DOI:10.1145/3769003

Yu Hua, Xue Liu, and Ion Stoica

Opinion
The Golden Rule of Big Memory: Persistence Is Not Harmful

Seeking a transformative memory scheme that grows in performance and capacity as the infrastructure expands.

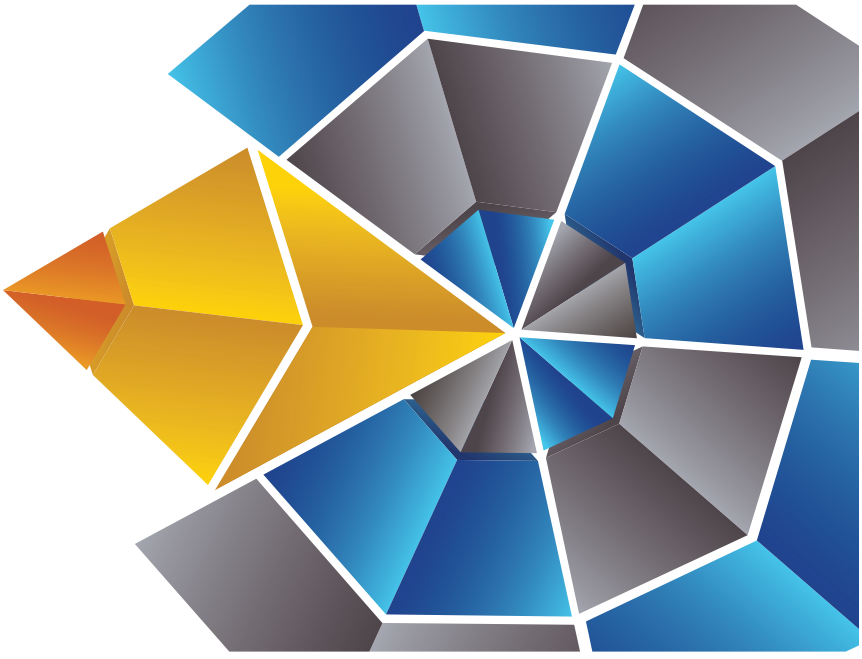

IMAGE BY ANDRIJ BORYS ASSOCIATES

Conventional memory schemes follow the Pareto Principle, in which approximately maintaining 20% hot data can meet 80% of requests. Large-scale applications, such as generative AI, recommendation systems, big data, and HPC systems, require large-capacity and high-speed memory and are changing the power-law locality, which necessitate the support of *Big Memory*. Big Memory is a transformative memory-centric system that consolidates massive and heterogeneous memory resources into a unified and shared address space, delivering near-DRAM latency and persistence at a terabyte-to-petabyte scale. This paradigm mitigates traditional I/O bottlenecks, simplifies data placement and programming models, and unlocks unprecedented scalability.

Memory is one of the most contended hardware resources and often becomes the performance bottleneck in the I/O critical path that connects computing with storage. Due to limited memory capacity, more applications have to explore and exploit the memory beyond traditional local main memory, including empty memory on remote servers, and disaggregated memory in a memory pool.[7] This non-local far memory can expand memory size and avoid memory stranding. Since far memory is much slower than local memory, existing systems leverage local memory as a cache by transparently swapping memory pages between local and far memory, which unfortunately causes read or write amplification.

To deliver high performance and achieve cost-efficiency, we need to carefully consider memory overhead that comes from the differential requirements of high-level applications and low-level devices. Specifically, high-level applications generally include peer-to-peer, blockchain, artificial intelligence, computability storage, and so forth. Different storage applications exhibit different needs and access patterns. On the other hand, the low-level devices become heterogeneous and highly hierarchical, which exhibit different physical properties.[3] For example, hard disks, SSD and PCM are non-volatile, and DRAM, cache, and registers are volatile. Hard disks and SSD are block-addressable, while others are byte-addressable. SSD and PCM suffer from the endurance with limited writes. These two-level different requirements, like a hamburger, introduce the high memory overhead, which means that to support program execution in memory, we need to consume extra costs for data movements and energy consumption, as well as high-security risks.[1] To alleviate high memory overhead, we build big memory that not only provides large capacity

but also supports fast persistence. The persistence guarantees a complete lifetime for data to be finally written into non-volatile devices via protocol buffers or checkpoints.[2] Since most operations can be executed and completed within the big memory, we significantly reduce data movements through the memory and storage hierarchy.

