RETROSPECTIVE: HeteroOS: OS Design for Heterogeneous Memory Management in Datacenters

Sudarsun Kannan
Rutgers University

Ada Gavrilovska
Georgia Institute of Technology

I. Inspiration and Motivation

In 2015, Intel and Micron announced their new, jointly-developed 3D-XPoint persistent memory technology with the promise of disrupting the memory-storage hierarchy. This was preceded by Intel adding new instructions, CLWB and PCOMMIT, aimed at better using a future persistent memory component. The computer architecture community had long investigated storage class and persistent memories from the technology and integration perspective, with a smaller number of important works focused on the software support in terms of programmability and correctness. We joined this space to explore the question of how to bring this technology to the OS virtual memory level and developed pVM [O32]. As we pursued this work, we made two important observations: (1) Many of the workloads used in our evaluation of pVM benefited not just from the persistence of an additional memory component but even more so from the additional capacity. (2) The computer architecture community was working on many different types of memories, not just slower, persistent ones. This motivated us to ask more concretely, what it would take for an OS to be able to manage future heterogeneous memory systems. And since in virtualized datacenters and clouds, our OSes run within virtual machines, on top of hypervisors, it was natural to extend the question and wonder how this works in such settings, and what needs to be done across the OS/hypervisor systems software stack.

Our work on pVM positioned us at the right point of the design space. Unlike other approaches at the time, which looked to expose the different persistent properties of the new memories by carving out new persistent zones in the memory space, we took the approach that a persistent memory node should be exposed as a NUMA node. After all, NUMA-ness was already an aspect of (performance) heterogeneity, and there was a decades-long investment in the OS community to provide support to deal with this. Why reinvent the wheel? Obviously, just using existing NUMA mechanisms was not sufficient. Traditional NUMA-based approaches primarily focus on enhancing data locality by maximizing CPU access to data in the local memory socket. However, in the context of heterogeneous memory, the challenge lies in identifying performance-critical data and placing it in the fastest memory while maximizing the utilization of the limited capacity of the fast memory. Furthermore, pVM lacked dynamic placement or data migration across heterogeneous memory. Finally, it was completely unclear whether any support should be kept at the OS level alone, delegated to hypervisors, or somehow split. This motivated us to delve deeper into building HeteroOS.

II. Fighting the Hardware Restrictions

A major question was, how to pursue this research in the absence of real hardware prototypes of different types of memories. We observed that most state-of-the-art research relied on hardware instruction-level simulators. Simulators are important, but may take days to capture just 10 seconds of application runtime. Furthermore, using simulators to capture the overheads imposed by the OS and virtualization layers, such as hypercall overheads, hotness detection, page table walks, data migration costs, locking overheads, and other related factors, can be cumbersome and impractical.

We decided to try emulation using existing systems so as to be able to run a full OS/hypervisor stack. We could not make the current DRAM faster than it was, but we realized we could explore the possibility of throttling memory (like throttling CPU using DVFS) to make some DRAM nodes slower. Using the Intel System Software guide and discussions with various teams at Intel, we identified a hidden gem of throttling PCIe registers to reduce memory frequencies, eventually throttling memory bandwidth and latency. Through laborious effort, we identified that throttling worked only for a few Intel processor families; fortunately, we had just the right kind of Intel Nehalem dual-socket system with two memory nodes. We used the Linux extensions we developed for pVM as a starting point for the guest OSes, Vishal's software-based hotness tracking mechanism [O24], and we were ready to start this work.

III. What we learnt

Using an emulated heterogeneous memory system allowed us to run complex application benchmarks and a full systems software stack and to make observations that would have been difficult to make, at best, if not impossible, purely with simulation. We summarize here our key lessons learned from the design of HeteroOS, and their relevance in recent research.

Applications have different sensitivities: We provided insights into the impact of heterogeneous memories on different application types, beyond just well-studied graph processing and data analytics applications. Future evaluations with real heterogeneous memory systems using Intel's Optane memory confirmed these trends, making a case for general-purpose vs. only domain-specific memory management solutions.

Heap and I/O pages as first-class citizen: We were first to identify the importance of memory allocations and placement in heterogeneous memory systems extends beyond heap allocations to encompass other types of allocations, such as I/O caches. Recent papers from the industry, including Meta's work on heterogeneous memory, reinforce our findings [3].

Maximizing direct allocations to faster memory is critical: HeteroOS highlighted the significant benefits of prioritizing and maximizing on-demand memory direct allocation to faster memory over-relying on data migrations. A recent study from Google on far-memory systems emphasized the importance of direct allocations to fast memory, employing page compression techniques to reduce the usage of fast memory [1].

