Contiguitas: The Pursuit of Physical Memory Contiguity in Datacenters

Kaiyang Zhao, *Carnegie Mellon University, Pittsburgh, PA, 15213, USA*
Kaiwen Xue, *Carnegie Mellon University, Pittsburgh, PA, 15213, USA*
Ziqi Wang, *Carnegie Mellon University, Pittsburgh, PA, 15213, USA*
Dan Schatzberg, *Meta Platforms Inc., Menlo Park, CA, 94025, USA*
Leon Yang, *Meta Platforms Inc., Menlo Park, CA, 94025, USA*
Antonis Manousis, *Meta Platforms Inc., Menlo Park, CA, 94025, USA*
Johannes Weiner, *Meta Platforms Inc., Menlo Park, CA, 94025, USA*
Rik Van Riel, *Meta Platforms Inc., Menlo Park, CA, 94025, USA*
Bikash Sharma, *Meta Platforms Inc., Menlo Park, CA, 94025, USA*
Chunqiang Tang, *Meta Platforms Inc., Menlo Park, CA, 94025, USA*
Dimitrios Skarlatos, *Carnegie Mellon University, Pittsburgh, PA, 15213, USA*

Abstract—The unabating growth of the memory needs of emerging datacenter applications has exacerbated the scalability bottleneck of virtual memory. However, reducing the overhead of address translation will remain onerous until the physical memory contiguity predicament gets resolved. To address this problem, Contiguitas provides ample physical memory contiguity by design. We identify that the primary cause of memory fragmentation in Meta's datacenters is unmovable allocations scattered across the address space that impede contiguity. To this end, Contiguitas in the operating system separates movable allocations from unmovable ones by placing them into two different dynamically adjustable regions in physical memory. Furthermore, Contiguitas drastically reduces unmovable allocations through hardware extensions that transparently migrate unmovable pages while they remain in use. Our experiments in production at Meta's datacenters show that Contiguitas achieves end-to-end performance improvements of 2-18%. Full-system simulations of the Contiguitas hardware show that it can efficiently migrate unmovable allocations without affecting applications.

Introduction

Memory capacity has increased dramatically over the last few decades, yet modern operating systems have throughout stuck with a small base page size. This divergence has resulted in excessive management overhead for memory-intensive applications. In particular, virtual memory implementations are plagued by expansive page table trees, and a corresponding appetite for hardware TLB capacity that is difficult to satiate. Even with hardware innovations such as larger and multi-level Translation Lookaside Buffers (TLB), applications today suffer a substantial performance penalty due to TLB misses. Case in point, as shown in Figure 1, servers at Meta have their memory capacity increased by almost 8× over a few hardware generations. However, the number

XXXX-XXX © 2024 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

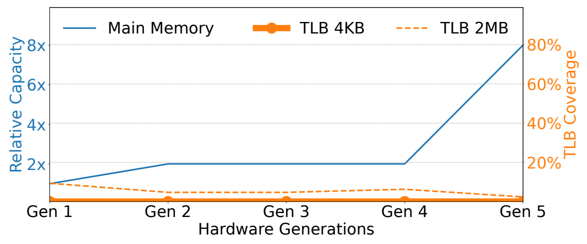

FIGURE 1. Memory and TLB coverage of computing hardware across generations.

of TLB entries remains stagnant, leading to minuscule and shrinking TLB coverage. Google's profiling further revealed that approximately 20% of cycles are stalled on TLB misses [1]. Unfortunately, this problem is only bound to get worse due to: i) the inherent hardware limits of TLB scaling, already surpassing L2 cache latencies, ii) terabyte-scale memory capacity through technologies like Compute Express Link (CXL), iii) additional levels of page tables, iv) the increase of memory-intensive applications, and v) upcoming confidential computing platforms that place security checks at page granularity during address translation.

A large body of prior research has focused on reducing the address translation overhead. Conceptually, we can separate prior work based on the amount of physical memory contiguity required and how it is exploited. On the one hand, earlier works propose leveraging physical memory contiguity to back the application dataset [2]. These approaches create range-based translations by relying on large contiguous physical memory. However, they face the fundamental challenge that it is very hard to create multi-gigabyte contiguous physical address ranges to cover the complete application dataset. On the other hand, another line of research has explored alternative page table structures such as hashed page tables [3]. Such solutions aim to replace sequential multi-level page tables and drastically reduce the cost of page walks by accelerating page table accesses. Notably, they relax the physical memory contiguity requirements to apply not on the whole dataset, but only on the page table organization. However, they impose strict requirements for physical memory contiguity availability on the critical path of page table creation. Several other architectural extensions that implicitly rely on contiguity [4], [5] are hindered by the same fundamental challenge.

