DuoSwap: adaptive concurrent swapping to compressed memory and NVMe SSD

Yuben Yang
University of Sydney

Baptiste Lepers
Inria

Kimberly Keeton
Google

Khaled Elmeleegy
Google

Willy Zwaenepoel
University of Sydney

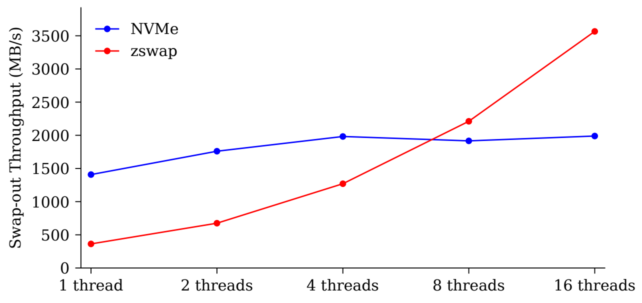
1 Motivation

DRAM has become one of the most important CapEx costs in a data center, as a result of the rising cost of memory and the very large address spaces used by many applications [3, 11]. At the same time, it has been repeatedly observed that much of the memory lies idle [6, 7, 14]. As a result, swapping has again become an important component of system management and an active topic of research [4, 12, 13].

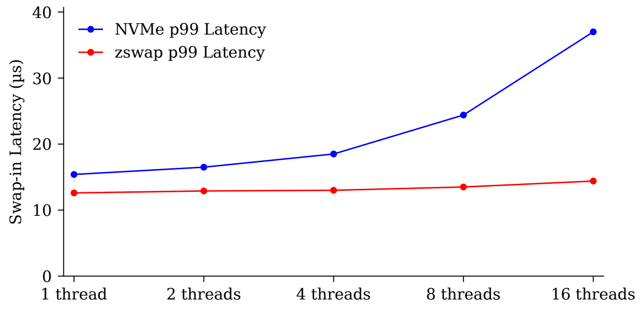
Swapping transparently reduces the memory footprint of applications. The standard options for swapping are compressing the infrequently accessed data in DRAM or moving that data to a slower but more cost-effective tier, typically an SSD [14]. Conventionally, there have been large gaps in bandwidth and latency between in-memory compression and moving data to SSDs. For that reason, compressing the data has been used as a first resort, and moving data to SSD is only done when swapping by compression is disabled, when the data is not compressible or when the system runs out of DRAM space for compression [5, 8, 10].

Observation #1: Comparable swap performance. The advent of new storage devices, such as NVMe SSDs, with higher bandwidths and lower latencies, calls for a reconsideration of this tradeoff. As shown in Figure 1, we observed that swapping by compression (represented by the popular zswap [1] kernel subsystem) and swapping with NVMe SSD have comparable performance in terms of swap-out throughput and swap-in tail latency. While zswap still performs better than NVMe SSD under high load and with sufficient CPU cycles available, the gap has become much smaller than what was previously the case, to the point that one can contemplate using both concurrently rather than hierarchically.

Observation #2: No single static split wins. The remaining technical question to be resolved is how to divide the work between in-memory compression and NVMe SSD swapping in a way that consistently provides good performance. In Figure 2, using swap-out throughput as an illustrative metric, we argue that no single static division of labour among the two swap backends prevails in all cases, because the relative performance of swapping by compression and NVMe SSD swapping varies depending on the resource availability. Furthermore, since compression imposes a load on the CPUs, CPU-bound workloads benefit from having a larger share of their pages sent to NVMe SSD. Vice versa, I/O-bound



(a) Swapout throughput for zswap and NVMe SSD.



(b) 99th percentile swapin latency for zswap and NVMe SSD.

Figure 1. Swap performance for zswap and NVMe SSD

workloads benefit from having more pages compressed. In summary, the "correct" split ratio depends on the application's behavior under swapping, resource availability, and the potential background load from other processes. Hence, the splitting needs to be adaptive.

Observation #3: Compressibility matters. Intuitively, the relative benefit of the two swapping targets also depends on the compressibility of the data to be swapped out. We found that not only does higher compressibility lead to better usage of zswap space, but also it results in higher swap-out throughput for compression, which further motivates the need for adaptive splitting.

2 DuoSwap Design Overview

We present DuoSwap which takes the aforementioned factors into account to decide the split of the swapping traffic between in-memory compression and NVMe SSD. DuoSwap

