Heterogeneous Memory Management in Datacenter

Sudarsun Kannan Ada Gavrilovska Vishal Gupta Karsten Schwan
UW-Madison, Georgia Tech, VMWare, Georgia Tech

ABSTRACT

Heterogeneous memory management combined with server virtualization in data-centers is expected to increase the software and OS management complexity. State-of-the-art solutions for both virtualized and non-virtualized systems rely exclusively on the expensive page migrations techniques, limiting the benefits from heterogeneity. To address this, we design HeteroOS, a novel application-transparent OS-level solution for managing memory heterogeneity in a virtualized system. Evaluation of HeteroOS with memory, storage, and network-intensive datacenter applications show up to 2x performance improvement compared to the state-of-the-art page migration techniques.

INTRODUCTION

To address the DRAM capacity scalability bottlenecks [1] and the need for lower access latency and higher bandwidth, researchers and commercial vendors are exploring alternative memory technologies such as 3D-stacked DRAM and non-volatile memory (NVM). These technologies differ significantly in latency, bandwidth, endurance; for instance, byte-addressable non-volatile memories (NVMs) such as phase change memory (PCM), are expected to offer higher capacity than DRAM, but with higher read (2x) and write (up to 5x) latency and lower bandwidth (5x-10x) [2]. Conversely, on-chip stacked 3D-DRAM [3] is expected to increase memory bandwidth by 8x-14x, but with a 2x-4x lower capacity than DRAM. These differences indicate that a single memory technology will not solve all memory-related bottlenecks for an application and that future systems must embrace memory heterogeneity, thereby, increasing software complexity.

While structuring these heterogeneous memories as NUMA node in OS seems natural [2], however, due to significantly different latency and bandwidth, NUMA's goal of achieving memory- and CPU-locality are insufficient. For heterogeneous memory, the challenge include: (a) identifying performance-critical data and placing them in the fastest memory, and (b) maximizing utilization of fast memory with limited capacity. To achieve these goals, prior heterogeneous memory management techniques [2] primarily rely on expensive hot page tracking and migration which adds substantial software overheads and are ineffective. To overcome these problems, we design and implement **HeteroOS** – a performance-efficient OS design for an application-transparent heterogeneous memory management in virtualized systems. The key idea of HeteroOS is to make the guest-OS of a virtualized datacenter memory heterogeneity-aware, and transparently capture applications' memory usage information to proactively allocate from the 'right' memory type without requiring page migrations. When direct allocation and placement from the guest-OS is not possible, HeteroOS enables a coordinated guest-VMM (hypervisor) management.

ANALYSES

To understand the implications of memory heterogeneity on applications, we analyzed several real-world datacenter applications, including memory-, CPU-, and I/O-intensive graph applications, network and memory-intensive key-value stores, and I/O-intensive NoSQL databases, on an in-house emulated (via memory-throttling) heterogeneous memory platform and on Intel's NVM emulator. We summarize below the observations from the analyses. More details of our study are available in the ISCA 2017 paper [4].

Observation 1. Applications show higher sensitivity to latency variations compared to bandwidth. We observe that memory-intensive applications (such as a graph and in-memory Map-Reduce) show higher sensitivity (slowdown) towards latency increase and bandwidth reduction compared to storage- (e.g., NoSQL DB) and network-intensive (e.g., key-value store) applications. Importantly, most applications (except graph applications) experience a significantly lower impact from bandwidth reduction compared to latency increase, mainly because they lack parallelism to saturate the bandwidth.

Observation 2. Incorrect memory placement cost in heterogeneous memory is significant (more than 5x) compared to traditional NUMA systems where incorrect placement to a remote memory impacts applications by less than 50%.

Observation 3. The placement of I/O (storage and network) pages in a heterogeneous memory system can significantly impact performance. Most prior work on heterogeneous memory mainly focus on placing/prioritizing application's heap memory to a faster memory. However, for I/O-intensive applications such as database and key-value stores, incorrect placement of the I/O page cache and buffer cache pages to slower memory significantly impacts performance.

Observation 4. Reactive hotness tracking and migration approaches induce significant software overhead. Most prior heterogenenous memory management techniques alway rely of page hotness tracking and migration techniques. However, software-based hotness tracking and migration can incur substantial overhead. Hotness tracking using page tables requires frequent TLB invalidation, whereas, migrations incur substantial page table traversal cost apart from data copy overheads.

DESIGN

Using the observations and the limitations of the state- of-the-art VMM-exclusive solutions, we briefly discuss the main design principles of HeteroOS for efficient and application-transparent management of heterogeneous memory.

Principle 1: Making guest-OS heterogeneity aware.

In contrast to current approaches which do not even expose NUMA information to the guest-OS, in HeteroOS, we expose the memory heterogeneity information to the guest-OS. As a result, a guest-OS can perform fine-grained direct page alloca-

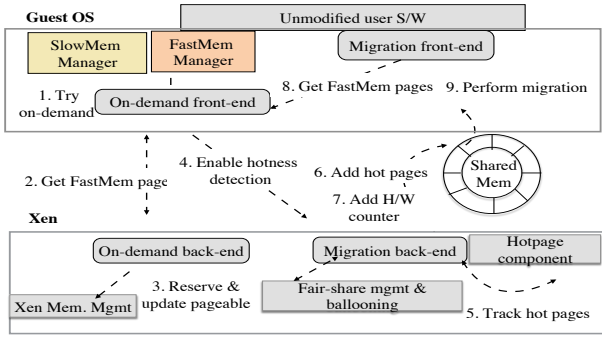

Figure 1: HeteroOS on-demand allocation, and coordinated management. In the figure, Steps 1-3 shows on-demand allocation, steps 4-9 shows hotness tracking and migration.

tion and data placement to different memory types and avoid frequent migrations. To expose memory heterogeneity to the guest-OS and enable memory type-specific allocations, we redesign and extend the OS-level page allocator, NUMA-related data structures, and the VMM drivers.