Today's operating systems (OS), such as Linux, have mostly relied on 2 MB Transparent Huge Pages (THP) that *opportunistically* provide 2 MB pages to land performance improvements. Unfortunately in today's systems, finding physical contiguity even for 2 MB pages is often hard due to memory fragmentation [6], [7]. THPs have also been under scrutiny due to performance implications such as latency spikes and memory bloating. Alternative approaches, such as userspace allocators, still rely on the OS to provide physical contiguity and larger mappings.

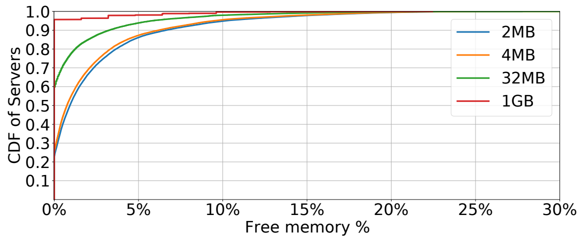

FIGURE 2. Contiguity availability as the percentage of free memory.

In this work, we start with a detailed investigation of physical memory contiguity at hyperscale across Meta's datacenters. We sampled servers across the fleet and the memory fragmentation distribution is shown in Figure 2. *23% of servers do not even have physical memory contiguity for a single 2MB huge page*. We also find that it is practically impossible to dynamically allocate 1 GB pages in production. Furthermore, our analysis shows that there is little to no correlation between memory contiguity and server uptime, with the Pearson correlation coefficient between server uptime and the number of free 2 MB pages being only 0.00286. Pertinently, this means that fragmentation affects all servers. In practice, servers can quickly get heavily fragmented within the first hour after boot-up while the mean server uptime is days or weeks—turning memory fragmentation into a major challenge. Finally, our study exposes unmovable memory allocations as the root cause for the lack of physical memory contiguity. In particular, we identify several sources of unmovable allocations, including networking buffers, slab, filesystems, and page tables.

To address these issues, we introduce Contiguitas with the goal of eliminating fragmentation due to unmovable allocations. Contiguitas separates movable allocations from unmovable ones by placing them into two different regions and dynamically adjusting the boundary of the two regions on demand. To avoid wasting memory in the unmovable region, Contiguitas solves two problems: i) how to dynamically resize the unmovable region and place unmovable allocations;

and ii) how to drastically reduce unmovable allocations. For the first problem, Contiguitas performs resizing by tracking the demand for unmovable allocations. Moreover, it reduces internal fragmentation of the unmovable region by differentiating types of allocations.

For the second problem, Contiguitas focuses on unmovable pages that cannot be moved with software alone because access to such pages cannot be blocked for a migration to take place. At Meta, networking allocations account for 73% of unmovable pages. We expect unmovable pages to become an increasingly bigger problem because of new input/output (I/O) technologies such as kernel-bypass and Remote Direct Memory Access (RDMA) for networking and storage, Graphics Processing Units (GPU), and other accelerators that heavily really on unmovable pages.

To this end, Contiguitas introduces a set of surgical hardware extensions in the last-level cache (LLC) that enable the *transparent* migration of unmovable pages while in use. Contiguitas's design builds off of two ideas: First, Contiguitas introduces migration mappings in the LLC, enabling hardware to redirect traffic to the appropriate cacheline of each page based on the progress of the migration. Second, Contiguitas relaxes the TLB shootdown operation from being synchronous and requiring acknowledgements from all victim TLBs to a local TLB invalidation that can be performed by each core independently and in a lazy manner. Naturally, movable page migrations can also benefit from this hardware support.

Our experiments in Meta's production datacenters show that Contiguitas successfully confines unmovable allocations, leading to significant performance gains. Full-system simulations showcase the effectiveness of Contiguitas's hardware. We are currently in the process of upstreaming Contiguitas into Linux.

Contiguitas Design

The goal of Contiguitas is to provide ample physical memory contiguity by reducing memory fragmentation due to unmovable allocations. To that end, Contiguitas redesigns memory management in the OS to confine unmovable allocations and completely separate them from movable ones. In addition, Contiguitas drastically reduces unmovable pages in datacenters. Specifically, Contiguitas introduces a set of hardware extensions in the LLC that enable the transparent migration of unmovable pages while in use.

