Fast Local Page-Tables for Virtualized NUMA Servers with *vMitosis* Extended Abstract

A. Panwar[1], R. Achermann[2], A. Basu[1], A. Bhattacharjee[3], K. Gopinath[1], J. Gandhi[4]
[1]*IISc Bangalore* [2]*ETH Zurich & University of British Columbia* [3]*Yale University* [4]*VMware Research*

Applications suffer non-uniform memory access (NUMA) latencies on modern multi-tier memory systems. As computer systems embrace extreme heterogeneity in the memory system, with innovations in die-stacked DRAM, high-bandwidth HBM, more socket counts, multi-chip module-based designs, the speed differences between local and remote memory continue to grow and become more complex to reason about [10, 13]. Placing, replicating, and migrating data among memory devices with variable latency and bandwidth is of paramount importance to the success of these technologies, and much work remains to be done on these topics. However, while there is at least prior work on data placement for application pages [2, 5, 7, 9, 12], kernel data structures have largely been ignored from this discussion. This is because kernel data has long been considered not to be crucial for performance – most kernel objects are pinned and unmovable in typical OS designs [11] – but we argue that this is no longer the case. This paper focuses on one critical kernel data structure, the page-table, and shows that virtualized NUMA servers must carefully reason about its placement for overall system performance.

Why focus on page-tables? Page-tables are vital to overall system performance. First, big-memory workloads require DRAM accesses on frequent page-table walks due to high TLB miss rates [3, 8, 4]. As system memory capacities grow to meet ever-increasing workload data demand, page-table growth is outstripping that of hardware TLBs. Larger address spaces require additional levels in page-tables (e.g., Intel's 5-level page-tables). TLB misses under virtualization are already expensive–a 2D page-table walk requires up to 24 memory accesses that will increase to 35 with 5-level page-tables. Finally, the page-table walk does not benefit from memory-level parallelism as they are an inherently serial process–each long DRAM access adds latency to address translation [4]. We show that misplacement of guest page-tables (gPT) and extended page-tables (ePT) is common in virtualized servers and explore techniques to improve their access locality.

Contribution 1: We provide an in-depth analysis of the misplacement of both levels of page-tables and its impact on page-table walks in virtualized NUMA systems. We partition several workloads into two groups: *Thin* workloads execute within a single NUMA socket (using up to 48 threads and 340GB of memory), and *Wide* workloads use the entire physical server (with 192 threads and up to 1.3TB of memory). We classify virtual machines (VMs) into two configurations: *NUMA-visible* mirrors the underlying server's NUMA topology for the guest OS, and *NUMA-oblivious* hides the NUMA topology from the guest OS. We summarize key observations from our analysis:

Observation 1: VM/workload migration has enduring performance implications on *Thin* workloads; while data pages migrate along with the thread, page-tables do not. The latency of remote page-table accesses is exacerbated by memory contention from other co-running workloads. We demonstrate this with a Memcached server instance that we migrate to a different socket during its execution. One would expect the server to fully recover its performance after the migration has been completed, i.e., when its data pages migrated to the new socket. However, we show that Memcached permanently loses as much as 50% performance post-migration. In the worst-case, migration can cause as much as a $3.1\times$ slowdown.

Observation 2: ePT becomes remote even in the absence of migration. This effect is observed when ePT is allocated by one virtual CPU (vCPU) but later used to translate addresses by another vCPU running on a different socket. Note that address translation on all vCPUs for a VM is performed with the same ePT. Consequently, even if the hypervisor honors local ePT placement, two workloads running in a VM will interfere with each other despite being partitioned using state-of-the-art CPU and memory binding techniques, due to a shared ePT.

Observation 3: Wide workloads that span multiple NUMA sockets are fundamentally susceptible to remote page walks–in a system with N NUMA sockets, each page-table entry (PTE) is local to one and remote to the other N-1 sockets. Assuming a random distribution of PTEs, the probability of a 2D page-table walk resulting in local DRAM access for both gPT and ePT is only $(1/N)^2$. Hence on our 4-socket system, we expect only about $(1/4)^2 = 1/16 \approx 6\%$ page-table walks to be served from local DRAM. Our offline analysis of page-table placement using page-table dumps and a software 2D page-table walker shows that virtualized page-table walks for most workloads closely resemble these expected local/remote access ratios. The fraction of local page-table walk access is much lower for NUMA-oblivious VMs as compared to NUMA-visible VMs since the guest OS is unable to optimize the placement of data structures in this case.

