. doi: 10.1016/j.future.2024.06.049. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X24003297>.
- [149] Y. Li *et al.*, “Iceberght: High performance hash tables through stability and low associativity,” in *Proceedings of the 52nd International Symposium on Computer Architecture (ISCA 2023)*, Association for Computing

- Machinery, 2023, pp. 606–619. DOI: 10.1145/3588727. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3588727>.
- [150] M. Chen *et al.*, “Gphash: An efficient hash index for gpu with byte-granularity persistent memory,” in *21st USENIX Conference on File and Storage Technologies (FAST 2025)*, USENIX Association, 2025. [Online]. Available: <https://www.usenix.org/conference/fast25/presentation/chen-menglei>.
 - [151] Z. Guo, Y. Luo, and H. Chen, “B3-tree: Byte-addressable binary b-tree for persistent memory,” *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1601–1616, 2020. DOI: 10.1145/3394025.3394144. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3394025>.
 - [152] Y. Yan *et al.*, “Ftgraph: A flexible tree-based graph store on persistent memory for large-scale dynamic graphs,” *IEEE Transactions on Parallel and Distributed Systems*, 2024. DOI: 10.1109/TPDS.2024.10740831. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10740831/>.
 - [153] T. Brown *et al.*, “Detectable recovery of lock-free data structures,” in *Proceedings of the ACM on Programming Languages (POPL 2022)*, ACM, 2022, 34:1–34:29. DOI: 10.1145/3503221.3508444. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3503221.3508444>.
 - [154] T. Brown and I. Almaqousi, “Elimination (a, b)-trees with fast, durable updates,” *Proceedings of the ACM on Programming Languages*, vol. 6, no. POPL, pp. 1–28, 2022. DOI: 10.1145/3508441. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3503221.3508441>.
 - [155] Z. He, T. Li, and J. Shu, “Failure-atomic byte-addressable r-tree for persistent memory,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 8, pp. 3288–3301, 2021. DOI: 10.1109/TKDE.2019.2918883. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9214450/>.
 - [156] Z. Xie *et al.*, “Write-optimized and consistent skiplists for non-volatile memory,” *IEEE Access*, vol. 9, pp. 69 850–69 859, 2021. DOI: 10.1109/ACCESS.2021.3071092. [Online]. Available: <https://ieeexplore.ieee.org/document/9424603>.
 - [157] Z. Zhang *et al.*, “Plin: A persistent learned index for non-volatile memory with high performance and instant recovery,” *Proceedings of the VLDB Endowment*, vol. 16, no. 2, pp. 243–255, 2022. DOI: 10.14778/3565816.3565826. [Online]. Available: <https://dl.acm.org/doi/abs/10.14778/3565816.3565826>.
 - [158] L. Wang *et al.*, “Cache-line transactions: Building blocks for persistent kernel data structures enabled by aspectc++,” in *Proceedings of the 14th European Conference on Computer Systems (EuroSys ’19)*, ACM, 2019, 40:1–40:16. DOI: 10.1145/3365137.3365396. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3365137.3365396>.
 - [159] T. Johnson, B. Liskov, and L. Shriram, “Recipe: Converting concurrent dram indexes to persistent-memory indexes,” *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2094–2107, 2019. DOI: 10.1145/3341301.3359635. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3341301.3359635>.
 - [160] Y. Zhou *et al.*, “Energy consumption evaluation of optane dc persistent memory for indexing data structures,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1595–1607, 2022. DOI: 10.1109/TPDS.2021.3121283. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10106332/>.
 - [161] L. Xiang *et al.*, “A performance study of intel optane persistent memory: From storage data structures’ perspective,” *Frontiers of Computer Science*, vol. 16, no. 4, 57:1–57:18, 2022. DOI: 10.1007/s42514-022-00123-x. [Online]. Available: <https://link.springer.com/article/10.1007/s42514-022-00123-x>.
 - [162] M. Tiwari *et al.*, “A performance study of optane persistent memory: From indexing data structures’ perspective,” in *Proceedings of the 16th IEEE International Conference on Mass Storage Systems and Technologies (MSST 2020)*, IEEE, 2020, pp. 1–6. [Online]. Available: <https://msstconference.org/MSST-history/2020/Papers/02.APerformanceStudyOptane.pdf>.
