# HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM

Amanda Raybuck, Tim Stamler, Wei Zhang[1], Mattan Erez, Simon Peter[2]
University of Texas at Austin, [1]Microsoft, [2]University of Washington

## 1 INTRODUCTION

Systems with hybrid DRAM/non-volatile memory are now commercially available, such as with Intel's Optane DC non-volatile memory (NVM) modules that can share the memory bus with DRAM [1]. While NVM offers 8× higher capacity per module compared to DRAM, it comes at the cost of up to 7× lower bandwidth and twice the latency if not managed well. Tiered main memory management systems using hybrid DRAM/NVM need to balance these tradeoffs to provide efficient tiered memory to applications.

Lag in OS support for tiered memory (e.g., Linux so far has no official tiered memory support beyond swapping) has led hardware designers to provide tiered memory completely in hardware [3]. This approach has the benefit that it does not require OS support to manage tiered memories, but it has little visibility into the high-level requirements of applications using tiered memory and it must rely on simple memory tracking techniques that can be efficiently implemented in hardware. Research in OS-based tiered memory [4, 7] was evaluated with emulated NVM and does not capture the performance characteristics of commercially available NVM. Existing OS-based systems have overheads that prevent them from scaling to the capacity of commercially available NVM. Existing systems also do not support asymmetric read/write memory performance and have limited flexibility to adapt to the diverse needs of big data applications.

HeMem [6] is an user-level library memory management system that dynamically manages tiered memory without the CPU overhead of page access bit tracking, associated TLB shootdowns, and memory copies, but with advanced policy support for various memory access and allocation patterns. HeMem uses hardware performance counters to asynchronously sample memory access instructions to distinguish hot from cold memory pages for better scalability compared to traditional page access bit tracking approaches. HeMem supports asymmetric NVM bandwidth by prioritizing frequently written pages to DRAM. Finally, for flexibility, HeMem is implemented as a user-level library.

## 2 HEMEM DESIGN

HeMem relies on a number of techniques to efficiently manage tiered memory.

***Scalable memory access measurement via PEBS.***. To identify memory access patterns in a way that scales with memory size, HeMem uses a Processor Event-Based Sampling (PEBS) based approach instead of scanning page tables. We configure PEBS to measure DRAM loads, NVM loads, and all stores. HeMem samples memory access via a separate *PEBS thread* that continuously reads the PEBS buffer and updates page statistics when appropriate. When a sample is ready, HeMem examines the virtual address target of the sampled instruction and determines which huge page the address falls on. HeMem tracks and manages memory at huge page granularity.

The PEBS thread classifies data as hot or cold by organizing tracked pages into separate hot and cold lists for DRAM and NVM based on the PEBS samples. HeMem uses separate counters for reads and writes, as identified by the sample. A page is considered hot and placed in HeMem's DRAM or NVM hot list once a threshold number of 8 load or 4 store accesses are recorded to it, which we determined experimentally. A page that exceeds the store access threshold is considered a *write-heavy page*. To maintain freshness of HeMem's estimation of the hot set, HeMem will regularly cool pages by halving the access counts.

***NVM aware policies.*** HeMem allocates DRAM if available by removing a page from the DRAM free list. This allows ephemeral data to remain in fast memory. When running out of DRAM, HeMem simply allocates from NVM and relies on its PEBS thread to identify when pages in NVM become hot and should be migrated to DRAM. HeMem determines the size of memory ranges from intercepted memory allocation calls. Small allocations are forwarded to the Linux kernel to handle. Large allocations are handled by HeMem. In this way, small memory objects automatically remain in DRAM. NVM is accessed more efficiently at larger granularities, so keeping small ranges in DRAM yields better performance.

HeMem's migration policy scans the DRAM cold list and the NVM hot list provided by the PEBS thread and migrates pages among them. The policy thread runs every 10 ms. Write-heavy pages are given higher priority for migration to DRAM than read-heavy pages due to NVM having lower write than read performance.

