Memory Tiering with Intel® In-Memory Analytics Accelerator (Intel® IAA)

Addressing Memory Challenges: Enhancing Data Center Efficiency with zswap and Intel IAA

Authors

Binuraj Ravindran
Principal Engineer

Bala Seshasayee
Software Research Engineer

Kanchana Sridhar
Cloud Software Architect

Rakib Al-Fahad
Cloud Software Development Engineer

Vinodh Gopal
Senior Principal Engineer

Pallavi G
Cloud Software Development Engineer

Muktadir Chowdhury
Cloud Software Development Engineer

The increasing demand for efficient memory management in modern data centers necessitates innovative strategies to optimize memory usage and performance. Memory tiering is one such approach, leveraging high-speed premium memory for frequently accessed data while relegating infrequently accessed data to slower, cost-effective memory tiers, achieving an optimal balance of performance and cost. Complementing this, the zswap feature in the Linux kernel enhances memory tiering by compressing less frequently accessed memory pages, retaining them in a compacted state to effectively extend DRAM capacity without relying solely on conventional disk swapping. However, the CPU cycles spent on compression and decompression operations, being in the critical execution path, can significantly impact zswap's performance and associated cost savings. Recent research papers[3,4] highlight the need for native hardware-assisted compression and decompression accelerators to enhance zswap's efficiency and scalability.

The 4th generation Intel® Xeon® processor and later models include the Intel® In-Memory Analytics Accelerator (Intel® IAA), a tightly integrated accelerator designed for low-latency and high-throughput compression and decompression. This paper investigates the integration of Intel IAA to augment zswap functionality and efficiency. By offloading compute-intensive compression and decompression tasks from the CPU to Intel IAA, we demonstrate substantial improvements in compression speed and efficiency, reducing both processing overhead and the DRAM footprint associated with zswap. Benchmarking results reveal that Intel IAA improves swap-in and swap-out latencies up to 2x-7x compared to software-based compression algorithms. These findings underscore the potential of hardware-accelerated memory management solutions in meeting the escalating memory demands of contemporary data centers.

Introduction

In the era of data-driven computing, modern data centers are tasked with managing vast amounts of information while delivering high performance and maintaining cost-efficiency. Memory management has emerged as a critical challenge as the cost of high-speed memory, such as DRAM, can quickly escalate in large-scale environments. To address this challenge, memory tiering has gained prominence as an effective strategy. Memory tiering involves organizing memory resources into distinct tiers based on speed, capacity, and cost. Frequently accessed data is stored in high-performance, low-latency memory, while less frequently accessed data is relegated to slower, more economical storage options. This hierarchical approach optimizes resource allocation, striking a balance between performance requirements and operational expenses.

Linux zswap is a complementary solution designed to extend the effectiveness of memory tiering. Zswap works by compressing infrequently accessed memory pages and storing them in a compact form within DRAM, effectively increasing the usable memory capacity without resorting to slower disk-based swapping. By reducing reliance on disk I/O, zswap enhances system responsiveness and minimizes latency. By avoiding the need for extra memory devices based on Open Memory Interface (OMI), Compute Express Link (CXL), and more, zswap is also a low-cost memory-tiering solution. In the Linux kernel, zswap is a part of the reclaim and page-fault handling flows, intercepting calls to storing and loading pages to and from the backing swap device. The amount of memory dedicated for zswap is configured during initialization, and then pages headed to the swap device are compressed before storing. Upon nearing capacity, Least Recently Used (LRU) writebacks to swap device are performed. Incompressible pages are also directly sent to the swap device.

While zswap offers a scalable solution, it is crucial to manage the overhead it introduces and its impact on workload performance. Compression and decompression latencies significantly contribute to the overall swap-out and swap-in times. This overhead can limit the scalability and cost savings that zswap can achieve in large-scale deployments.

One of the solutions to help minimize the overhead of zswap is to offload compression and decompression operations from the CPU to a dedicated hardware accelerator to further improve zswap's efficiency and scalability. The need for native hardware-assisted compression and decompression accelerators to enhance zswap is also highlighted in the recent research papers[3,4] on large-scale zswap deployments.

Intel IAA is one of the closely coupled accelerators available from the 4th generation Intel Xeon processor, designed to efficiently compress and decompress small data blocks (such as the 4 KB size commonly used in Linux virtual memory pages) with very low latency and high throughputs.