In this hierarchical architecture, modern networking technologies such as RDMA and CXL are erasing the boundaries between memory and storage, turning distributed resources into a seamless, low-latency memory fabric. CXL enables elastic, high-speed memory pooling across servers at near-DRAM speed, whereas RDMA facilitates efficient data transfers by bypassing CPU and software overheads. Instead of slow block storage, systems now operate on byte-addressable memory, where persistence is a property. Traditional cache, DRAM, and disk hierarchies become inefficient, thus replacing them with a scalable network-spanning memory system that grows in performance and capacity as the infrastructure expands.

The transformation of conventional memory into big memory systems leverages both vertical and horizontal extensions to existing memory hierarchies. Vertically, this expansion enables each node to host significantly larger datasets by increasing memory capacity and delivering high performance within a flattened hierarchy that coalesces traditional memory and storage tiers into a unified, high-speed access layer. Horizontally, the architecture scales out across distributed systems, integrating multiple nodes into a pool that collectively provides massive, scalable memory resources while maintaining efficient data access and coherence across the entire system. More data is aggregated together via atomic operation guarantee, which actually coalesces the central and distributed designs. Hence, by leveraging the vertical and horizontal ways, we build a big computer, as shown in Figure 1.

The foundational rationale for this architecture stems from the influential methodology of "the computer as a network." This perspective draws a structural analogy between traditional computer hierarchy, which comprises low-level storage, intermediate memory, and high-level CPUs, and networked systems, where analogous roles are fulfilled by remote cloud resources, adjacent edge nodes, and end-user clients, respectively. This conceptual mapping allows for the application of well-understood network principles to the design of integrated computing systems, thereby enabling more scalable, efficient, and flexible resource orchestration across localized and distributed environments. In the context of a big computer, these items share some similarities. The long-latency storage seems like a remote cloud, the high-speed memory is like an adjacent edge, and the computing units are like clients that generate data. After being generated from clients/CPUs, data will be instantaneously transmitted to the close-by edges/memory, and consume a long latency to arrive at the remote clouds/storage.

Figure 1. Big computer coalesces the memory and storage, and the central and distributed schemes.

This methodology can be implemented via existing devices. For example, non-volatile devices offer about TB-scale memory capacity, which can be considered a vertical extension. Moreover, from the horizontal view, RDMA and CXL protocols support memory and cache coherence.[7] The

The foundational rationale for this architecture stems from the influential methodology of "the computer as a network."

big computer hence achieves persistence in both vertical and horizontal extensions. A program in a big computer can be atomically executed in an end-to-end way, whether from CPU to non-volatile devices or from clients to clouds.

Understanding the 'Big'

In the context of big memory, we need to clarify the *BIG*, which is also the essential difference between big memory and traditional one. To fully understand the mentioned BIG, we first discuss other related BIG terms in the research community. Specifically, the difference between data and big data is the well-known Vs, including volume, velocity, variety, veracity, and value. Moreover, the difference between a model and a big model comes from algorithms, data, and computation. Compared with a chip, the big chip needs to consider the programmable, scalable, highly available, and adaptive properties. The salient feature behind BIGs is their Golden Rule.

The transition from conventional memory to "big memory" necessitates a clear understanding of their fundamental differences and the establishment of a guiding principle, that is, the "Golden Rule of Big Memory." Vertically, this evolution signifies not only an expansion in capacity but also an extension in effective access distance, moving beyond local DRAM to incorporate broader hierarchies. Horizontally, it introduces immense scaling complexity in distributed systems deploying numerous GPUs for large model training. In such environments, horizontal scaling often relies on distributed shared memory to temporarily maintain intermediate results as replicas in remote volatile memory, offering probabilistic persistence but posing high risks of data loss. Without robust persistence guarantee, these volatile checkpoints consume excessive memory space and incur substantial broadcast overhead for synchronization and recovery.[5] This problem is exemplified by large language models, whose checkpoint states reach terabyte scales, which far surpasses the capacity of individual GPU memories. This leads to significant performance degradation when remote GPUs cannot host entire checkpoints.

Persistence needs to be guaranteed by storing states and data in non-volatile storage devices.