 - [163] M. Chen *et al.*, “Lock-free concurrent level hashing for persistent memory,” in *18th USENIX Conference on File and Storage Technologies (FAST 2020)*, USENIX Association, 2020, pp. 197–210. [Online]. Available: <https://www.usenix.org/conference/atc20/presentation/chen>.

- [164] R. Zhao *et al.*, “Optimizing both performance and tail latency for b+ tree on persistent memory,” *Journal of Systems Architecture*, p. 101 884, 2025. doi: 10.1016/j.sysarc.2025.101884. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1383762125000785>.
- [165] C. Wang, M. Chen, and H. Chen, “Seph: Scalable, efficient, and predictable hashing on persistent memory,” *Proceedings of the 17th USENIX Conference on Operating Systems Design and Implementation (OSDI '23)*, 2023. [Online]. Available: <https://www.usenix.org/conference/osdi23/presentation/wang-chao>.
- [166] R. Zhao *et al.*, “A write-optimal and concurrent persistent dynamic hashing with radix tree assistance,” *Journal of Systems Architecture*, vol. 126, p. 102 652, 2022. doi: 10.1016/j.sysarc.2022.102652. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1383762122000522>.
- [167] Y. Wang *et al.*, “Hierarchical hashing: A dynamic hashing method with low write amplification and high performance for non-volatile memory,” *IEEE Transactions on Parallel and Distributed Systems*, 2025. doi: 10.1109/TPDS.2025.10803027. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10803027/>.
- [168] Y. Sun *et al.*, “Canrt: A client-active nvm-based radix tree for fast remote access,” in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXI*, Springer, 2020, pp. 573–594. doi: 10.1007/978-3-030-60245-1_30. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-60245-1_30.
- [169] X. Chen *et al.*, “Hmeh: Write-optimal extendible hashing for hybrid dram-nvm memory,” *IEEE International Conference on Mass Storage Systems and Technologies*, pp. 1–14, 2020. [Online]. Available: <https://msstconference.org/MSST-history/2020/Papers/11.HMEH.pdf>.
- [170] Y. Zhou *et al.*, “Consistent rdma-friendly hashing on remote persistent memory,” *IEEE Transactions on Parallel and Distributed Systems*, 2024. doi: 10.1109/TPDS.2024.9643819. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9643819/>.
- [171] X. Jin *et al.*, “Lb+ trees: Optimizing persistent index performance on 3d xpoint memory,” *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3085–3097, 2020. doi: 10.14778/3384345.3384355. [Online]. Available: <https://dl.acm.org/doi/abs/10.14778/3384345.3384355>.
- [172] R. Zhao, B. Zang, and H. Chen, “Tlbtrees: A read/write-optimized tree index for non-volatile memory,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 136–149, 2021. doi: 10.1109/TPDS.2020.3028031. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9458817/>.
- [173] J. Liu *et al.*, “Turbohash: A hash table for key-value store on persistent memory,” *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 4363–4375, 2023. doi: 10.1145/3579370.3594766. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3579370.3594766>.
- [174] J. Izraelevitz *et al.*, “Durable queues: The second amendment,” in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '21)*, ACM, 2021, pp. 1201–1214. doi: 10.1145/3409964.3461791. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3409964.3461791>.
- [175] J. Lyu *et al.*, “Hdnh: A read-efficient and write-optimized hashing scheme for hybrid dram-nvm memory,” in *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, ACM, 2021, pp. 2914–2927. doi: 10.1145/3472456.3473515. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3472456.3473515>.
- [176] L. Brandon and Z. Xuechen, “Pm-rtree: A highly-efficient crash-consistent r-tree for persistent memory,” in *Proceedings of the 34th International Conference on Scientific and Statistical Database Management (SSDBM 2022)*, New York, NY, USA: Association for Computing Machinery, Jul. 2022. doi: 10.1145/3538712.3538713. [Online]. Available: <https://dl.acm.org/doi/10.1145/3538712.3538713>.
- [177] R. Guerraoui *et al.*, “Mirror: Making lock-free data structures persistent,” *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021. doi: 10.1145/3454105. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3453483.3454105>.