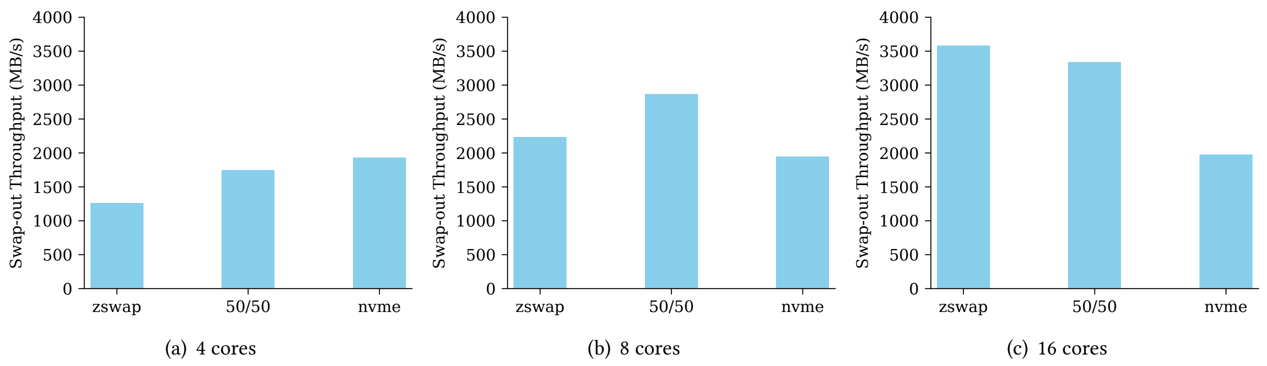


Figure 2. Swap-out throughput comparison among zswap, NVMe and 50-50 split with different numbers of CPU cores.

is built on top of the default Linux reclaim mechanism that reclaims pages in batches. DuoSwap consists of three subsystems: splitting, compressibility estimation and ranking, and pageout. The first subsystem, splitting, periodically recomputes the split ratio (which determines the fraction of dirty pages in a reclaim batch that should be compressed and the fraction that should be sent to the NVMe SSD), by monitoring CPU pressure and I/O pressure indicated by Pressure Stall Information (PSI) [2, 14]. The second subsystem, compressibility estimation and ranking, estimates the compressibility of each dirty page using Information Entropy [9, 10] and ranks pages accordingly. The third subsystem, pageout, takes as input the split ratio and dirty page list sorted in the order of compressibility. Given the fractions that should go to zswap and NVMe SSD, it compresses the most compressible pages and sends the remainder to NVMe SSD.

We have implemented DuoSwap in the Linux kernel v6.1. We show that DuoSwap provides better performance in most circumstances and always provides performance at least equal to Linux. Moreover, by measuring the performance across the range of static splits of the data we show that our adaptive algorithm always approximates the performance of the best (static) split.

The contributions of our work are:

- The idea of concurrently using compression and NVMe SSD for swapping.
- An adaptive algorithm for splitting swapping between compression and NVMe SSD, based on data compressibility and resource availability.
- The implementation of the algorithm in the Linux kernel and its evaluation.

References

[1] Linux zswap. https://docs.kernel.org/admin-guide/mm/zswap.html.

[2] Psi - pressure stall information. https://docs.kernel.org/accounting/psi.html.

[3] M. Ahn, A. Chang, D. Lee, J. Gim, J. Kim, J. Jung, O. Rebholz, V. Pham, K. Malladi, and Y. S. Ki. Enabling cxl memory expansion for in-memory database management systems. In *Proceedings of the 18th International Workshop on Data Management on New Hardware*, DaMoN '22, New York, NY, USA, 2022. Association for Computing Machinery.

[4] S. Bergman, N. Cassel, M. Bjørling, and M. Silberstein. Znswap: unblock your swap. *ACM Trans. Storage*, 19(2), Mar. 2023.

[5] W. Cao and L. Liu. Dynamic and transparent memory sharing for accelerating big data analytics workloads in virtualized cloud. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 191–200, 2018.

[6] M. Ewais and P. Chow. Disaggregated memory in the datacenter: A survey. *IEEE Access*, 11:20688–20712, 2023.

[7] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao. Who limits the resource efficiency of my datacenter: an analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[8] S. Jennings. Transparent memory compression in linux. In *LinuxCon North America*, 2013.

[9] C. Ji, L.-P. Chang, L. Shi, C. Gao, C. Wu, Y. Wang, and C. J. Xue. Lightweight data compression for mobile flash storage. *ACM Trans. Embed. Comput. Syst.*, 16(5s), Sept. 2017.

[10] J. Kim, C. Kim, and E. Seo. $ezswap$: Enhanced compressed swap scheme for mobile devices. *IEEE Access*, 7:139678–139691, 2019.

[11] A. Lagar-Cavilla, J. Ahn, S. Souhlal, N. Agarwal, R. Burny, S. Butt, J. Chang, A. Chaugule, N. Deng, J. Shahid, G. Thelen, K. A. Yurtsever, Y. Zhao, and P. Ranganathan. Software-defined far memory in warehouse-scale computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 317–330, New York, NY, USA, 2019. Association for Computing Machinery.

[12] J. Lee, S. Park, M. Ryu, and S. Kang. Performance evaluation of the ssd-based swap system for big data processing. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 673–680, 2014.

[13] M. Saxena and M. M. Swift. Flashvm: virtual memory management on flash. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, page 14, USA, 2010. USENIX Association.

[14] 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 *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 609–621, New York, NY, USA, 2022. Association for Computing Machinery.