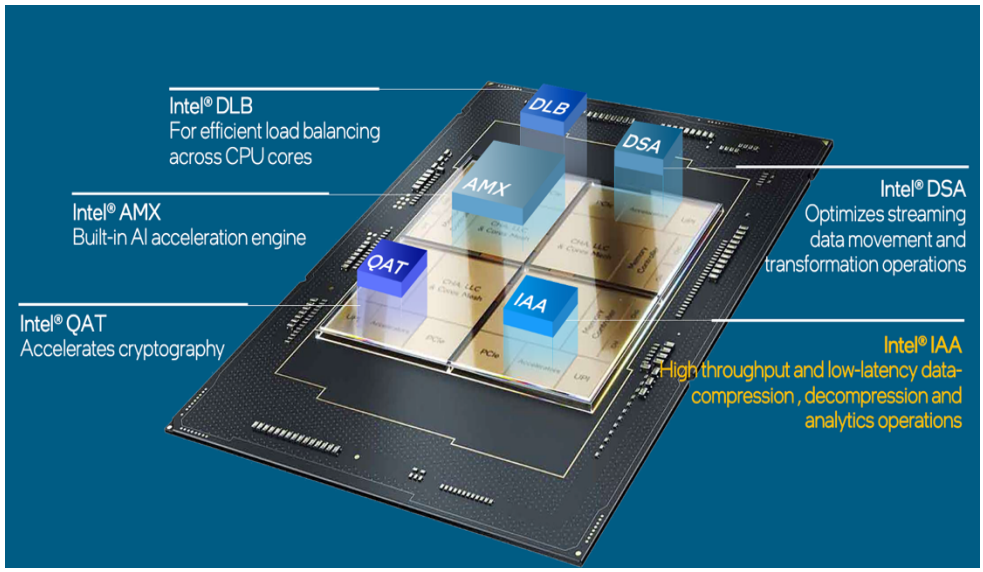

Figure 1. Hardware accelerator in 4th generation and higher Intel® Xeon® Scalable processors

Figure 1 illustrates the various analytics engines in the Intel Xeon Scalable processor, from the 4th generation and higher, that include Intel IAA. In the rest of the paper, we will go over how Intel IAA is integrated to zswap, the performance benchmarking methodology, the performance improvements with Intel IAA, and the innovations in the future Intel IAA hardware versions.

Intel IAA Integration with zswap and Compression Modes

Intel IAA is integrated with zswap at the kernel level via the Intel IAA crypto driver,[2] which is available starting with the Linux kernel version 6.8. The zswap compressor can be selected through the sysfs interface located at /sys/module/zswap/parameters/compressor. Traditionally, zswap supports software compressors such as zstd, lzo, and lz4. The Intel IAA crypto driver simplifies the process of using Intel IAA by allowing users to specify one of the Intel IAA compression modes through the same sysfs interface. For instance, to use the default Intel IAA compression mode, one can run echo deflate-iaa > /sys/module/zswap/parameters/compressor. This default mode, deflate-iaa, is provided in the initial version of the Intel IAA crypto driver. Additionally, kernel patch sets currently under review introduce two more Intel IAA compression modes,[8] canned and dynamic, thus further expanding the capabilities and flexibility of Intel IAA in optimizing zswap performance.

In the field of data compression, balancing the trade-off between compression ratio and latency is a crucial consideration for system architects and developers. The Intel IAA crypto driver, with its versatile compression modes, provides a range of options for fine-tuning this balance. Each mode is designed to address specific use cases, enabling users to prioritize either compression ratio or latency according to their application requirements. Table 1 outlines the different Intel IAA compression modes and compares them with other commonly used software compression algorithms in zswap, highlighting their respective strengths and trade-offs.

Table 1. Comparison of Intel IAA compression modes against software compression algorithms

Algorithms/Modes	Description
deflate-iaa[-fixed]	A hardware-accelerated deflate algorithm[9] with a 4KB history buffer. This is the default mode enabled in the Intel IAA crypto module. [8] It provides the best latency with a lower compression ratio than other modes. Available from 4th generation Intel Xeon processors and Linux kernel version 6.8.
deflate-iaa-dynamic	A hardware-accelerated dynamic mode uses Huffman tables generated and optimized for the input. This mode gives the best compression ratio but has the longest latency among the modes. The dynamic mode is supported in hardware only from 6th generation Intel Xeon processors.
deflate-iaa-canned	A hardware-accelerated deflate algorithm with a custom Huffman table fine-tuned for Spec CPU® 2017[20] memory content. This mode is a middle ground between the fixed and dynamic modes and has reduced latency compared to the dynamic mode while achieving a higher compression ratio than the fixed mode. Available from 4th generation Intel Xeon processors. The kernel patches are under review for upstreaming as of this writing.
lz4[14]	Software compression algorithm with a very low decompression latency.
zstd[16]	Software compression algorithm with a high compression ratio, but with a high latency.