Contiguitas Overview.

Figure 3 provides a high level overview of Contiguitas. First, Contiguitas redesigns memory management in the OS to confine unmovable allocations and completely separate them from movable ones (Step ①), preventing unmovable allocations from scattering across the address space. Then, Contiguitas dynamically resizes regions in response to memory pressure (Step ②). Finally, Contiguitas drastically reduces unmovable pages through hardware extensions in the LLC that enable transparent migration of unmovable pages while in use (Step ③). Naturally, these hardware extensions can further be leveraged by movable pages.

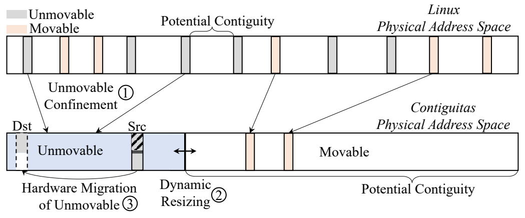

FIGURE 3. Contiguitas design overview.

Unmovable Confinement.

The first design principle of Contiguitas is to strictly separate unmovable from movable allocations using two dedicated regions, movable and unmovable regions in the physical address space. Allocations are confined in their respective region. Contiguitas categorizes the physical pages based on their addresses and keeps them on distinct free lists for each region. Memory in each region can only be allocated from pages in the free lists belonging to that region. When a page is freed, it is returned back to its respective list. This approach simplifies the critical path of allocations as the OS can quickly pick a free page while avoiding mixing different types of allocations. For allocations that are first allocated as movable but later become unmovable, Contiguitas migrates them to the unmovable region and marks them as unmovable. This approach avoids the dynamic pollution of the movable region and subsequent compaction failures.

The crucial part in designing confinement is the sizing of the unmovable region. If it is too big, unused memory in the unmovable region is wasted while there is limited movable memory for the applications, causing frequent reclaims, swapping, or even allocation failures. On the other hand, if the unmovable region is too small, it may fail unmovable allocations. Therefore, Contiguitas dynamically balances the sizes of the movable and unmovable regions while not negatively affecting application performance.

There are three major challenge in dynamic resizing of the unmovable region. The first challenge is to move resizing operations off the critical path of memory allocation. Contiguitas performs resizing off the critical path of memory allocation to avoid latency overheads. This is accomplished by monitoring the amount of free memory when periodic memory reclaim is triggered by the kernel. Contiguitas extends reclaim to wake up a kernel thread to perform resizing when the free memory in either region falls below a low-watermark threshold.

The second challenge is to decide when and how much to resize. Contiguitas introduces the concept of per-region memory pressure and extends Pressure Stall Information (PSI) [8] to track time wasted due to lack of free memory in the movable and unmovable region separately, which is then used to calculate the expansion/shrinking amount influenced by tunable parameters.

The third challenge is to ease resizing and reduce data movement. Contiguitas introduces a bias to prefer physical pages further away from the region border. Some unmovable allocations that are inherently long lived, e.g., kernel code pages, are safely placed by Contiguitas early on, at the end of the unmovable region that is farthest from the movable region. On the other hand, pages that are initially in the movable region and later on migrated to the unmovable region often exhibit short lifetimes. In general, Contiguitas prefers allocating pages away from the region border as long as sufficient free space is available. This approach increases the chance that resizing will be successful and need little data movement.

Hardware Migration of Unmovables.

The second design principle guiding Contiguitas is to drastically reduce the amount of unmovable allocations by turning them into movable ones. Our study at Meta's datacenters revealed that a significant portion of unmovable allocations used for input/output (I/O) are impossible to move as access to the page cannot be blocked for a software migration to take place.

Hardware support is required because it is impossible for software to move such pages as it cannot atomically perform both the translation update and the page copy operation. Hence, software has to block access to the page for the duration of page migration in order to avoid spurious writes to it. Even if access to the page could be blocked, software page migration induces a long downtime due to (a) TLB shootdowns that scale poorly with the number of victim TLBs, and (b) the page copy.