***Efficient user-space mechanisms.*** HeMem manages tiered memory in a userspace library. Placing tiered memory management decisions in a library allows HeMem to efficiently obtain runtime-level information about memory use from the application with minimal overhead. HeMem handles page faults using userfaultfd [2]. When HeMem intercepts memory allocation calls, it registers the virtual address range with userfaultfd, allowing it to receive page and write-protection faults on this range. In the event of a page missing fault,

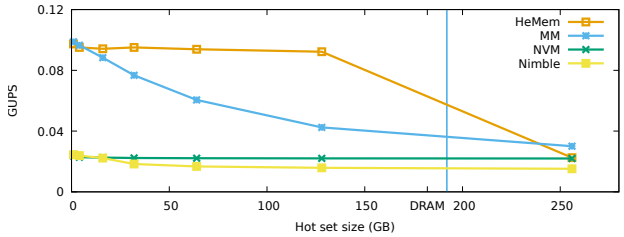


**Figure 1: GUPS with different hot set sizes (512 GB working set; higher is better).**

HeMem will map a zero-filled page from DRAM or NVM according to its policy at the faulting address and then wake the faulting application thread. For a write-protection fault, if the page is currently undergoing migration, HeMem simply waits until the migration is complete before waking the faulting thread.

When HeMem decides to migrate a page, it first uses userfaultfd to mark the page as write-protected. This allows reads to proceed to the page while under migration, but any writes to the page must wait until the migration is complete. Once the migration completes, page access rights are restored. If available, HeMem offloads the data migration to an I/OAT DMA engine [5], freeing the CPU of this task. If a DMA engine is not available, HeMem uses 4 additional threads to copy a page in parallel, akin to Nimble [7].

## 3 EVALUATION

HeMem is implemented in 4,177 lines of C code for the HeMem library as well as 1,337 lines of C code modified in the Linux kernel to add support for userfaultfd handling of DAX files and DMA. We run our evaluation on a single socket of a dual-socket Intel Cascade Lake-SP system running at 2.2GHz with 24 cores per socket and a 100 GbE ConnectX-5 Mellanox NIC. Each socket has 192 GB of DDR4 DRAM and 768 GB of Intel Optane DC NVM. To leverage all 6 memory channels, there are 6 DIMMs of DRAM and NVM per socket. The machine runs Debian 10.9 with Linux kernel version 5.1.0rc4. We compare HeMem to Intel Optane DC memory mode (hardware tiered memory management), as well as the Linux Nimble tiered memory management system [7].

We use GUPS as a microbenchmark to evaluate the behavior of the different tiered memory management systems. GUPS executes parallel read-modify-write operations to fixed size objects in its working set and measures the giga update operations per second (GUPS) it performs. Each thread has its own exclusive working set partition that it accesses without synchronization. We run GUPS with 16 threads, each performing 1 billion updates (16 billion updates in aggregate) to 8-byte objects. We modify GUPS to make a portion of each thread's objects *hot* (frequently accessed): 90% of the operations of each thread uniformly access its hot objects, while the remaining 10% of operations uniformly access the thread's entire working set. After a warm-up round, we run the benchmark 3 times and report the average GUPS in Figure 1.

As long as the hot set fits into DRAM, HeMem identifies it and ensures that it remains in DRAM. HeMem occasionally migrates cold data between DRAM and NVM, incurring minimal overhead. MM performance suffers as the GUPS hot set size approaches the capacity of DRAM. As the hot set grows, MM's direct-mapped caching approach exhibits more misses and more of the hot data is being pushed to NVM. HeMem performs up to 2× better with increasing hot set size. Nimble suffers from high overhead due to sequential scan and migration. Even when the hot set fits in DRAM, Nimble achieves only 25% of the GUPS of MM. When the hot set does not fit in DRAM, the performance of all configurations converges. HeMem identifies this case and stops migration.

## 4 CONCLUSION

HeMem is a software-based tiered memory management system designed from scratch for commercially available NVM. HeMem dynamically manages tiered memory without the CPU overhead of page access bit tracking, associated TLB shootdowns, and memory copies, but with advanced policy support for various memory access and allocation patterns.

## REFERENCES

[1] Intel Optane DC Persistent Memory, March 2019. http://www.intel.com/optanedcpersistentmemory.

[2] userfaultfd(2). http://man7.org/linux/man-pages/man2/userfaultfd.2.html, February 2020.

[3] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. Basic performance measurements of the Intel Optane DC Persistent Memory Module, April 2019. https://arxiv.org/abs/1903.05714v2.

[4] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. HeteroOS: OS design for heterogeneous memory management in datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17. Association for Computing Machinery, 2017.

[5] Thai Le, Jonathan Stern, and Stephen Briscoe. Fast memcpy with SPDK and Intel I/OAT DMA engine, April 2017. https://software.intel.com/en-us/articles/fast-memcpy-using-spdk-and-ioat-dma-engine.

[6] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. Hemem: Scalable tiered memory management for big data applications and real NVM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21. Association for Computing Machinery, 2021.

[7] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19. Association for Computing Machinery, 2019.