As highlighted earlier, compression and decompression operations can consume a substantial portion of CPU cycles during zswap operations, leading to significant overhead and impacting overall workload performance. By offloading the compression and decompression tasks from the CPU, we can substantially reduce this overhead. Intel IAA provides lower decompression latencies and achieves comparable or better compression ratios compared to software-based algorithms in most cases. Taking advantage of Intel IAA for offloading compression and decompression can therefore significantly reduce workload impact when compared to software-based compression algorithms. In the next section, we will present a detailed analysis of the performance improvements achieved by integrating Intel IAA at the microbenchmark level compared to traditional software compression algorithms.

Performance Benchmarking Methodology

To quantify the improvements in zswap performance from using Intel IAA compared to software compression algorithms, we employed a user-space workload designed to generate repeated swap outs followed by swap ins of anonymous pages. The framework uses a user-space workload, Workload(madvise), which relies on the madvise() system call with the PAGEOUT directive to force the swap out of pages, followed by subsequent access to each page to trigger a swap in. Using the commonly available bpftrace mechanism, the Observer(bpftrace) monitors compression and decompression latencies, compression ratios, and other relevant metrics as the pages are swapped in and out.

The benchmarking methodology is illustrated in Figure *2*. The source code for the benchmark, along with the supporting scripts used for data collection, is available on GitHub.[17]

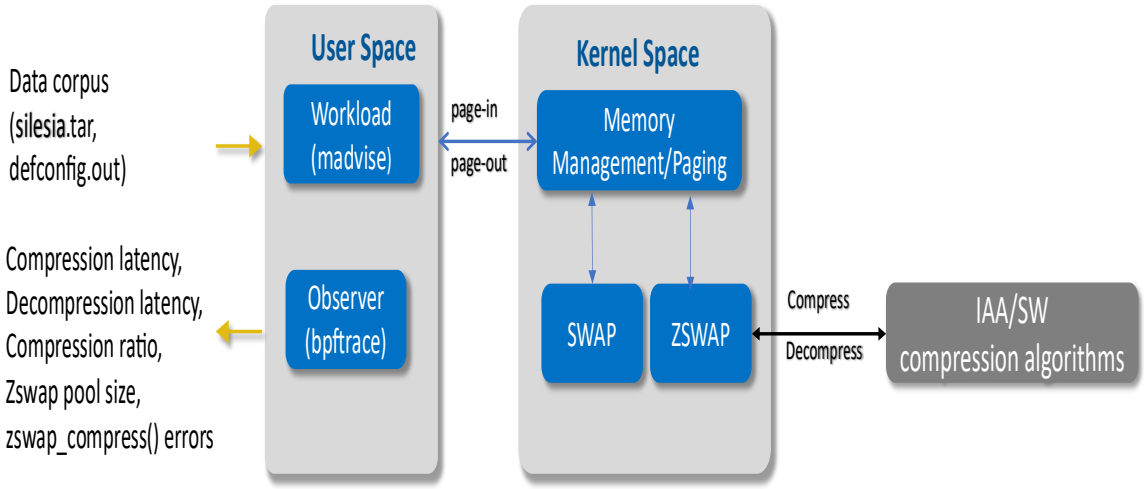

Figure 2. The methodology and framework to benchmark zswap performance improvement with IAA.

In real-world scenarios, the nature of the data being swapped is typically determined by the specific workload and the intensity of memory pressure applied. To create a consistent and repeatable testing environment that mimics these conditions, we populated the system's memory with a fixed data corpus. After loading the memory with the data corpus, we used the madvise() system call to deliberately trigger the swapping process, simulating a swap out as would occur under memory pressure. With zswap enabled, pages successfully compressed by zswap_compress() are stored in the zswap pool, allowing us to monitor compression latency, compression ratio, and zswap pool size. Pages that are not successfully compressed (considered incompressible) are stored in the backing swap device by the kernel. During the swap-in process, zswap_decompress() calls enable us to monitor decompression latency. This experiment is repeated with different Intel IAA compression modes and software compression algorithms, as described in Table 1, to compare their performance.

Table 2 details two different data corpuses used to populate the memory to capture the performance across different compression ratio ranges with diverse data content.

Table 2. Datasets used to populate the memory to allow for a wide range of compression ratios.

Datasets	Description
silesia.tar[12]	Well-known corpus for data compression with diverse contents
defconfig.out[17]	Swap memory snapshot from a Linux kernel build benchmark[21] in defconfig mode; a proxy for a typical zswap content with slightly higher compression ratios than silesia.tar

It is important to acknowledge that this approach does not perfectly mirror the complexities and unpredictable nature of actual workloads. Real-world applications often involve a multitude of variables that can affect the swapping behavior and, consequently, the performance of compression algorithms. However, this methodology provides a controlled way of comparing compression algorithms. By using a single-threaded workload and standardized data corpus with varying compressibility, we can objectively evaluate and compare the latency and compression efficiency of different compression algorithms.