- [178] M. Lu *et al.*, “Revisiting persistent hash table design for commercial non-volatile memory,” *IEEE Transactions on Computers*, vol. 69, no. 12, pp. 1795–1808, 2020. doi: 10.1109/TC.2020.3003978. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9116223/>.
- [179] K. Kim *et al.*, “Pactree: A high performance persistent range index using pac guidelines,” in *VLDB 2022 - Conference paper (author self-archived PDF)*, 2022. [Online]. Available: <https://rs3lab.github.io/assets/papers/2021/kim:pactree.pdf>.
- [180] Y. Wang *et al.*, “Mosiqs: Persistent memory object storage with metadata indexing and querying for scientific computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 860–873, 2021. doi: 10.1109/TPDS.2020.3017089. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9448213/>.
- [181] W. Liu *et al.*, “Capri: Compiler and architecture support for whole-system persistence,” *Proceedings of the 2022 ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI ’22)*, pp. 784–799, 2022. doi: 10.1145/3502181.3531474. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3502181.3531474>.
- [182] Q. Zhou *et al.*, “Optimizing b+-tree for hybrid memory with in-node hotspot cache and eadr awareness,” *Frontiers of Computer Science*, vol. 18, p. 113, 2024. doi: 10.1007/s11704-023-3344-x. [Online]. Available: <https://link.springer.com/article/10.1007/s11704-023-3344-x>.
- [183] S. Mackenzie and D. M. Waddington, “A fast, general system for buffered persistent data structures,” in *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, ACM, 2021, pp. 3033–3037. doi: 10.1145/3472456.3472458. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3472456.3472458>.
- [184] J. Izraelevitz, S. Swanson, and M. L. Scott, “Mod: Minimally ordered durable datastructures for persistent memory,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ACM, 2020, pp. 1637–1650. doi: 10.1145/3373376.3378472. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3373376.3378472>.
- [185] J. Wang *et al.*, “Pacman: An efficient compaction approach for log-structured key-value store on persistent memory,” in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, USENIX Association, 2022, pp. 849–863. [Online]. Available: <https://www.usenix.org/conference/atc22/presentation/wang-jing>.
- [186] Y. Lu *et al.*, “Octopus+: An rdma-enabled distributed persistent memory file system,” in *Proceedings of the 16th European Conference on Computer Systems (EuroSys ’21)*, ACM, 2021, pp. 420–436. doi: 10.1145/3448418. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3448418>.
- [187] Z. Wang, Y. Zhang, and H. Chen, “Moonkv: Optimizing update-intensive workloads for nvm-based key-value stores,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 4, pp. 678–689, 2023. doi: 10.1109/TCAD.2023.3169521. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10415821/>.
- [188] D. G. Waddington *et al.*, “An architecture for memory centric active storage (mcas),” *arXiv preprint arXiv:2103.00007*, 2021. [Online]. Available: <https://arxiv.org/abs/2103.00007>.
- [189] L. Yang *et al.*, “Burger-tree: A three-layer cache-conscious tree index for persistent memory,” *IEEE Transactions on Computers*, 2022. doi: 10.1109/TC.2022.3162645. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10074794/>.
- [190] W. Tang *et al.*, “P-massive: A real-time search engine for a multi-terabyte mass spectrometry database,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1073–1086, 2022. doi: 10.1109/TPDS.2021.3093614. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10046108/>.
- [191] G. Gu, T. Wang, and H. Chen, “Reh: Redesigning extendible hashing for commercial non-volatile memory,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1635–1647, 2022. doi: 10.1109/TPDS.2021.3076637. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9774546/>.

- [192] S. He *et al.*, “Optimizing adaptive radix trees for nvm-based hybrid memory architecture,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1819–1833, 2021. doi: 10.1109/TPDS.2021.3068801. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9378180/>.
- [193] Z. Xu *et al.*, “Portus: Efficient dnn checkpointing to persistent memory with zero-copy,” in *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2024, pp. 380–393. doi: 10.1109/ISCA60324.2024.00036. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10630969/>.
- [194] Y. Chen *et al.*, “Enforcing crash consistency of scientific applications in non-volatile main memory systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 2, pp. 343–356, 2019. doi: 10.1109/TPDS.2018.2855516. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8945958/>.
