PTEMagnet: Fine-grained Physical Memory Reservation for Faster Page Walks in Public Clouds
Extended Abstract

Artemiy Margaritov, Dmitrii Ustiugov, Amna Shahab, and Boris Grot
University of Edinburgh

1. Motivation

Cloud computing offers great flexibility through on-demand resource scaling, high resource utilization and low operating costs. Businesses deploy their services in the public cloud in order to enjoy these benefits as reflected in the global cloud computing market growing from $350 billion in 2020 to an anticipated $800 billion by 2025 [6]. To ensure safety, isolation and to hide the complexity of managing physical machines, cloud resources operate under virtualization and rent virtual machines (VMs) to cloud users.

The applications that commonly run in the cloud, such as data analytics frameworks, key-value stores, and databases, operate on massive – and continually expanding – in-memory datasets. The large footprint of the datasets pushes beyond TLB reach and, together with irregular memory access patterns, reduces the efficacy of the processor TLBs, resulting in frequent TLB misses. Each TLB miss triggers a page walk – a long pointer chase through the page table (PT). When operating under virtualization, a page walk requires traversing both the guest and host PTs (i.e., a *nested* page walk) thereby incurring a particularly high latency as noted by prior works [5]. The nested page walk latency is further amplified when multiple applications are colocated within a single VM.

With rapid adoption of cloud computing for data-intensive tasks, the performance cost of address translation is destined to increase in the future. These observations highlight the need to reduce the page walk latency of big-data applications running in public clouds.

2. Limitations of the State of the Art

Prior attempts at accelerating address translation are either disruptive [3, 1, 9, 8] – requiring a radical re-engineering of the virtual memory subsystem – or incremental to the existing mechanisms [5]. While attractive from a performance perspective, the disruptive approach entails a major overhaul of the virtual memory subsystem, which presents an onerous path to adoption. Incremental approaches, such as TLB coalescing [7], enable greater TLB reach but are ultimately limited by latency and capacity constraints of TLB structures. Translation prefetching [5] uses software support and light-weight hardware to shorten the latency of traversing both the guest and host PTs. However, it requires architectural and microarchitectural modifications, which impedes rapid adoption.

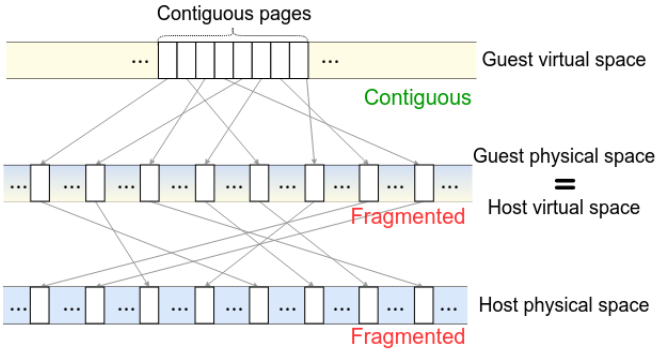

Figure 1: Contiguity (or lack of it) in virtual and physical address spaces under virtualization.

3. Key Insights

In this work, we investigate which accesses in a nested page walk are most significant for the overall latency by examining from where in the memory hierarchy these accesses are served. We observe that while accesses to the guest PT are often served by higher levels of the memory hierarchy (closer to the core), a significant fraction of accesses to host PT levels are served by the main memory, which results in long latencies in the nested page walk. We study the discrepancy in behaviour of accesses to the guest and host PTs and find that cache behaviour (hit or miss) of page walks is defined by spatial locality of page table entries (PTEs) that reside in the leaf PT level, which dominates the overall PT footprint. While guest PTEs corresponding to nearby virtual addresses reside in the same cache block, host PTEs corresponding to these guest virtual addresses are often scattered among multiple cache blocks.

Why are PTEs scattered? To understand why PTEs may reside in different cache blocks, let's consider a simple case where applications are running natively. A PT is indexed through virtual addresses, where the PTEs for two adjacent virtual pages A and A+1 sit in neighbouring leaf nodes and within the same cache block. While the A and A+1 are adjacent in virtual address space, their virtual-to-physical mappings are determined by the memory allocator. If the memory allocator is allocating memory for a single application, adjacent virtual pages are likely to be mapped to adjacent physical pages, carrying over their spatial locality to physical address space. However, if the memory allocator is allocating memory for multiple applications, the allocations for virtual pages A and A+1 may be interleaved by memory allocation requests for co-running application(s). In this case, A and A+1 are un-

likely to be mapped to adjacent physical pages and their spatial locality is lost in the physical address space. In the worst case, a set of pages that are contiguous in the virtual space may be allocated to physical pages that are entirely non-contiguous. This results in application's memory being fragmented in the physical address space.

Under virtualization, when the memory allocator in a VM is stressed by colocated applications, each individual application's memory is fragmented in the guest OS's physical address space. The host OS deals with the VM like another process and treats the guest physical address space as the VM process' virtual memory. Problematically, the fragmentation in the guest physical memory carries over to the host virtual memory (see Figure 1). Guest virtual pages A and A+1 which were mapped to non-adjacent guest physical pages will now be non-adjacent in host virtual address space. As a result, they will not occupy neighbouring PTEs in the host page table, and will not reside in the same cache block. This increases the footprint of the host page table nodes corresponding to each application running inside the VM.