The Golden Rule

We believe that *persistence* is the Golden Rule of Big Memory systems. Contemporary big memory architectures have fundamentally reimagined data persistence, transforming it from a system constraint into a powerful performance enabler. This shift is driven by hardware-optimized approaches. Specifically, speculative persistence employs parallel RDMA replication to maintain multiple synchronized copies across nodes, enabling non-blocking durable operations. Deterministic persistence leverages CXL-connected NVM with efficient cache management to deliver memory-tier durability without traditional storage penalties. The result is a paradigm where persistence accelerates rather than hinders operations. In-memory databases achieve instant commits, machine learning systems maintain seamless checkpoints and distributed applications preserve consistency at memory speeds. By merging the performance of volatile memory with the reliability of persistent storage, this advancement is reshaping the foundation of memory-centric computing.

A case in point comes from an insight that both Von Neumann and non-Von Neumann architectures need to guarantee the persistence. In practice, the memory/storage hierarchy in the Von Neumann architecture is a multi-level subset model, which includes disks, SSD, PCM, DRAM, L1/L2/L3 caches, and registers. In this subset model, the devices in the lower levels contain the data in the higher ones. All data needs to be written into persistent devices. On the other hand, for non-Von Neumann architecture, persistence still plays an important role. A typical workflow in the memristors needs to compute the input vector in the matrix and the result is a new vector that is volatile. By using extra operations, the new vector can be finally persisted in the non-volatile devices. Hence, both Von Neumann and non-Von Neumann architectures follow the persistence principle.

There exist some non-persistence operations in conventional systems, for example, replicating data into the DRAMs across servers or writing data in a batch through a cache/memory/storage hierarchy. These operations often cause misleading the understanding that persistence is harmful to high performance, which is actually not. To avoid the high overheads of rereading or recomputing the lost data, systems persist data but suffer from the huge gap between high-speed memory and low-speed disks. Many efforts have to be put into maintaining data in a volatile manner, which is vulnerable to just one system crash or failure. This awkward situation can be efficiently addressed using the big memory that offers large-size non-volatility with a shortened critical path. For example, eADR (extended Asynchronous DRAM Refresh) offers persistence for in-cache data and guarantees to flush them into non-volatile devices in case of system crash or failure. Hence, the number of flushing data is significantly reduced even during normal system execution. Systems thus deliver high performance with the aid of persistence.

Conventional computing systems operate on a heterogeneous hierarchy: fast, byte-addressable, volatile main memory, and slow, block-addressable, non-volatile storage. This separation necessitates data to migrate frequently and expensively between the two tiers to achieve persistence, which not only introduces significant performance overheads due to constant data swapping but also amplifies the risk of consistency bugs. Persistence needs to be guaranteed by storing states and data in non-volatile storage devices. A program hence transfers states and data to and from storage devices via an I/O bus, which is supported by leveraging specific instructions in a program. Persistence is important to deal with system failure and power down without data loss.

Persistent memory[9] fundamentally rearchitects data persistence by merg-

Figure 2. The full-stack and moving persistence.

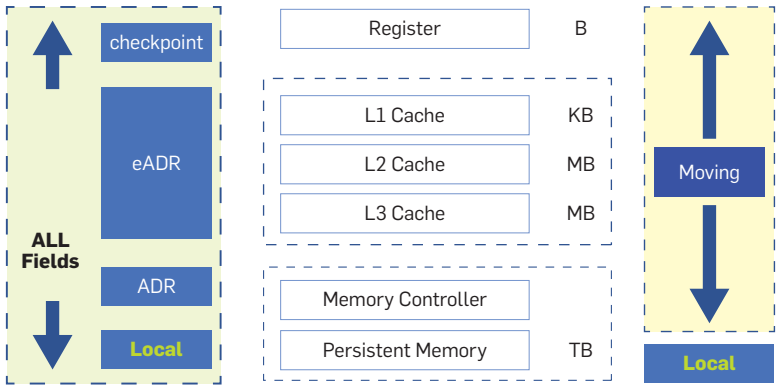

Table. Quantitative performance comparisons in terms of multiple metrics.