The key performance indicators selected for evaluating software compression algorithms in comparison to Intel IAA are provided in Table 3.

Table 3. Key metrics measured to evaluate the performance improvement compared to software compression algorithms.

Metrics	Description	Comments
Swap-out (compression) latency	Time to swap out a page. This includes zswap compress, storage in zpool, and other overheads.	Monitored at swap_writepage() through bpftrace with zswap enabled. Lower is better.
Swap-in (decompression) latency	Time to swap in a page. This includes zswap decompress and other overheads.	Monitored at swap_read_folio() through bpftrace with zswap enabled. Lower is better.
Compression ratio	Uncompressed_size and compressed_size measured at zswap_compress()	Metric to compare the effectiveness of the compression algorithms. Higher is better.
zswap pool size	zswap size reported by /sys/kernel/debug/zswap/pool_total_size	Higher is the compression ratio. Lower is the zswap size. Lower zswap size is better.
Pages rejected by zswap_compress()	Cumulative errors reported by /sys/kernel/debug/zswap/reject_compress_fail, /sys/kernel/debug/zswap/reject_compress_poor, and /sys/kernel/debug/zswap/reject_alloc_fail	Measure of incompressibility of the page content. Lower is better. Incompressible pages are moved to swap, incurring higher latencies from disk access.

Swap-out and swap-in latencies are directly influenced by compression and decompression latencies, respectively. The compression ratio plays a key role in determining the zswap pool size and the overall memory efficiency of zswap. Additionally, the number of incompressible pages is critical, as these pages are offloaded to the swap device instead of being stored in the in-memory zswap pool, impacting system performance and efficiency. In the following section, we will examine how Intel IAA enhances zswap performance using the previously outlined metrics.

Performance Improvement with Intel IAA

zswap with Intel IAA delivers significant improvements in both swap-out and swap-in latency, as well as compression ratio, when compared to software compression algorithms. The performance with a 4 KB page size is used as the baseline for comparison against other configurations, such as parallel processing. The data points are normalized to the performance of the deflate-iaa mode to provide comparison points as this mode is already available in Linux kernel version 6.8 and later. The deflate-iaa-canned and deflate-iaa-dynamic modes offer better compression ratios compared to the deflate-iaa mode and are currently available as RFC kernel patches[8] for evaluation at the time of writing this paper. Performance improvements are dependent on the dataset being used for the characterization. Two datasets–silesia.tar and defconfig.out–are characterized here to allow for a diverse set of data contents.

Swap-out and Swap-in Latency

The swap-out (compression) latency is measured at swap_writepage() and swap-in (decompression) latency is measured at swap_read_folio() kernel functions, respectively. In Figure 3, the average swap-out and swap-in latencies of Intel IAA compression modes are compared to those of software compression algorithms such as lz4 and zstd. The results are normalized against deflate-iaa for easier comparison.

