xmap: Transparent, Hugepage-Driven Heap Extension over Fast Storage Devices

Ioannis Malliotakis
ICS-FORTH & University of Crete
Heraklion, Greece
jmal@ics.forth.gr

Anastasios Papagiannis
Isovalent
Athens, Greece
anastasios.papagiannis@isovalent.com

Manolis Marazakis
ICS-FORTH
Heraklion, Greece
maraz@ics.forth.gr

Angelos Bilas
ICS-FORTH & University of Crete
Heraklion, Greece
bilas@ics.forth.gr

Abstract

Increasing dataset sizes requires that applications use larger heaps. Two characteristic examples are graph analytics and machine learning (ML), both emerging workloads with rapidly increasing memory demands. With recent technology limitations in DRAM scaling, one important question is how to grow the heap further. One approach is the use of specialized out-of-core frameworks which can process datasets larger than the available DRAM capacity. However, developing and tuning such frameworks for performance is both complex and time-consuming, and frameworks for graph analytics and ML have traditionally been in-memory. Another approach for extending the application heap transparently to applications is fast, block-addressable storage devices, such as NVMe SSDs, over the OS memory mapped I/O (*mmio*) or swapper (*swap*) path. These paths allow the extension of the application heap without application modifications through the abstraction of the process virtual address space.

Although both *mmio* and *swap* are part of the core kernel operation, they have significant limitations for use with application heaps. First, they fail to scale beyond 8 cores due to shared structure contention in OS paths stressed during heap extension, namely the page fault, page reclamation and page writeback paths. Second, they do not offer read-write support for hugepages over block storage. Hugepages are contiguous memory frames larger than the regular page size (e.g., 2MB or 1GB on x86_64), which can be mapped with a single TLB and (huge) page table entry. Hugepages have received increased attention in in-memory setups for their potential to reduce CPU cycles spent on TLB misses, and reduce page faults, thus reducing kernel processing software overheads. Applying hugepages to heap extension can aid in harnessing the increasing throughput capabilities of NVMe SSDs. Third, they offer limited asynchronous operations, which are important to mask the higher latency of the backing storage medium, namely aggressive readahead, which is ill-fitted for heap extension, where memory must be treated as a scarce resource.

Driven by these limitations, we design *xmap* as an alternative *mmio* path for the Linux kernel, tailored towards heap extension over fast block storage devices. *xmap* provides improved scalability, support for transparent hugepages over block-based storage, and asynchronous hugepage promotions. To our knowledge, *xmap* is the first system that provides this support for the Linux kernel. *xmap* is implemented as a kernel module, requiring no kernel modifications. It utilizes its own preallocated and statically divided regular and hugepage pools, to operate under a strict DRAM budget and prevent external memory fragmentation due to the use of hugepages. It organizes pages in its own sharded page cache and metadata structures, to mitigate shared structure contention and improve scalability. *xmap* utilizes hugepages either via explicit application demand (i.e., *madvise*) or transparently in a policy-driven manner. *xmap* supports both the synchronous preparation and mapping of hugepages to the application address space at page fault time, and the asynchronous promotion of hugepage-sized virtual address ranges from mapping to regular pages to mapping to a hugepage.

We first examine the performance implications of extending the application heap over NVMe SSDs, by characterizing the application-perceived memory access latency and throughput through LMbench workloads. We then evaluate *xmap* with a random page fault microbenchmark, a machine learning workload (LIBLINEAR), and graph processing frameworks (Ligra+, GridGraph), by transparently extending their heap over storage without any code modifications. We find that *xmap* scales beyond 8 cores, increasing random page fault throughput by up to 3.9× compared to Linux *mmap*. When using a heap/DRAM ratio of 2×, *xmap* improves linear regression training time by up to 7.9× compared to Linux *mmap*. *xmap* allows an in-memory graph processing framework (Ligra+) to perform comparably to or even better than a hand-written out-of-memory graph processing framework (GridGraph) when processing graphs 4×-8× larger than the available DRAM, without any code modifications.

Acknowledgments

We thankfully acknowledge the support of the European High-Performance Computing Joint Undertaking (EuroHPC JU) under project EUPEX (grant agreement No 101033975). The EuroHPC JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Germany, Italy, Greece, United Kingdom, Czech Republic, Croatia.