To this end, Contiguitas-HW enables transparent

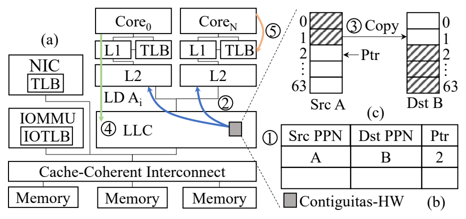

FIGURE 4. Contiguitas hardware overview.

page migration while the page remains in use. Such migrations can substantially reduce the size of the unmovable region and lead to more efficient defragmentation and memory management as the vast majority of pages can be moved on demand. While Contiguitas-HW is motivated by unmovable allocations, its design is suitable for both movable and unmovable allocations.

The hardware extensions of Contiguitas-HW are located in the LLC as shown in Figure 4. Contiguitas-HW targets a multi-core processor with a cache-coherent interconnect. The hardware platform further includes an Input-Output Memory Management Unit (IOMMU) with local TLBs.

At a high level, Contiguitas-HW aliases a physical page under migration with a destination page and redirects appropriate traffic to the destination page based on the progress of the migration. Specifically, a page migration is initiated in Step ① by the OS that provides the source and destination physical page numbers (PPN) to the Contiguitas-HW. Contiguitas-HW stores them in a metadata table shown in Figure 4(b) along a *Ptr* field that points to the next line to be copied [3], and enables access redirection. The OS then modifies the page table entry to point to the destination page and starts local TLB invalidations, shown in Step ⑤. In Contiguitas-HW, TLB shootdowns do not require inter-processor interrupts (IPI), as both mappings are concurrently active during the migration.

During this process, a page may be accessed with either the source or the destination mapping. If a request hits in the private caches, it is serviced normally as in a regular cache hit. As shown in ④, on a miss the Contiguitas-HW checks whether a line is currently stored in private caches with the opposite mapping of the request i.e., if a request is for the source mapping and the line is stored with the destination mapping, and vice versa. If so, it invalidates any cached copy. Otherwise, the request is serviced regularly. This invariant allows the caching of lines under migration as only the source or the destination mapping is active in the private caches. When the TLB invalidations are complete, the OS notifies the Contiguitas-HW to start

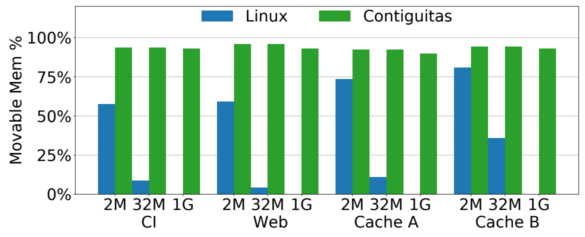

FIGURE 5. Potential memory contiguity as a percentage of total memory.

the copy. At this point only the destination mapping is active at any given TLB. Contiguitas-HW copies a cacheline by bringing it into the LLC (Step ②) and copying it from the source to the destination (Step ③). Finally, it increments *Ptr*. This process continues until the page is copied and Contiguitas-HW notifies the OS. The full paper discusses how Contiguitas-HW can alternatively be implemented through noncacheable memory accesses and how it supports sliced LLCs.

Results: Ample Contiguity in Datacenters

We build the OS component of Contiguitas into Linux and run our experiments in Meta's production datacenters with four production workloads: a continuous-integration service (CI), a web server (Web), and two caching services (Cache A and Cache B).

Potential Memory Contiguity

To quantify the impact of Contiguitas on memory contiguity, we compare each workload's steady state under Linux and Contiguitas. Specifically, we quantify the contiguous regions that can be formed if we *hypothetically* run a perfect software compaction in order to service allocation requests of 2 MB, 32 MB, and 1 GB. Figure 5 shows the results. With Linux we see that some 2 MB allocation are possible given that less than half the memory is composed of unmovable 2MB pages as we discussed above. However, Linux struggles as we search for larger contiguous regions, and fails to find even a single 1 GB page. On the other hand, Contiguitas, by design, isolates the unmovable region and hence the whole movable region can potentially be used after compaction for large contiguous allocations, even 1 GB pages.