Software bottlenecks from hotness detection can surpass migration costs: HeteroOS identified that for memory-intensive applications, hotness tracking overheads could exceed data migration costs, and that, in virtualized environments, exclusively assigning hotness tracking and migration responsibilities to a hypervisor often resulted in stale or inactive data migration. Follow-on research has partially addressed these challenges by proposing parallel page migration, adding support for kernel page migration, and hardware support.

Support for resource sharing and multitenancy support not an afterthought: To achieve fairness and efficient resource allocation across different memory types, HeteroOS extended the Dominant Resource Fairness (DRF) algorithm to treat each memory type as a distinct resource. However, it is important to acknowledge that DRF [O19] may have limitations in handling variable resource demands. Further advancements are required to devise more sophisticated algorithms, and several recent works, including from Microsoft Azure, have explored managing heterogeneous memory for multitenant systems [2].

IV. ISCA 2017 Paper and Prior Attempts

We appreciate the feedback received from prior submissions and the ISCA 2017 reviewers. Despite not making a direct hardware architecture contribution, we are grateful for the ISCA reviewers' enthusiastic reception of our work. There are challenges in publishing purely systems software work that targets future hardware, as it can raise concerns among architecture reviewers regarding the lack of hardware-level contributions, and skepticism among systems software reviewers regarding the future availability of such hardware. This posed challenges in our experience with HeteroOS as well. Although the reviewers raised the concern that the work may be better suited for a systems venue, they observed that HeteroOS "targets a timely and important problem (multiple levels in the memory hierarchy) that will potentially become mainstream soon enough that we need to be lining up solutions."

The reviewers praised HeteroOS for its general principles in supporting virtualized and non-virtualized systems and its handling of anonymous and non-anonymous I/O pages. They specifically noted the advantages of cross-layer collaboration between the guest OS and hypervisor. By assigning memory management to the guest OS and utilizing direct hardware pagetable access for hotness detection by the hypervisor, we effectively mitigated the risk of the hypervisor becoming a bottleneck, particularly in multi-application scenarios. These valuable insights greatly contributed to the acceptance of the paper and the enhancements made in the camera-ready draft.

V. Where we are and Open Challenges

We briefly summarize the hardware and software progress since HeteroOS and highlight some open challenges.

Recent Hardware and Software Innovations: First, hardware innovations have focused on developing architectural support for heterogeneous memory technologies. Technologies like Compute Express Link (CXL) have emerged as industry standards, providing unified memory semantics and high-performance connectivity. These trends will continue to further advancements in managing heterogeneous memory.

Similarly, hardware support for memory profiling and hotness tracking with techniques like PEBS has evolved. However, limitations such as coarse profiling granularity and high sampling overhead persist. Recent studies have explored combining PEBS with performance counters to alleviate hotness detection bottlenecks, highlighting the need for hardware-software co-design.

Finally, several new applications- and runtime-guided data placement techniques have been proposed to overcome the limitations of OSes in prioritizing and differentiating between data objects within applications.

Open Challenges: Several open challenges persist.

System-level coordination involves coordinating hardware, operating systems, and applications to manage heterogeneous memory efficiently. Challenges include developing efficient communication mechanisms, ensuring compatibility with existing software ecosystems, and minimizing overheads associated with managing heterogeneous memory.

Secure memory sharing and virtualization among multiple virtual machines (VMs) or containers remains an open challenge. This includes designing secure memory-sharing mechanisms to prevent unauthorized access or data leakage across applications, VMs, or containers, while ensuring performance and efficient utilization of available memory resources.

Performance portability, code optimization, and debuggers are crucial for achieving performance across heterogeneous memory architectures. Designing efficient programming languages and tools involves exploring code optimization techniques, memory-aware compilers, and adaptive runtime systems. Effective debugging and profiling tools are essential for optimizing code and ensuring performance on diverse memory architectures. While hardware-specific implementations like CUDA support for HBMs exist, there is a need for more generic support to enhance compatibility and usability.

Finally, *managing new dimensions of memory heterogeneity,* such as for near-memory computing with fixed-function logic or (limited) programmability, is an exciting open challenge.

References

[1] A. Lagar-Cavilla, J. Ahn *et al.*, "Software-Defined Far Memory in Warehouse-Scale Computers," in *ASPLOS '19*, 2019.

[2] H. Li, D. Berger *et al.*, "Pond: CXL-Based Memory Pooling Systems for Cloud Platforms," in *ASPLOS'23*, 2023.

[3] J. Weiner, N. Agarwal *et al.*, "TMO: Transparent Memory Offloading in Datacenters," in *ASPLOS'22*, 2022.