- [195] S. He *et al.*, “Hbtree: An efficient index structure based on hybrid dram-nvm,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 719–732, 2021. doi: 10.1109/TPDS.2020.3034917. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9628870/>.
- [196] W. Wu *et al.*, “On the performance intricacies of persistent memory aware storage engines,” *IEEE Transactions on Parallel and Distributed Systems*, 2023. doi: 10.1109/TPDS.2023.3251891. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10091894/>.
- [197] M. S. H. Molla *et al.*, “Hamming tree: The case for energy-aware indexing for nvms,” *Proceedings of the ACM on Management of Data*, vol. 1, no. SIGMOD/PODS, pp. 1710–1722, 2023. doi: 10.1145/3589327. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3589327>.
- [198] L. A. Moura *et al.*, “Pronto: Easy and fast persistence for volatile data structures,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ACM, 2020, pp. 2861–2875. doi: 10.1145/3373376.3378456. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3373376.3378456>.
- [199] Y. Sun *et al.*, “Characterizing the dilemma of performance and index size in billion-scale vector search and breaking it with second-tier memory,” *arXiv preprint arXiv:2405.03267*, 2024. [Online]. Available: <https://arxiv.org/abs/2405.03267>.
- [200] Z. Hu *et al.*, “Flydb: Query optimization of blockchain system based on hybrid storage architecture,” *IEEE Transactions on Computers*, 2023. doi: 10.1109/TC.2023.3291620. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10212824/>.
- [201] Y. Tang *et al.*, “Single-node partitioned-memory for huge graph analytics: Cost and performance trade-offs,” *Proceedings of the ACM Symposium on Cloud Computing (SoCC 2021)*, pp. 611–623, 2021. doi: 10.1145/3458817.3476156. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3458817.3476156>.
- [202] J. Shen *et al.*, “Autotm: Automatic tensor movement in heterogeneous memory systems using integer linear programming,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 4, pp. 1–23, 2020. doi: 10.1145/3378465. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3373376.3378465>.
- [203] Z. Tang *et al.*, “Tac: An anti-caching key-value store on heterogeneous memory architectures,” in *Proceedings of the 27th International Conference on Extending Database Technology (EDBT 2024)*, OpenProceedings, 2024, pp. 1151–1162. [Online]. Available: <https://www.comp.nus.edu.sg/~huang/assets/works/EDBT-2024/TaC/paper-121.pdf>.
- [204] S. He *et al.*, “Hbtree: A heterogeneous b+ tree with multi-granularity for hybrid nvm-ssd storage,” *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 274–288, 2022. doi: 10.1109/TC.2021.3073084. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9925539/>.
- [205] Q. Zhou *et al.*, “An lsm tree augmented with b+ tree on nonvolatile memory,” *Proceedings of the ACM on Management of Data*, 2024. doi: 10.1145/3633475. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3633475>.

- [206] F. Wang *et al.*, “Umap: Enabling application-driven optimizations for page management,” *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1500–1514, 2021. doi: 10.1109/TC.2021.3074682. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9407112/>.
- [207] Z. Qian *et al.*, “Mtm: Rethinking memory profiling and migration for multi-tiered large memory,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Association for Computing Machinery, 2024, pp. 785–800. doi: 10.1145/3627703.3650075. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3627703.3650075>.
- [208] T. Zhang, H. Chen, and B. Zang, “Persistent state machines for recoverable in-memory storage systems with nvram,” in *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys 2020)*, Association for Computing Machinery, 2020, pp. 1–15. doi: 10.5555/3488766.3488824. [Online]. Available: <https://dl.acm.org/doi/10.5555/3488766.3488824>.
- [209] X. Zhang *et al.*, “Flexible and effective object tiering for heterogeneous memory systems,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2025. doi: 10.1145/3708540. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3708540>.
- [210] R. Zhao, H. Chen, and B. Zang, “Portable application guidance for complex memory systems,” *Proceedings of the ACM Symposium on Cloud Computing (SoCC 2019)*, pp. 19–32, 2019. doi: 10.1145/3357526.3357575. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3357526.3357575>.
- [211] Z. Zhou *et al.*, “Unimem: Runtime data management on non-volatile memory-based heterogeneous main memory for high performance computing,” *Science China Information Sciences*, vol. 64, no. 9, p. 192 102, 2021. doi: 10.1007/s11390-020-0942-z. [Online]. Available: <https://link.springer.com/article/10.1007/s11390-020-0942-z>.