Observation-4: Large pages can reduce NUMA effects on address translation due to fewer TLB misses and shorter page-table walks. However, we still observe considerable NUMA effects for some important workloads (e.g., Redis). Further, large pages are responsible for many performance anomalies, including memory bloat (on our system, it leads to out-of-memory for some workloads, e.g., Memcached), latency, and OS jitters. Memory fragmentation also limits the OS's ability to allocate large pages over time. In conclusion, while important under ideal conditions, large pages cannot always be relied upon to eliminate NUMA effects on page-table walks.

Contribution 2: We design a system, *vMitosis*, that provides mechanisms and policies to mitigate NUMA effects on page-table walks for VMs. Our design relies on migration and replication of page-tables to ensure that TLB misses are always served locally. In *vMitosis*, under the NUMA-visible configuration, the hypervisor and the guest OS independently deploy NUMA optimizations for their page-tables. For the NUMA-oblivious configuration, we propose two techniques to enable gPT replication. Our first technique is based on para-virtualization in which the hypervisor communicates NUMA topology to the guest OS. Our second technique that is fully-virtualized implements gPT replication by reverse-engineering the physical server's NUMA topology in the guest OS.

Contribution 3: We provide an implementation of *vMitosis* in Linux/KVM. First, we integrate our gPT and ePT migration with the data-page migration mechanism (AutoNUMA [6]). Second, we implement replication of ePT for VMs spanning multiple NUMA sockets. And, to enable gPT replication in the NUMA-visible configuration, we leverage the open-source Mitosis implementation [1]. Third, we implement the two NUMA-oblivious techniques for gPT replication: *1)* a hypercall API that the guest uses to allocate and manage the gPT in co-operation with the hypervisor. *2)* a tool to reverse engineer the NUMA topology using online measurements of core-to-core communication latency. The discovered NUMA topology by *vMitosis* exactly mirrors the physical server topology. Using this tool, we can implement NUMA optimizations inside a NUMA-oblivious VM without any hypervisor support.

Closely Related Work: Our previous work, *Mitosis*, showed the importance of page-table placement on *bare-metal* NUMA machines [1]. We earlier showed how remote page-table walks impact performance and proposed replication of page-tables to mitigate NUMA effects on address translation for *native* execution. In contrast, *vMitosis* introduces incremental migration of page-tables instead of replication-based migration in *Mitosis* to have a robust design. Moreover, *vMitosis* introduces two new techniques to support NUMA-oblivious configurations (not supported by *Mitosis*) that serves as the basis of placing kernel data structures in heterogeneous memory systems. Table 1 shows that *vMitosis* is the first system to support NUMA optimizations for 2D page-tables in virtualized environments.

Challenges: Our design and implementation are non-trivial due to the following challenges. First, the NUMA topology required for replication is not always available under virtualization (e.g., in the NUMA-oblivious configuration). Second,

System	Guest Page-Tables		Extended Page-Tables	
	Migration	Replication	Migration	Replication
Linux/KVM	No	No	No	No
Mitosis	via Replication*	Yes*	No	No
vMitosis	Yes	Yes	Yes	Yes

Table 1: NUMA support for page-tables in current systems. (*) Replication is possible in Mitosis only if the server's NUMA topology is exposed to the guest OS.

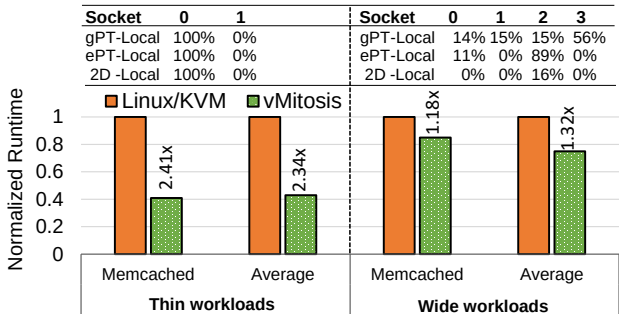

Figure 1: Normalized runtime with and without *vMitosis*. Table at the top represents the fraction of page-table entries local to different sockets involved in the execution of Memcached.

