KLOCs: Kernel-Level Object Contexts for Heterogeneous Memory Systems

Sudarsun Kannan
Rutgers University

Yujie Ren
Rutgers University

Abhishek Bhattacharjee
Yale University

1. Introduction

Emerging computer systems combine the best properties of memory technologies optimized for latency, bandwidth, capacity, persistence and offer better performance, energy-efficiency, and cost trade-offs [3, 10, 11, 13, 16, 19]. However, prior research has demonstrated the challenge of data allocation and migration in multi-socket non-uniform memory access (NUMA) architectures [4,5,7,15,17,21]. Heterogeneous memory systems amplify this challenge by integrating memory devices with even more varied latency and bandwidth characteristics. Consequently OS-directed data tiering for heterogeneous memories has become an active research area [9, 12, 17, 23]. Unfortunately, most prior studies focus on application-level data, ignoring kernel objects. We show that modern I/O-intensive applications, driven by advances in networking and storage speeds, require significant memory capacities for kernel objects. Modern OSes, designed for the days when kernel object memory usage was relatively low, leave as much as 4× performance on the table due to poor tiering decisions.

Challenges. OS-controlled kernel object tiering requires solutions to: (1) the absence of OS abstractions/mechanisms to efficiently group kernel objects, (2) orders of magnitude shorter lifetimes of kernel objects compared to application pages, and (3) lack of support for kernel object migration.

Proposed Approach. To overcome these challenges, we introduce a new OS abstraction, *kernel-level object contexts (KLOCs)*, that permits fluid tiering of kernel objects by introducing a novel mechanism for grouping related kernel objects, efficiently identifying their hotness and coldness, and enabling their low-overhead migration when desirable.

Kernel Object Grouping. To solve the challenge of grouping kernel objects, KLOCs capture groups of kernel objects associated with OS entities requested by applications. The kernel entities requested by applications are files and sockets, while the kernel objects range from structures associated with files (e.g., inodes, blocks, extents, etc.) to those associated with sockets (e.g., packet buffers, headers, data buffers, etc.). Our proposed abstraction exploits the fundamental Unix-based "everything is a file" abstraction and maintains a KLOC for each file and socket entity, and groups all kernel objects allocated and used to serve requests to these entities.

Kernel Object Hotness Tracking and Migration. KLOCs offer a principled way to tame this diverse ecosystem of kernel objects and enable quick determination of their hotness/coldness instead of relying on expensive hotness scan across the entire address space. When the OS determines that an inode has become cold (because, for example, the file or socket associated with the inode has been closed), the data structures used to implement the KLOC abstraction permit direct identification of all kernel objects associated with the inode and mark them as candidates for migration to slow memory.

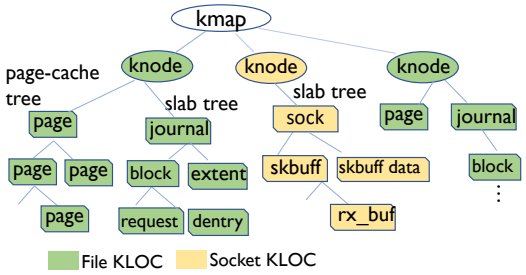

Figure 1: All of the kernel objects associated with each active file and active socket represent individual KLOCs. All the KLOCs in the system are tracked using a kmap. The inode of each active file or socket maintains a pointer to a knode data structure, which tracks associated kernel objects.

Implementation and Evaluation. Our implementation using Linux 4.17 kernel shows that KLOCs improve the performance of I/O-intensive workloads like RocksDB, Redis, Cassandra, and Spark over the state-of-the-art application tiering (*Nimble* [23]) by up to 2.7× on a two-tier memory system, and 1.4× on a multi-socket Intel Optane system.

2. Motivation

To manage a heterogeneous mix of memory devices, recent studies have focused on software and hardware techniques. Software approaches rely on tracking page hotness by scanning page tables to migrate hot application pages of different sizes to fast memory [17, 23, 24]. In contrast, hardware approaches for data tiering augment the memory controller to identify hot pages appropriate for fast memory [14, 18, 20]. None of the prior studies consider kernel object tiering. The closest prior work, Mitosis, studies page tables placement across NUMA memory sockets, but ignores file or networking objects which dominate kernel memory footprint [8]. Finally, current OSes lack support for kernel object migration which further complicates management.