End-to-end Performance

To measure Contiguitas's improvements to end-to-end performance due to increased memory contiguity, we use requests per second under certain latency SLAs based on the characteristics of each workload. We consider two setups, *Full Fragmentation* and *Partial Fragmentation*. *Full Fragmentation* represents the case where a workload lands on a server whose memory is already fully fragmented. *Partial Fragmentation* represents the case where a workload lands on a partially fragmented server that is representative of the majority of servers at Meta. Figure 6 shows the results. Contiguitas achieves performance improvements between 2-9% for partially fragmented servers that represent the majority of the servers, and between 7-18% for highly fragmented servers that represent nearly a quarter of Meta's fleet. Notably, Contiguitas's contiguity gains enable Web, one of Meta's largest services, to dynamically allocate 1 GB huge pages, leading to a 7.5% performance win that is unattainable with 2 MB pages alone. We are currently in the process of upstreaming the operating system component of Contiguitas into Linux [9].

Hardware Evaluation

We use full-system simulations to show that Contiguitas-HW efficiently migrates unmovable allocations without affecting application performance. We consider two open source applications Memcached and NGINX to cover applications that do and do not benefit from huge page availability. Even at a rate of 1000 pages migrated per second, which would be unwarranted for a real environment, Contiguitas-HW does not have an impact on both applications' performance. When combined with the benefits of contiguity and 2MB huge pages, Memcached performance improves by 7%.

Furthermore, Contiguitas-HW scales well with the number of TLBs, keeping the page unavailable time during a page migration constant and equal to a local TLB invalidation, whereas under status quo the page unavailable time increases linearly.

Overall, Contiguitas-HW does not negatively impact applications that do not benefit from contiguity while improving contiguity for those that do.

Impact and Future Research Direction

Contiguitas is a holistic solution that addresses the long-standing problem of memory contiguity in datacenters. Contiguitas is already having a major impact

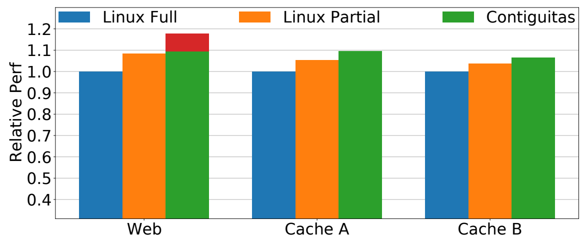

FIGURE 6. End-to-end performance over Meta's production workloads. The red bar of Contiguitas for Web shows the performance gains from 1GB pages.

as it is in the process of being upstreamed in the Linux kernel, and Meta is actively working on deploying it in production.

A Holistic Solution for Memory Contiguity in Datacenters.

Memory contiguity is a fundamental requirement for a vast body of work that aims to reduce the cost of address translation. However, we identify that memory contiguity is scarce resource in production datacenters. Contiguitas is the first solution that resolves this problem and provides ample memory contiguity by design. Contiguitas addresses the challenges of fragmentation due to unmovable allocations that are prevalent in datacenter and will only get worse due to RDMA, GPUs, Network Interface Controllers (NIC), accelerators, CXL, and devices that rely on unmovable pinned memory. Contiguitas introduces a future-proof OS memory management design that defragments the physical address space by isolating unmovable allocations. Finally, Contiguitas drastically reduces unmovable pages through hardware extensions that enable the transparent migration of unmovable and movable pages while in use.

Understanding Memory Fragmentation, TLB, and Page Walk Characteristics in Datacenter Applications.

Contiguitas is the first detailed study of memory fragmentation and unmovable pages in datacenters. It further sheds light into the limited TLB coverage and the cost of address translation of data and instructions. Contiguitas shows that page walk cycles can account for close to 20% of total cycles in Meta's datacenters for major workload classes, such as Web serving, key-value stores, Ads, and graph data stores. Importantly, our study reveals that almost a quarter of the servers do not have enough contiguity for even a single 2MB allocation. It further identifies that there is little correlation between fragmentation and server uptime, leading to heavily fragmented servers across the fleet within the first hour of uptime. Finally, we present the causes of fragmentation and especially unmovable allocations, 73% of which originate from networking. The results will guide future research to take into account the memory and address translation characteristics of datacenter workloads.

Major Performance Gains for Production Workloads.

Contiguitas' ample memory contiguity enables the use of huge pages of 2MB and even 1GB in production at Meta's datacenters. Our experiments with real traffic across major workload classes demonstrate that Contiguitas can provide significant performance gains between 2-18%.

Upstreaming into Linux and Industry Impact.

We are upstreaming Contiguitas into the Linux kernel. The first set of patches has already been submitted to the kernel mailing list [9] and has been well-received. We are refining remaining patches that will bring additional Contiguitas features to the Linux kernel. After these steps are complete, Meta will deploy Contiguitas in production to materialize its significant benefits at scale.