- [212] F. Shen *et al.*, “Lb-hm: Load balance-aware data placement on heterogeneous memory for task-parallel hpc applications,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 2, 40:1–40:21, 2022. doi: 10.1145/3503221.3508406. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3503221.3508406>.
- [213] J. Krishnan, C. Li, and O. Mutlu, “Tips: Making volatile index structures persistent with dram-nvmm tiering,” in *19th USENIX Conference on File and Storage Technologies (FAST ’21)*, USENIX Association, 2021, pp. 25–38. [Online]. Available: <https://www.usenix.org/conference/atc21/presentation/krishnan>.
- [214] H. Gong *et al.*, “Athena: High-performance sparse tensor contraction sequence on heterogeneous memory,” *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining*, pp. 3550–3559, 2021. doi: 10.1145/3447818.3460355. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3447818.3460355>.
- [215] Q. Zhou *et al.*, “Hemem: Scalable tiered memory management for big data applications and real nvmm,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 3, 45:1–45:22, 2021. doi: 10.1145/3477132.3483550. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3477132.3483550>.
- [216] C. Wang *et al.*, “Merchandiser: Data placement on heterogeneous memory for task-parallel hpc applications with load-balance awareness,” *Proceedings of the 27th ACM International Conference on Supercomputing (ICS 2023)*, 2023. doi: 10.1145/3572848.3577497. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3572848.3577497>.
- [217] L. Xiang *et al.*, “Characterizing the performance of intel optane persistent memory: A close look at its on-dimm buffering,” *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2022)*, pp. 787–800, 2022. doi: 10.1145/3492321.3519556. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3492321.3519556>.
- [218] Z. Zhou *et al.*, “Atmem: Adaptive data placement in graph applications on heterogeneous memories,” *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP ’20)*, pp. 410–424, 2020. doi: 10.1145/3368826.3377922. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3368826.3377922>.

- [219] L. Jin *et al.*, “Cori: Dancing to the right beat of periodic data movements over hybrid memory systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 17–31, 2021. doi: 10.1109/TPDS.2020.3008152. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9460495/>.
- [220] X. Fan *et al.*, “Optimizing large-scale plasma simulations on persistent memory-based heterogeneous memory with effective data placement across memory hierarchy,” *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2632–2640, 2021. doi: 10.1145/3447818.3460356. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3447818.3460356>.
- [221] T. Rabl, L. Voigt, and F. Funke, “Online application guidance for heterogeneous memory systems,” *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, pp. 1567–1578, 2022. doi: 10.1145/3533855. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3533855>.
- [222] Y. He *et al.*, “Flexhm: A practical system for heterogeneous memory with flexible and efficient performance optimizations,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 3, 44:1–44:20, 2022. doi: 10.1145/3565885. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3565885>.
- [223] H. Chen, J. Shu, and H. Chen, “On the implications of heterogeneous memory tiering on spark in-memory analytics,” *IEEE Transactions on Parallel and Distributed Systems*, 2023. doi: 10.1109/TPDS.2023.3289806. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10196534/>.
- [224] J. Feng *et al.*, “Persistent memory as an effective alternative to ram in metagenome assembly,” *BMC Bioinformatics*, vol. 23, no. 1, p. 101, 2022. doi: 10.1186/s12859-022-05052-8. [Online]. Available: <https://link.springer.com/article/10.1186/s12859-022-05052-8>.
- [225] Z. Yue *et al.*, “Towards persistent memory-based stateful serverless computing for big data applications,” in *Proceedings of the 14th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2023)*, USENIX Association, 2023. [Online]. Available: https://people.cs.vt.edu/lyuze/files/pm_serverless.pdf.
- [226] K. Zhou *et al.*, “Architecting ddr5 dram caches for non-volatile memory systems,” *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2022)*, pp. 457–469, 2022. doi: 10.1145/3489517.3530570. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3489517.3530570>.
- [227] L. Zhou *et al.*, “Optimizing the page hotness measurement with re-fault latency for tiered memory systems,” *Journal of Computer Science and Technology*, 2024. doi: 10.1007/s11390-024-3365-8. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9736522/>.