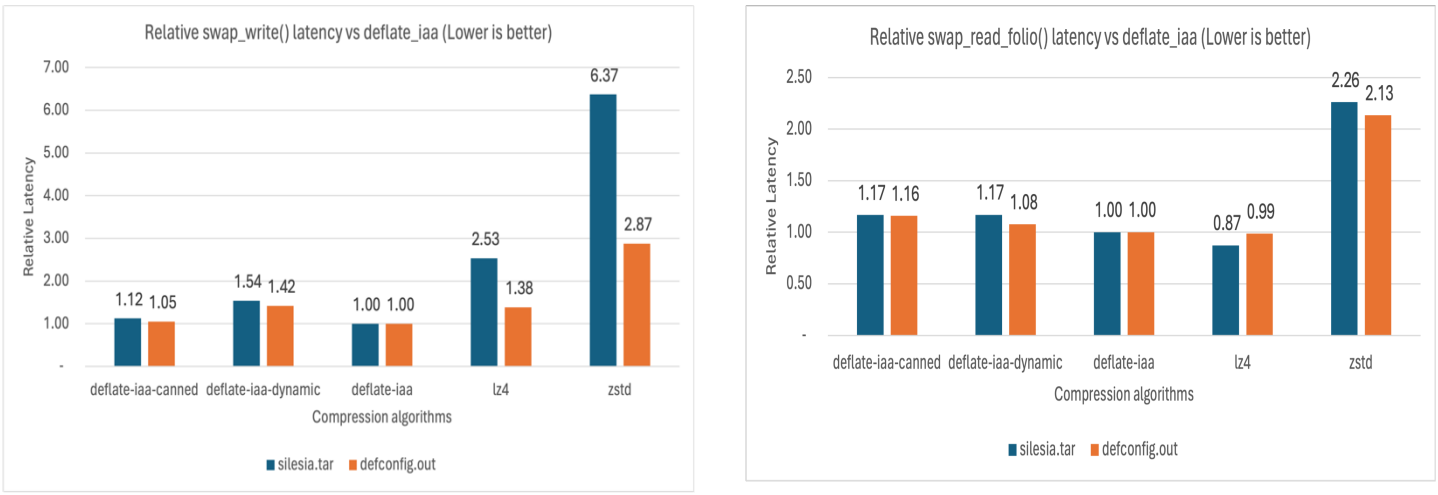

Figure 3. Latency improvement with Intel IAA compared to other software compression algorithms.

With Intel IAA compression modes, swap-out latency improves by up to 1.38x-2.53x compared to lz4, and up to 2.87x-6.37x compared to zstd. For swap-in latency, Intel IAA compression modes offer an improvement of up to 2.13x-2.26x compared to zstd. While there is a slight increase in latency for Intel IAA as compared to lz4, this trade-off is offset by the improved compression ratio.

Compression Ratio and zswap Pool Size

The compression ratio plays an important role in the efficiency of zswap by directly influencing the space required to store compressed pages in the DRAM-based compressed pool (zswap pool). Higher compression ratios enable a greater number of memory pages to be stored within the same pool, effectively increasing the system's usable memory capacity, and reducing dependency on slower, disk-based swapping. Since compression algorithms represent a trade-off between compression ratio and compression and decompression speed, it is essential to evaluate their performance with consideration for both factors.

Figure 4 compares the compression ratio of Intel IAA compress modes and software compression algorithms for silesia.tar and defconfig.out data corpus. Compression ratio is computed as the ratio of the uncompressed to compressed size, as measured in the zswap_compress() kernel function. Intel IAA compression modes achieve compression ratios that fall between those of the lz4 and zstd algorithms while simultaneously providing lower decompression latencies as discussed in the previous section. The diverse compression modes offered by Intel IAA provide the flexibility to choose the optimal mode based on whether compression ratio or speed is the priority in the usage scenarios.

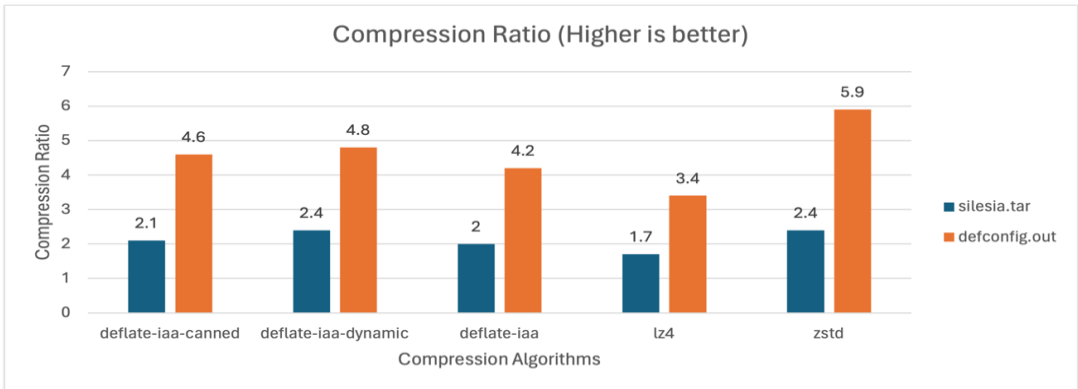

Figure 4. Compression ratio comparison across Intel IAA compression modes and software compression algorithms

Another method to evaluate the impact of compression algorithms on zswap performance is to monitor the zswap pool size and the number of rejected pages reported by the zswap_compress() kernel function. As noted earlier, a higher compression ratio leads to a smaller zswap pool size, improving memory efficiency. A lower count of rejected, or incompressible, pages is preferred

since such pages are offloaded to swap space, typically located on disk devices. Given that disk access incurs significantly higher latency compared to in-memory zswap operations, reducing the number of rejected pages is essential to preserve overall system performance.