Applications	Metrics Examples	Throughput	Memory Overhead	Operation Latency
Non-Persistent Memory	Standard	1	1	1
Transactions	Motor[OSDI 24]	2.15	0.57	N/A
	SiLo[HPCA 23]	3.63	0.24	N/A
Indexing	DALdex[ICS 25]	2.66	0.61	N/A
	Level[OSDI 18]	N/A	N/A	0.42(Insert)/0.51(Update)
	LightCheck[ICCD 23]	N/A	0.62	0.07(Recovery)
	FINEdex[VLDB 22]	2.26	0.05	N/A
Security	STAR[HPCA 21]	N/A	N/A	0.35 (Write)
	Secon[DAC 22]	1.95	N/A	N/A

ing memory and storage into a single tier, offering DRAM-like performance with disk-like non-volatility through a unified, flat address space. This design allows applications to maintain consistent states directly in memory and flush them to non-volatile media in batches,[6] reducing the costly data movements while simultaneously avoiding security vulnerabilities associated with bus snooping. The flat addressing model simplifies error-prone persistence logic, enables near-instantaneous recovery, and permits direct manipulation of persistent data. However, while technologies like eADR ensure the persistence of data residing in multi-level CPU caches, data within registers remain vulnerable to power loss, as onboard power capacitors cannot efficiently flush register contents to non-volatile media. To address this limitation, modern systems employ checkpointing techniques to capture and guarantee the persistence of full-stack architectural states, ensuring comprehensive recovery integrity across the entire hierarchy.

Full-Stack and Moving Persistence

As shown in Figure 2, achieving full-stack persistence across the entire memory hierarchy represents a paradigm shift in system design, fundamentally moving the persistence boundary from the traditional interface between SSDs and DRAM directly into CPU. This transformation is enabled through a combination of hardware and software mechanisms. Persistent memory provides foundational data persistence at scale due to its inherent non-volatile physical properties. ADR ensures that data within the memory controller are flushed to non-volatile media during power events. eADR further guarantees that data residing in multi-level CPU caches are persisted. To address the volatility of registers, which remain beyond the reach of capacitor-based flushing mechanisms, systems employ frequent and expensive checkpointing techniques to capture computational states.[8] Consequently, the entire hierarchy, from computational registers to large-scale persistent memory, now supports end-to-end persistence, enabling not only near-instantaneous recovery and simplified programming models but also a substantial expansion of effective memory capacity available for persistent operations.

Full-stack persistence requires the support from non-volatile devices and periodic checkpointing to facilitate fast recovery from system crashes or power failures. Although this hierarchy supports full-stack persistence, the implementations are different. For persistent memory, persistence can be naturally obtained due to its physical features. However, the memory controller and multi-level caches, which are volatile, require extra power to flush data into non-volatile devices, called moving persistence. Since data move via the I/O bus, there exist security issues due to bus snooping. These problems can be addressed by efficiently coordinating the hierarchy, capacity, and operations for high performance and strong security, thus achieving a suitable trade-off between performance and security.

For large-scale distributed systems, we conventionally leverage IP protocol to connect the hard disks in multiple nodes. RDMA is further used to aggregate memory resources together. CXL can achieve the cache coherence.[4] These designs allow the processing boundary to move up and more operations can be completed in the higher levels. Since data need to be finally written into non-volatile devices, the end-to-end critical path can be significantly shortened. We hence achieve the savings of data movements and decrease energy consumption, as well as alleviating bus conflicts and side-channel attacks.

Quantitative Validation

To demonstrate the real-world impact of persistence, we select our state-of-

the-art designs and real implementations for persistent memory, which provide detailed comparisons with standard non-persistent memory systems (that is, conventional DRAM-based systems). As shown in the accompanying table, the widely used evaluation metrics include throughput, memory overhead, and typical operation latency. The comparisons leverage various workloads from real-world applications and large-scale benchmarks. These workloads exhibit the typical patterns of existing and emerging applications in the context of big memory. We mainly show the average values due to space limitation. More details about configurations can be found in the experiments of our original papers. Moreover, applications consist of transactions, indexing, and security, which are widely used and well recognized in the memory community. Due to the heterogeneity of these applications, we present the evaluation results normalized to non-persistent memory to demonstrate the strength of persistence.

Persistent memory systems achieve significant performance improvements in all evaluation metrics. Specifically, in terms of throughput, the persistent systems achieve almost two times improvements compared with the non-persistent ones. The memory space can be significantly saved in most applications. Furthermore, the typical operations decrease the latency in the persistent memory systems. The main reason comes from the fact that the persistence guarantees consistency, integrity and reliability, which essentially decrease system overheads and avoid extra costs, such as frequent data movements and CPU/memory instructions.