Principle 2: Capture page usage information to reduce migrations.

To reduce the overhead of page migrations across the heterogeneous memory, and provide on-demand allocation and direct placement, HeteroOS, using the guest-OS heterogeneity-awareness, captures how applications are using the different memory pages. In particular, HeteroOS monitors heap and kernel pages (I/O cache, kernel buffer) allocation ratios, and based on the demand, equally prioritizes their allocation to FastMem; this is against the traditional wisdom of always prioritizing the heap before I/O pages (e.g., disk swapping). To reduce the contention across heap and OS-level I/O pages, we also design a HeteroOS-LRU mechanism that aggressively evicts inactive FastMem heap and IO pages moving them to slower memory.

Principle 3. Supporting coordinated guestOS-VMM management to exploit VMM's holistic view

Direct page allocation and placement using the guest-OS is not always possible due to capacity limitations. Moreover, guest-OSes lack a holistic view of other VMs, and direct hardware control is required for privileged operations such as hot page tracking that must frequently invalidate the TLB for forcing access to a page table. To address this, we design a VMM-guest coordinated approach where the VMM performs hotness tracking, and the guest-OSes guide the VMM about what to track and when to track. The actual page migrations are performed in the guest OS. This is because, guest-OSes have application-level information and can avoid false migrations, for example, when pages are detected hot by VMM, but are released by an application or made inactive.

EVALUATION

We use a 16-core Intel Xeon 2.67 GHz dual socket system, with 16GB memory per socket. We emulate a fast memory (FastMem) and a slower memory (SlowMem) by applying DRAM thermal throttling to decrease the bandwidth by ~9x and increase the latency by ~5x based on the industrial projections [5, 6]. We evaluate HeteroOS with well-known Graph (GraphChi, X-Stream), Redis, and in-memory Map-Reduce Metis. More details on the heterogeneous memory emulation and other application results are described in the full paper [4].

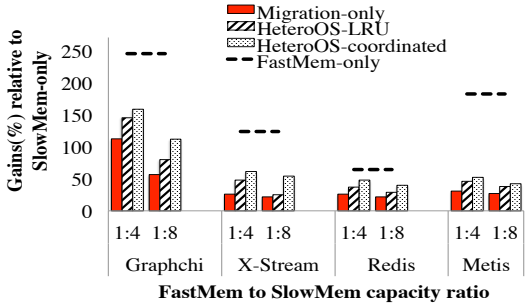

Figure 2: HeteroOS-LRU and HeteroOS-coordinated performance gains compared to the state-of-the-art migration-only approach.

In Figure 2, the X-axis shows the FastMem to SlowMem capacity ratio and the Y-axis shows the improvements compared to a naive approach of using SlowMem only. In our experiments, we set SlowMem memory size to 8GB per guest-VM, and vary the FastMem capacity ratio to 1GB and 2GB. We compare the following approaches: Migration-only corresponds to the state-of-the-art techniques, where hotness tracking and migration is done by the VMM transparent to the guest-OS. The HeteroOS-LRU represents a guest-OS-based approach that performs direct memory allocation and placement based on the demand for heap, and I/O pages. The HeteroOS-coordinated approach combines the on-demand HeteroOS-LRU approach with the VMM-level hotness tracking. Finally, the FastMem-only approach represents a hypothetical configuration where a system has sufficient FastMem.

First, the HeteroOS-LRU approach, with its on-demand guest-aware memory placement improves performance substantially compared to both the naive approach and the migration only approach. For instance, by reducing page migrations, the HeteroOS-LRU approach provides an average (across applications) of 69% gains relative to naive approach, and 20% over Migration-only for 1:4 FastMem to SlowMem ratio. However, the guestOS-level direct placement is insufficient for memory hungry applications. The VMM-guest coordinated approach combines the guest-OS level direct placement with the hotness tracking and migration approach to increase the average gains by 79.7% compared to naive placement, and 31% over Migration-only approach for 1:4 ratio (and up to 2x gains for GraphChi). More evaluation results can be found in our full paper [4].

1. REFERENCES

[1] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, pp. 20–24, Mar. 1995.

[2] S. R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data tiering in heterogeneous memory systems," in *EuroSys*, 2016.

[3] G. Loh and M. Hill, "Supporting very large dram caches with compound-access scheduling and missmap," *Micro, IEEE*, vol. 32, pp. 70–78, May 2012.

[4] S. Kannan, A. Gavrilovska, V. Gupta, and K. Schwan, "HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter," in *ISCA*, 2017.

[5] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die stacking (3d) microarchitecture," MICRO 39, 2006.

[6] "Intel-Micron Memory 3D XPoint." `http://intel.ly/1eICR0a`.