PTEs contain physical addresses of next level page-table or data pages that make a bit-by-bit copy (like data pages) infeasible and require the construction of a new page-table that points to the destination NUMA socket. Finally, the access and dirty bits updated by the hardware in both gPT and ePT must be consistent when multiple copies of the page-table exist.

Contribution 4: Finally, we present a comprehensive evaluation of *vMitosis* on a recent Intel Cascade Lake 4-socket NUMA server with 1.5TB RAM running Linux with KVM hypervisor as the baseline system. Figure 1 provides a summary of the results showing the normalized runtime of *vMitosis* over the baseline. We include the average over all workloads with Memcached as a representative workload. The table at the top shows the fraction of page-table walk accesses serviced from local DRAM for different sockets for Memcached. Under *Thin* workloads, threads run on socket-1 (post-migration) while gPT and ePT remain on socket-0. This leads to all-remote page-table accesses. In this case, *vMitosis* provides 2.41× speedup for Memcached and $1.8-3.1\times$ overall (2.34× on average). For the *Wide* workloads, threads running on all sockets experience remote DRAM accesses during a 2D page-table walk, only a small fraction are local. We obtain a 1.16× speedup for Memcached and $1.06-1.6\times$ overall (1.34× on average).

Why ASPLOS? *vMitosis* explores various migration and replication strategies for two-dimensional page-tables on virtualized NUMA servers to eliminate high latency page walk accesses. Therefore, our system design interacts with computer architecture, operating systems and virtualization. We believe it fits well in ASPLOS due to its multi-disciplinary research focus, and our discussions on the impact of kernel object's access latency will be interesting for the community.

Long Term Impact: We show that current OSes are missing NUMA optimizations for performance-critical dynamic kernel data-structures. The placement of frequently-accessed kernel data-structures in virtualized OSes, as demonstrated by page-tables, is important as technology trends continue to move towards heterogeneity and tiered memory systems. Our work is going to be released in a commercial hypervisor and would be open-sourced for Linux/KVM.

References

[1] Reto Achermann, Ashish Panwar, Abhishek Bhattacharjee, Timothy Roscoe, and Jayneel Gandhi. Mitosis: Transparently self-replicating page-tables for large-memory machines. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 283–300, Lausanne, Switzerland, 2020.

[2] Neha Agarwal and Thomas F. Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, page 631–644, Xi'an, China, 2017.

[3] Hanna Alam, Tianhao Zhang, Mattan Erez, and Yoav Etsion. Do-it-yourself virtual memory translation. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 457–468, Toronto, ON, Canada, 2017.

[4] Abhishek Bhattacharjee. Translation-Triggered Prefetching. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 63–76, Xi'an, China, 2017.

[5] Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. Black-box Concurrent Data Structures for NUMA Architectures. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 207–221, Xi'an, China, 2017.

[6] Jonathan Corbet. AutoNUMA: the other approach to NUMA scheduling. `https://lwn.net/Articles/488709/`, 2012.

[7] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 381–394, Houston, Texas, USA, 2013.

[8] Jayneel Gandhi, Mark D. Hill, and Michael M. Swift. Agile Paging: Exceeding the Best of Nested and Shadow Paging. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 707–718, Seoul, Republic of Korea, 2016.

[9] 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, page 521–534, Toronto, ON, Canada, 2017.

[10] Jeffrey C. Mogul, Eduardo Argollo, Mehul Shah, and Paolo Faraboschi. Operating system support for nvm+dram hybrid main memory. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09, page 14, Monte Verità, Switzerland, 2009.

[11] Ashish Panwar, Aravinda Prasad, and K. Gopinath. Making huge pages actually useful. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 679–692, Williamsburg, VA, USA, 2018.

[12] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11, page 85–95, Tucson, Arizona, USA, 2011.

[13] 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, page 331–345, Providence, RI, USA, 2019.