Our main insight is that fragmentation of guest physical memory under virtualization and colocation inside the same VM can significantly increase page walk latency and degrade application performance.

4. Main Artifacts

We introduce PTEMagnet, a legacy-preserving software-only technique to reduce page walk latency in cloud environments by improving locality of host PTEs. Cache locality of host PTEs can be improved by limiting memory fragmentation in the guest OS. We show that prohibiting memory fragmentation within a small contiguous region greatly increases locality for host PTEs. PTEMagnet uses this observation and employs a custom guest OS memory allocator which prohibits fragmentation within small virtual address regions mapped to guest physical address space. PTEMagnet improves locality of the host PTEs and accelerates nested page walks.

Based on this insight, we propose PTEMagnet – a reservation-based approach to prevent fragmentation. To determine the optimal reservation granularity, we note that a 64B cache block can fit a maximum of 8 host PTEs, assuming an 8-bytes PTE, typical for x86. The 8 adjacent host PTEs represent a contiguous 8*4KB=32KB memory region in the VM's virtual memory and, in turn, the guest OS's physical memory. We find that by prohibiting fragmentation in each 32KB guest physical memory region, host PTEs can enjoy the benefits of maximum spatial locality from a cache block. PTEMagnet deploys a custom OS memory allocator that, on the first page fault to a given 32KB region, allocates the full 32KB (8 pages) but returns only 4KB (1 page) to an application, keeping the rest reserved for later use. On subsequent page faults within a reserved region, PTEMagnet's custom allocator instantly returns the already-reserved memory to the application.

Methodology We prototype PTEMagnet in Linux kernel v4.19 and evaluate its performance improvement on real hardware.[1] As a metric of performance, we directly measure application execution time. In addition, we quantify PTEMagnet's ability to reduce the number of page walk cycles. We study a set of diverse applications that are representative of those run in the cloud and that exhibit significant TLB pressure. We assume public cloud deployment (as with Amazon or Google VPCs [2, 4]) where multiple applications are scheduled on a fleet of VMs, and use QEMU/KVM for virtualized execution.

5. Key Results and Contributions

- We observe that under virtualization and colocation, page walks within the host PT incur 4.4× more cache misses than page walks within the guest PT.
- We show that the guest OS memory allocator, when operating under colocation, fragments the guest physical memory across the colocated applications. This results in host PTEs corresponding to each application being scattered over multiple cache blocks. For `pagerank` colocated with memory-intensive co-runners, the stressed guest OS memory allocator fragments over 63% of `pagerank`'s contiguous memory regions, scattering their host PTEs over many cache blocks.
- We propose PTEMagnet, a software-only technique that prevents scattering host PTEs across multiple cache blocks by prohibiting fragmentation within small memory regions using a reservation-based allocation approach.
- We demonstrate that PTEMagnet achieves 4% performance improvement on average (9% max) for big-memory applications sharing a VM with other workloads. Critically, applications that do not benefit from PTEMagnet are never slowed down by it, making PTEMagnet broadly attractive for cloud deployment.

6. Why ASPLOS

Our work focuses on the interaction of system-level technologies (address translation, virtualization and memory allocation) with hardware caches. The proposed mechanism, PTEMagnet, improves caching efficiency of host page tables entries through an OS memory allocator that enhances contiguity within small regions of physical memory.

7. Citation for Most Influential Paper Award

For showing that the OS memory allocator directly affects page walk latency, and for introducing an overhead-free OS memory allocator for accelerating page walks in public clouds.

[1] The code is available at https://github.com/amargaritov/PTEMagnet_AE

References

[1] Hanna Alam, Tianhao Zhang, Mattan Erez, and Yoav Etsion. Do-It-Yourself Virtual Memory Translation. In *Proceedings of the 44th International Symposium on Computer Architecture (ISCA)*, pages 457–468, 2017.

[2] Amazon. AWS virtual private cloud. Available at https://aws.amazon.com/vpc.

[3] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. Efficient virtual memory for big memory servers. In *Proceedings of the 40th International Symposium on Computer Architecture (ISCA)*, pages 237–248, 2013.

[4] Google Cloud. Virtual Private Cloud. Available at https://cloud.google.com/vpc.

[5] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion, and Boris Grot. Prefetched Address Translation. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1023–1036, 2019.

[6] MarketsandMarkets. Cloud Computing Market Report, 2020. Available at https://www.marketsandmarkets.com/Market-Reports/cloud-computing-market-234.html.

[7] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. CoLT: Coalesced Large-Reach TLBs. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 258–269, 2012.

[8] Dimitrios Skarlatos, Apostolos Kokolis, Tianyin Xu, and Josep Torrellas. Elastic cuckoo page tables: Rethinking virtual memory translation for parallelism. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1093–1108, 2020.

[9] Idan Yaniv and Dan Tsafrir. Hash, Don't Cache (the Page Table). In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 337–350, 2016.