- [228] J. Li, J. Shu, and H. Chen, “Integer compression in nvram-centric data stores: Comparative experimental analysis to dram,” *Proceedings of the 25th ACM SIGMOD International Conference on Management of Data*, pp. 2005–2018, 2019. doi: 10.1145/3329785.3329923. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3329785.3329923>.
- [229] P. Dong *et al.*, “Initial experience with 3d xpoint main memory,” *Cluster Computing*, vol. 23, no. 3, pp. 1721–1737, 2020. doi: 10.1007/s10619-019-07277-8. [Online]. Available: <https://link.springer.com/article/10.1007/s10619-019-07277-8>.
- [230] Y. Chen *et al.*, “Symbiotic hw cache and sw dtlb prefetching for dram/nvm hybrid memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 3, pp. 497–510, 2020. doi: 10.1109/TCAD.2019.2940115. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9285963/>.
- [231] R. Zhao, H. Chen, and B. Zang, “A study of r-tree performance in hybrid flash/3d xpoint storage,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 10, pp. 1846–1859, 2019. doi: 10.1109/TKDE.2018.2876205. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8900716/>.
- [232] H. Ren, X. Liao, and H. Chen, “Evaluation of intel 3d-xpoint nvdim technology for memory-intensive genomic workloads,” *Proceedings of the ACM Symposium on Cloud Computing (SoCC 2019)*, pp. 28–39, 2019. doi: 10.1145/3357526.3357528. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3357526.3357528>.

- [233] Z. Zhao *et al.*, “Workload-driven placement of column-store data structures on dram and nvm,” in *Proceedings of the 2021 International Conference on Management of Data (SIGMOD ’21)*, ACM, 2021, pp. 1245–1258. doi: 10.1145/3465998.3466008. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3465998.3466008>.
- [234] A. Chakraborty and K. Pingali, “Maphea: A lightweight memory hierarchy-aware profile-guided heap allocation framework,” in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI ’21)*, ACM, 2021, pp. 984–999. doi: 10.1145/3461648.3463844. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3461648.3463844>.
- [235] K. Zhou *et al.*, “Evaluating persistent memory range indexes: Part two,” *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2463–2476, 2022. doi: 10.14778/3551793.3551808. [Online]. Available: <https://dl.acm.org/doi/abs/10.14778/3551793.3551808>.
- [236] G. Liu *et al.*, “Memory management methodology for application data structure refinement and placement on heterogeneous dram/nvm systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1970–1984, 2022. doi: 10.1109/TPDS.2022.3177942. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9774666/>.
- [237] Y. Li *et al.*, “Performance, energy and nvm lifetime-aware data structure refinement and placement for heterogeneous memory systems,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 9, no. 1, pp. 1–32, 2025. doi: 10.1145/3736174. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3736174>.
- [238] Y. Lin *et al.*, “Pm-blade: A persistent memory-augmented lsm-tree storage for database,” *IEEE Transactions on Computers*, vol. 72, no. 5, pp. 1261–1274, 2023. doi: 10.1109/TC.2023.3270605. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10184789/>.
- [239] H. Avni *et al.*, “Extending in-memory oltp with persistent memory,” Tech. Rep., 2020, Technical report.
- [240] F. Shen *et al.*, “Sparta: High-performance, element-wise sparse tensor contraction on heterogeneous memory,” *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2021)*, pp. 24–38, 2021. doi: 10.1145/3437801.3441581. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3437801.3441581>.
- [241] X. Huang *et al.*, “Demystifying the performance of hpc scientific applications on nvm-based memory systems,” *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 327–336, 2020. doi: 10.1109/IPDPS47924.2020.00098. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9139816/>.
- [242] Y. Xu *et al.*, “Ecohmem: Improving object placement methodology for hybrid memory systems in hpc,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 837–850, 2022. doi: 10.1109/TPDS.2021.3112379. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9912666/>.
- [243] X. Yang *et al.*, “Performance potential of mixed data management modes for heterogeneous memory systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2740–2753, 2020. doi: 10.1109/TPDS.2020.2991967. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9306996/>.