To understand the prevalence of kernel objects, our analysis of I/O-intensive real-world application reveals that kernel objects can occupy 28-79% of total workload memory footprints and account for 25-81% of the total memory references. These kernel objects include page cache pages, journals, metadata radix trees, block driver buffers, socket buffers, and journals. While we wish to tier these objects to maximize performance, kernel objects have lifetimes of tens of milliseconds, compared to tens of minutes for application pages. This means that existing LRU code paths, which can take many seconds to identify hot/cold application pages, are simply too slow for

kernel object hotness measurement.

3. Design of KLOCs

To efficiently manage kernel objects in heterogeneous memory systems, we introduce a new OS abstraction, *kernel-level object contexts (KLOCs)* that offers mechanisms for (1) grouping related kernel objects, (2) low overhead hotness identification and migration support, and (3) support for existing OS-level data tiering policies. These activities are handled entirely within the OS and are transparent to user space.

Grouping related kernel objects At the core of the KLOC abstraction is the ability to group related kernel objects together. Careful grouping enables acceleration of application-initiated operations that require access to multiple related kernel objects. In Figure 1, we show that KLOCs use a knode data structure to group and act as a "table of contents" to the locations of all associated kernel objects associated with each inode. The knodes are allocated during inode creation (i.e., file or network socket creation) and every file's inode maintains a pointer to its associated knode. Knodes maintain a *page-cache tree* to track large kernel objects such as page cache and a *slab tree* for smaller kernel objects allocated using slab allocators. Maintaining independent trees allows quick identification of related kernel objects and enables prioritization of slab-based kernel objects with short lifetimes.

Placement and Migration of Kernel Objects We place and migrate kernel objects using knodes and their associated inodes. We allocate kernel objects associated with an inode to fast memory before adding references to them to the knode. Allocations can fail due to limited fast memory capacity. Hence, we must provide mechanisms to continuously identify hot and cold kernel objects and move them to fast and slow memory, respectively.

Per-CPU fast paths and knode LRU: A kernel object is hot when the knode with which it is associated is hot. A knode is hot when the corresponding inode is actively used. To track all knodes, they are mapped to a global kmap, which we implement using a red-black tree. The migration mechanism identifies hot and cold knodes and migrates all associated kernel objects to fast memory when hot, or memory when cold. One challenge is that there may be hundreds of knodes that are accessed or shared across tens of CPUs. Consequently, updates to the global kmap must be serialized, which could impact CPU scaling. To overcome this, we co-opt existing per-cpu structures already used by the Linux community for scalable tracking of kernel objects, and add per-cpu knode lists that provide fast access to recently-used knodes. We associate each knode with an *age* variable to track its time since last access. We augment the LRU scanner to increment the knode age field and migrate all kernel objects from colder knodes to slow memory.

Making KLOCs migratable: Current OSes support migration of application and kernel objects (e.g., page cache and *vmalloc* allocations) that are mapped in the virtual address space, but lack support for relocation of widely-used slab-allocated objects. Consequently, we design a new allocation interface for kernel objects that enables allocation of kernel objects into virtual address spaces by leveraging existing code paths for anonymous virtual memory area regions to support kernel object migration. When a KLOC corresponding to a file is migrated, the kernel objects pointed to by the appropriate knode sub-tree in Figure 1 are all migrated together.

Support for I/O prefetching, LRU, AutoNUMA We enhance Linux's existing support for I/O prefetching and AutoNUMA policies to take advantage of kernel object tiering. We also augment I/O prefetcher to allocate and prefetch I/O data to fast memory.

4. Evaluation

To evaluate KLOCs' kernel object tiering effectiveness towards improving application performance, we evaluate the KLOC abstraction on widely used storage and network-intensive benchmarks and applications such as Filebench [22], RocksDB [2], Redis [6], Cassandra [1], and Spark [25]. Our evaluation uses a two-tier fast and slow DRAM platform emulated using thermal throttling and a two-socket Intel Optane DC system representative of a hybrid OS-hardware approach for data management. Our comparison of KLOCs performance against the state-of-the-art OS-controlled application tiering shows throughput gains up to 2.7× on the two-tier platform and 1.4× on the Intel Optane platform. The benefits stem from KLOCs support for grouping related kernel objects, inexpensive hotness detection, and kernel object migration.