As illustrated in Figure 5, Intel IAA compression modes can reduce the zswap pool size up to 1.3x compared to lz4 depending on the compression modes. Zstd has a better compression ratio and hence a similar or slightly lower zswap zpool size compared to

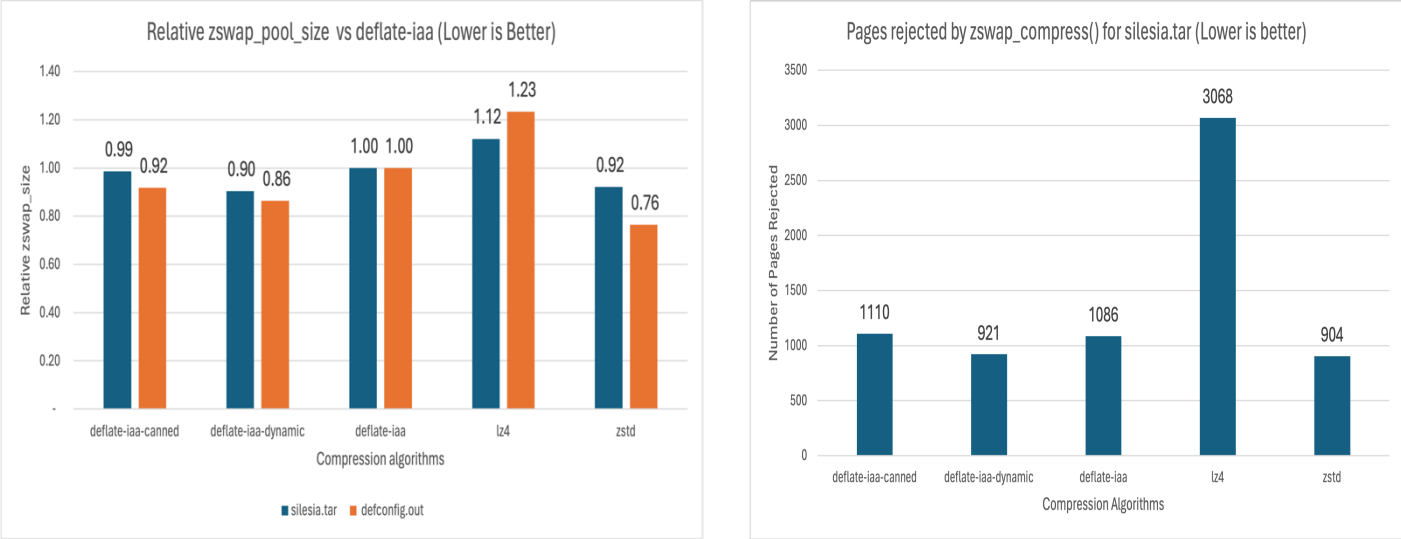

Figure 5. zswap pool size comparison with deflate-iaa and pages rejected during the zswap_compress() call

Intel IAA compression modes, which is a trade-off for the 2x-4x improvement in the compression and decompression latencies achievable with Intel IAA compared to zstd. Additionally, Intel IAA compression modes produce fewer incompressible pages (such as pages rejected by the zswap_compress() kernel function) compared to lz4. A lower number of incompressible pages translates to reduced swap overhead, thereby minimizing the performance impact on workloads. For the defconfig dataset, the number of rejected pages was nearly zero, and thus it was excluded to simplify the analysis, focusing solely on the silesia.tar data points.

zswap Performance Improvement with Intel IAA Batching

Intel Xeon Scalable processors provide significant opportunities for parallel compression and decompression through multiple Intel IAA devices and multiple compression and decompression engines within the same Intel IAA device. Depending on the processor model, up to four Intel IAA devices may be available per socket. These engines enable parallel compression and decompression operations and enable further reductions in swap-out and swap-in latencies, particularly in scenarios such as page reclaim and swap-in readahead flows triggered from multiple CPU cores and help to reduce the tail latencies. These scenarios often involve multiple pages scheduled for compression and decompression, making them ideal candidates for batching operations.

Additionally, when multisize Transparent Huge Pages (mTHP)[7] is enabled, further opportunities arise to leverage the parallel compression and decompression capabilities of Intel IAA. mTHP allows folio sizes larger than the traditional 4 KB, such as 16 KB, 32 KB, or 64 KB, and supports a combination of these sizes. For folios larger than 4 KB, they can be divided into “n” smaller 4 KB blocks, enabling parallel compression. This section examines the potential of parallel compression and decompression by employing batching through a case study aimed at optimizing zswap with mTHP.

At the time of writing, Linux kernel patches[9,10,11] that enable Intel IAA batching capabilities are under review as part of a Request for Comments (RFC) process. With Intel IAA batching patches, we can aggregate hybrid folios comprising of any-order mTHP folio sizes (like a mix of 4 KB, 16 KB, 32 KB, and 64 KB pages) to maximize swap-out throughput and minimize per-folio compression latency with Intel IAA. These patches use interfaces from the reclaim and swapin_readahead modules to create batches such as 2, 4, 8, 16, and 32 folios to perform decompression in parallel.