Shaping the Future

Big memory systems, leveraging technologies like persistent memory, large-scale DRAM, and disaggregated memory architectures, have transformed data-intensive computing by enabling faster access to vast datasets. However, significant challenges remain in ensuring efficient and reliable operations. Crash consistency and durability in big memory systems pose a key challenge, since balancing performance with correctness is non-trivial. Memory disaggregation introduces latency and contention issues, while security concerns such as persistent memory vulnerabilities and secure memory sharing in distributed setups require further exploration. Moreover, programmability is still difficult, since developers lack universal models for managing big memory systems, and debugging persistent applications is still complex. Addressing these challenges is crucial for advancing big memory adoption.

Beyond technical improvements, big memory systems unlock novel applications across various domains.

To overcome these challenges, hardware and software innovations are essential. Hardware advancements, such as CXL 3.0+ for improved memory pooling and near-memory processing to minimize data movement, could significantly boost performance. On the software side, persistent memory-aware file systems, lightweight checkpointing mechanisms, and machine learning-driven memory tiering could optimize data placement across DRAM and NVM. Distributed coherence protocols and hardware-accelerated RDMA persistence mechanisms are also needed to ensure efficient memory sharing in disaggregated environments. Multi-tier memory techniques are narrowing performance gaps and enabling scalable, cost-effective memory systems. CXL, in particular, seamlessly integrates persistent memory, GPUs, and accelerators, allowing direct memory access and improving system scalability.

Beyond technical improvements, big memory systems unlock novel applications across various domains. In AI and real-time analytics, they enable in-memory machine learning training and large-scale graph processing. Databases benefit from NVM-optimized designs that support instant recovery and high-speed transactions, while scientific computing leverages big memory for in-situ visualization and simulations. Edge and cloud computing can adopt memory-as-a-service (MaaS) models for dynamic resource allocation. Modern networking technologies are flattening memory and storage hierarchy, creating a seamless, low-latency memory fabric. This evolution enables byte-addressable persistence, replacing traditional storage hierarchies with scalable, network-spanning memory systems. As research progresses, the focus remains on hardware-software co-design, intelligent memory management, and expanding use cases in emerging fields, thus paving the way for transformative paradigms. C

References

1. Alshboul, M. et al. BBB: Simplifying persistent programming using battery-backed buffers. In *Proceedings of the IEEE Intern. Symp. on High-Performance Computer Architecture* (2021).
2. Chen, M. et al. A cost-efficient failure-tolerant scheme for distributed DNN training. In *Proceedings of IEEE Intern. Conf. on Computer Design* (2023).
3. Lee, S.K. et al. RECIPE: Converting concurrent DRAM indexes to persistent-memory indexes. In *Proceedings of the 27th ACM Symp. on Operating Systems Principles* (2019).
4. Li, H. et al. Pond: CXL-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM Intern. Conf. on Architectural Support for Programming Languages and Operating Systems* (2023).
5. Maruf, H. et al. TPP: Transparent page placement for CXL-enabled tiered-memory. In *Proceedings of the 28th ACM Intern. Conf. on Architectural Support for Programming Languages and Operating Systems* (2023).
6. Xu, J. and Swanson, S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In *Proceedings of the 14th USENIX Conf. on File and Storage Technologies* (2016).
7. Zhang, M. et al. Motor: Enabling multi-versioning for distributed transactions on disaggregated memory. In *Proceedings of USENIX Symp. on Operating Systems Design and Implementation* (2024).
8. Zhou, Y. et al. LightWSP: Whole-system persistence on the cheap. In *Proceedings of IEEE/ACM Symp. on Microarchitecture* (2024).
9. Zuo, P. et al. Write-optimized and high-performance hashing index scheme for persistent memory. In *Proceedings of USENIX Symp. on Operating Systems Design and Implementation* (2018).

Yu Hua (csyhua@hust.edu.cn) is a professor at Huazhong University of Science and Technology, Wuhan, Hubei, China.

Xue Liu (xueliu@cs.mcgill.ca) is a professor at McGill University, Montreal, Quebec, Canada.

Ion Stoica (istoica@berkeley.edu) is a professor at University of California, Berkeley, CA, USA.

Yu Hua's work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 62125202.