5. Why ASPLOS?

The computing industry is undergoing profound changes as it embraces heterogeneity at all levels of the systems stack. Heterogeneity in compute and memory are forcing researchers to reconsider conventional wisdom in systems software and hardware design and the abstractions that separate the two. This paper is the first to show that although kernel objects have traditionally been viewed as second-class citizens in memory management compared to application pages, the advent of memory heterogeneity, growth in overall memory capacity, and improvements in storage and networking speeds make memory placement of kernel objects critical to performance. As befitting a conference focused on the intersection of architecture and systems software, ASPLOS is an apt venue to showcase the power of reasoning about kernel objects in terms of contexts as an organization principle to tier kernel data.

Citation for most influential paper award This paper helped influence research on memory tiering of kernel objects, long ignored in memory management research, in the face of increasing heterogeneity of memory systems.

References

[1] Apache Cassandra. http://cassandra.apache.org/.

[2] Facebook RocksDB. http://rocksdb.org/.

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

[4] Linux Page Migration. https://www.kernel.org/doc/Documentation/vm/page_migration.

[5] Nginx memory usage. https://www.nginx.com/blog/nginx-websockets-performance/.

[6] Redis. http://redis.io/.

[7] VMWare vNUMA. https://www.vmware.com/files/pdf/techpaper/VMware-vSphere-CPU-Sched-Perf.pdf.

[8] 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 International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'20, 2020.

[9] 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, pages 631–644, New York, NY, USA, 2017. ACM.

[10] Ameen Akel, Adrian M. Caulfield, Todor I. Mollov, Rajesh K. Gupta, and Steven Swanson. Onyx: A Protoype Phase Change Memory Storage Array. In *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*, HotStorage'11, Portland, OR, 2011.

[11] Berkin Akin, Franz Franchetti, and James C. Hoe. Data reorganization in memory using 3d-stacked dram. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 131–143, New York, NY, USA, 2015. ACM.

[12] Oren Avissar, Rajeev Barua, and Dave Stewart. An optimal memory allocation scheme for scratch-pad-based embedded systems. *ACM Trans. Embed. Comput. Syst.*, 1(1):6–26, November 2002.

[13] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H. Loh, Don McCaule, Pat Morrow, Donald W. Nelson, Daniel Pantuso, Paul Reed, Jeff Rupley, Sadasivan Shankar, John Shen, and Clair Webb. Die stacking (3d) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 469–479, Washington, DC, USA, 2006. IEEE Computer Society.

[14] Chia-Chen Chou, Aamer Jaleel, and Moinuddin Qureshi. Batman: Maximizing bandwidth utilization for hybrid memory systems. In *Technical Report, TR-CARET-2015-01 (March 9, 2015)*, 2015.

[15] Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. Data tiering in heterogeneous memory systems. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 15:1–15:16, New York, NY, USA, 2016. ACM.

[16] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 404–415, New York, NY, USA, 2013. ACM.

[17] 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, pages 521–534, New York, NY, USA, 2017. ACM.

[18] M.R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G.H. Loh. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 126–136, Feb 2015.

[19] Milan Radulovic, Darko Zivanovic, Daniel Ruiz, Bronis R. de Supinski, Sally A. McKee, Petar Radojković, and Eduard Ayguadé. Another trip to the wall: How much will stacked dram benefit hpc? In *Proceedings of the 2015 International Symposium on Memory Systems*, MEMSYS '15, pages 31–36, New York, NY, USA, 2015. ACM.

[20] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 85–95, New York, NY, USA, 2011. ACM.

[21] Jia Rao, Kun Wang, Xiaobo Zhou, and Cheng zhong Xu. Optimizing virtual machine scheduling in numa multicore systems. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 306–317, Feb 2013.

[22] Tarasov Vasily. Filebench. https://github.com/filebench/filebench.

[23] 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, pages 331–345, New York, NY, USA, 2019. ACM.

[24] Zi Yan, Jan Vesely, Guilherme Cox, and Abhishek Bhattacharjee. Hardware translation coherence for virtualized systems. In *International Symposium on Computer Architecture*, ISCA '17, 2017.

[25] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.