Hybrid batching introduces the concept of a compress batch size (CBn), which allows the Linux kernel to reclaim code to send CBn-folios of any size to be stored as a batch. zswap will construct batches of up to eight pages to be compressed in parallel to

maximize the compression performance at the Intel IAA crypto driver level. Similarly, we can construct batches of up to 32 folios (DBn) prefetched by swapin_readahead to maximize swap-in throughput with Intel IAA decompression batching. The folios batched for decompression are still 4 KB, in keeping with the existing Linux kernel behavior. However, with swap out, our batching solution uses large folios unsplit by reclaim. These large folios are compressed in batches of 4 KB subpages, thereby using all Intel IAA compress engines. We see significant improvements in reclaim throughput, workload performance and per-folio latencies with Intel IAA batching. These improvements are feasible in both proactive and reactive reclaim scenarios.

To analyze the latency improvement with Intel IAA batching, we will first look at the compression and decompression latency scaling as we sweep the CBn and DBn. The latency numbers are measured for deflate-iaa compression mode at crypto_acomp_batch_compress() and crypto_acomp_batch_decompress() kernel functions using silesia.tar data corpus. The relative compression and decompression latency is normalized to CBn, DBn=8 as illustrated in Figure 6. As a greater number of pages can be batched, the compression and decompression latency can be reduced up to 3.2x-3.68x, respectively, with a batch of 8. As the maximum number of engines in an Intel IAA device is 8, the batching is limited to 8 at Intel IAA crypto driver level. Hence the latency improvement saturates beyond CBn, DBn=8.

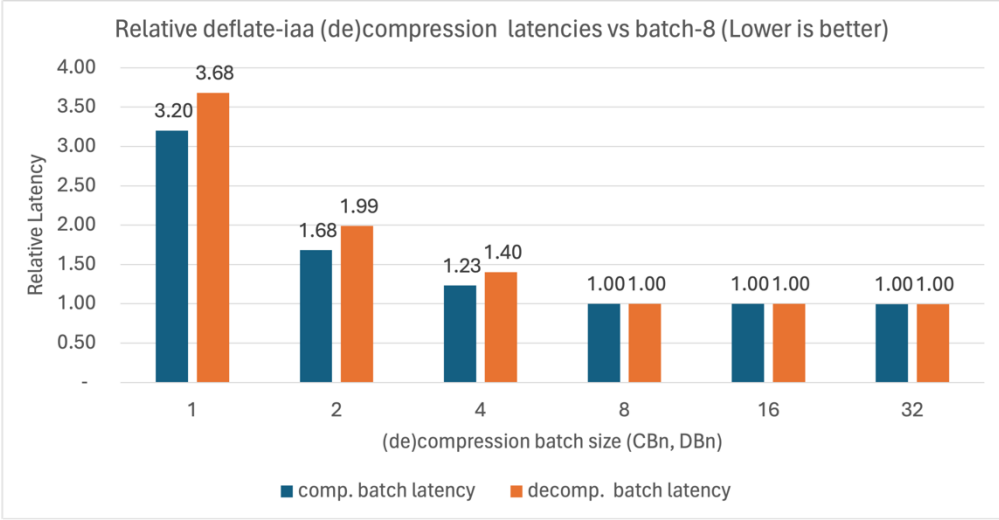

Figure 6. Compression and decompression latency scaling with Intel IAA batching

Given the significant improvements observed with Intel IAA batching in crypto_acomp_batch_compress() and crypto_acomp_batch_decompress(), we examine its impact on swap-out and swap-in latency compared to software compression algorithms. Specifically, we measure the average time required to swap out all pages from the silesia compression corpus (silesia.tar) and then swap them back in. The average swap-out and swap-in latencies are determined by normalizing the

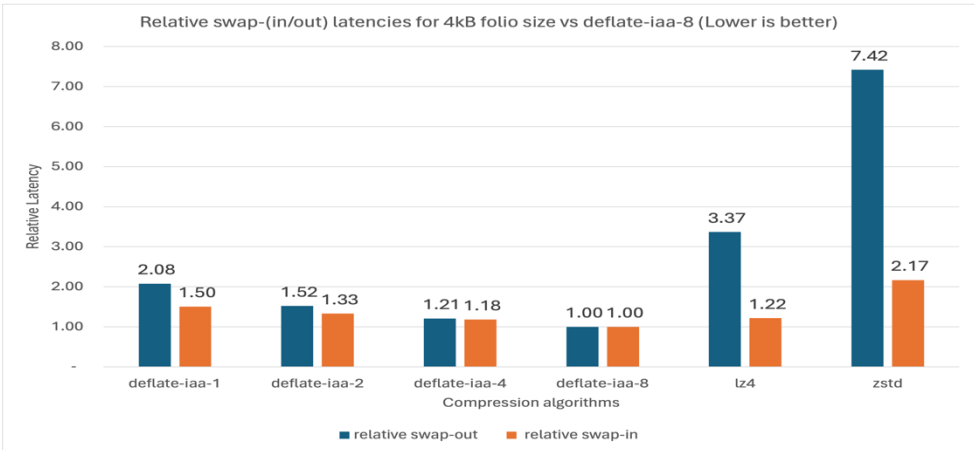

Figure 7. Swap-out and swap-in latency scaling with Intel IAA batching

total time taken to complete these operations by the total number of pages in silesia.tar. Figure 7 shows the relative swap-out and swap-in latencies with siliesia.tar dataset, normalized to deflate-iaa compression mode with a batch size of 8 (deflate-iaa-8),