- [244] R. Zhao *et al.*, “System evaluation of the intel optane byte-addressable nvm,” in *Proceedings of the ACM Symposium on Cloud Computing (SoCC 2019)*, Association for Computing Machinery, 2019, pp. 259–273. doi: 10.1145/3357526.3357568. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3357526.3357568>.
- [245] M. Arbel, E. Petrank, and T. Brown, “Petra: Persistent transactional non-blocking linked data structures,” *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–30, 2021. doi: 10.1145/3446391. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3446391>.
- [246] Z. Zhou *et al.*, “Understanding and improving persistent transactions on optane™ dc memory,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1244–1257, 2020. doi: 10.1109/TPDS.2019.2915410. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9139799/>.

- [247] M. Arbel, T. Brown, and E. Petrank, “Don’t persist all: Efficient persistent data structures,” *arXiv preprint arXiv:1905.13011*, 2019. [Online]. Available: <https://arxiv.org/abs/1905.13011>.
- [248] C. Su *et al.*, “Revitalizing the forgotten on-chip dma to expedite data movement in nvm-based storage systems,” in *21st USENIX Conference on File and Storage Technologies (FAST 2023)*, USENIX Association, 2023. [Online]. Available: <https://www.usenix.org/conference/fast23/presentation/su>.
- [249] P. Zhang *et al.*, “Recovery of distributed iterative solvers for linear systems using non-volatile ram,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1923–1936, 2022. doi: 10.1109/TPDS.2021.3065458. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10024018/>.
- [250] V. Mantripragada *et al.*, “Fine-grained policy-driven i/o sharing for burst buffers (themisio),” in *Proceedings of the 28th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2023)*, ACM, 2023, pp. 428–440. doi: 10.1145/3581784.3607041. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3581784.3607041>.
- [251] M. Zhang *et al.*, “Ford: Fast one-sided rdma-based distributed transactions for disaggregated persistent memory,” in *20th USENIX Conference on File and Storage Technologies (FAST 2022)*, USENIX Association, 2022, pp. 255–270. [Online]. Available: <https://www.usenix.org/conference/fast22/presentation/zhang-ming>.
- [252] J. Liu *et al.*, “Exploring efficient architectures on remote in-memory nvm over rdma,” *ACM Transactions on Storage*, vol. 17, no. 2, pp. 17:1–17:24, 2021. doi: 10.1145/3477004. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3477004>.
- [253] R. Zhao, H. Chen, and B. Zang, “Accelerating range queries of primary and secondary indices for key-value separation,” in *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, ACM, 2022, pp. 3090–3102. doi: 10.1145/3542929.3563479. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3542929.3563479>.
- [254] D. Liu *et al.*, “A hierarchical heterogeneous iot time series data index for nvm,” *IEEE Internet of Things Journal*, 2025. doi: 10.1109/JIOT.2025.1234567. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10831234/>.
- [255] G. Lin *et al.*, “Scheduling hpc workflows with intel optane persistent memory,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2439–2452, 2021. doi: 10.1109/TPDS.2020.3030805. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9460699/>.
- [256] X. Wang *et al.*, “A novel non-volatile memory update mechanism for 6g edge computing,” *Future Generation Computer Systems*, vol. 140, pp. 52–62, 2023. doi: 10.1016/j.future.2022.10.047.
- [257] J. Arulraj *et al.*, “Integrating non-volatile main memory in a deterministic database,” in *Conference on Innovative Data Systems Research (CIDR’15)*, 2015, pp. 1–7.
- [258] H. Almatary *et al.*, “Splitfs: Reducing software overhead in file systems for persistent memory,” in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 645–659. [Online]. Available: <https://www.usenix.org/conference/nsdi20/presentation/almatary>.
- [259] Q. Li *et al.*, “Light-dedup: A light-weight inline deduplication framework for non-volatile memory file systems,” in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 125–136. doi: 10.1109/DSN48987.2021.00025.
- [260] Y. Lu *et al.*, “Squirrels: Using the rust compiler to check file-system crash consistency,” in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023, pp. 325–343. [Online]. Available: <https://www.usenix.org/conference/fast23/presentation/lu>.
- [261] A. Baumann *et al.*, “High velocity kernel file systems with bento,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 541–557. [Online]. Available: <https://www.usenix.org/conference/osdi20/presentation/baumann>.