Memory folios with Contiguitas.

Memory folios [10] is a promising solution to support larger page sizes in the Linux kernel. Folios have the potential to help reduce the large overhead of managing memory in base 4KB pages. A primary requirement for folios is contiguity for larger allocations. Contiguitas makes folios practical by providing the needed contiguity. Furthermore, the combination of Contiguitas with folios is a promising research direction that will help reduce the major cost of memory movement due to defragmentation.

Large Block Sizes Support in SSDs.

Large Block Sizes (LBS) [11] is a promising solution to improve I/O performance by leveraging the increased capabilities of modern flash storage devices with larger mappings. Existing solutions rely on 4KB mappings, despite the substantial benefits of larger mappings such as 16KB or larger sizes. Contiguitas provides the necessary contiguity for such solutions to be integrated into the kernel and improve I/O and page cache performance.

Contiguitas Enables New Research Directions in OS and Computer Architecture.

Contiguitas enables future research on how to resolve the address translation cost. Contiguitas provides the contiguity required by techniques such as huge pages, TLB compaction, larger translation ranges, prefetching, alternative virtual memory designs, and hashed page tables. With ample memory contiguity, Contiguitas enables a large space for research explorations of improved TLBs coverage, page walk latency reduction, and memory management.

REFERENCES

1. A. Hunter, C. Kennelly, P. Turner, D. Gove, T. Moseley, and P. Ranganathan, "Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021.
2. A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient Virtual Memory for Big Memory Servers," in *40th International Symposium on Computer Architecture (ISCA)*, 2013.
3. D. Skarlatos, A. Kokolis, T. Xu, and J. Torrellas, "Elastic cuckoo page tables: Rethinking virtual memory translation for parallelism," in *25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.
4. B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "CoLT: Coalesced Large-Reach TLBs," in *45th International Symposium on Microarchitecture (MICRO)*, 2012.
5. S. Gupta, A. Bhattacharyya, Y. Oh, A. Bhattacharjee, B. Falsafi, and M. Payer, "Rebooting virtual memory with midgard," in *48th International Symposium on Computer Architecture (ISCA)*, 2021.
6. Y. Kwon, H. Yu, S. Peter, C. J. Rossbach, and E. Witchel, "Coordinated and Efficient Huge Page Management with Ingens," in *12th Conference on Operating Systems Design and Implementation (OSDI)*, 2016.
7. A. Panwar, A. Prasad, and K. Gopinath, "Making Huge Pages Actually Useful," in *23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.
8. J. Weiner, N. Agarwal, D. Schatzberg, L. Yang, H. Wang, B. Sanouillet, B. Sharma, T. Heo, M. Jain, C. Tang, and D. Skarlatos, "TMO: Transparent memory offloading in datacenters," in *27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.
9. "A Reliable Huge Page Allocator," https://lore.kernel.org/lkml/20230418191313.268131-1-hannes@cmpxchg.org/, 2023.
10. "The State of The Page in 2023," https://lwn.net/Articles/931794/, 2023.
11. "Large Block Size," https://kernelnewbies.org/KernelProjects/large-block-size, 2020.

Kaiyang Zhao Kaiyang is a PhD student at Carnegie Mellon University and can be reached at kaiyang2@cs.cmu.edu.

Kaiwen Xue Kaiwen was a Master's student at Carnegie Mellon University and can be reached at kaiwenx@andrew.cmu.edu.

Ziqi Wang Ziqi was a PhD student at Carnegie Mellon University and can be reached at ziqiw@andrew.cmu.edu.

Dan Schatzberg Dan is a research scientist at Meta and can be reached at dschatzberg@meta.com.

Leon Yang Leon is a software engineer at Meta and can be reached at lnyng@meta.com.

Antonis Manousis Antonis is a research scientist at Meta and can be reached at amanousis@meta.com.

Johannes Weiner Johannes is a kernel engineer at Meta and can be reached at jweiner@meta.com.

Rik van Riel Rik is a kernel engineer at Meta and can be reached at riel@meta.com.

Bikash Sharma Bikash is a performance and capacity engineer at Meta and can be reached at bsharma@meta.com.

Chunqiang Tang Chunqiang is a senior director at Meta and can be reached at tang@meta.com.

Dimitrios Skarlatos Dimitrios is an assistant professor of computer science at Carnegie Mellon University and can be reached at dskarlat@cs.cmu.edu.