and compares it with other Intel IAA batch configurations and software compression algorithms. Using deflate-iaa-8, swap-out latencies can be improved by 3.37x-7.42x compared to lz4 and zstd respectively. Similarly, swap-in latencies can be enhanced by 1.22x-2.17x with deflate-iaa-8 compared to lz4 and zstd respectively.

Increasing the mTHP folio size to larger values like 64 KB allows more opportunities for parallel compressions, as large pages can be broken down into smaller pages like 4 KB and compression performed in parallel. Figure 8 illustrates the further improvement in swap out and swap in latency that can be achieved with 64 KB mTHP folios. Swap-out latency can be improved by 4.17x-10.25x and swap-latency can be improved by 1.22x-2.21x compared to lz4 and zstd, respectively.

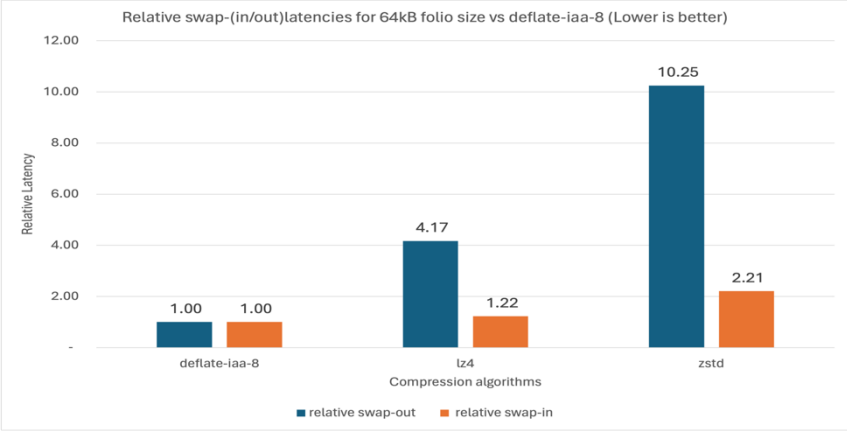

Figure 8. Swap-out and swap-in latency compared to software compression algorithms with Intel IAA batching and 64 KB

In this section, we have explored the significant potential of parallelized compression and decompression with batching using multiple Intel IAA engines in Intel Xeon Scalable processors, particularly for optimizing zswap with mTHP. The data demonstrates that Intel IAA batching achieves substantial reductions in swap-out and swap-in latencies, delivering notable performance gains.

Concluding Insights and Future Directions: Advancing zswap with Intel IAA Batching

In response to expensive memory parts and the surge in memory consumption within data centers, there is a pressing need for innovative approaches to memory-tiering strategies. One promising approach, Linux zswap, is the integration of a compressed tier within the memory architecture itself. This method selectively compresses memory pages that are less frequently accessed, serves to increase DRAM utilization, and improve system efficiency. Intel IAA integration in zswap significantly enhances the speed of compression and decompression, providing a more efficient alternative to conventional software-based compression methods like lz4 and zstd.

Intel IAA offers a variety of compression modes, enabling system architects to tailor their configurations to achieve an optimal balance between fast compression and decompression speeds and reduced storage space, particularly within zswap framework. Additionally, recent developments in multi-size Transparent Huge Pages (mTHP) pave the way for more efficient hardware-accelerated memory management through enhanced parallel processing capabilities enabled by zswap Intel IAA batching. Future generations of Intel IAA are expected to deliver enhanced compression and decompression speed and improved compression ratios. These enhancements in the upcoming versions of Intel IAA are estimated to reduce the processing overhead associated with zswap, as well as to reduce the DRAM footprint, thereby enhancing the memory efficiency in the future datacenters significantly.

This paper has primarily focused on microbenchmark-level analysis to showcase the improvements achievable by offloading compression and decompression tasks to Intel IAA. We are looking forward to sharing more proof points at the workload level, beyond microbenchmarks as a follow-up to this paper. We would like to call upon the research community and industry practitioners to join us in further exploring and validating these findings at the workload level and provide feedback on how to improve this further in the next generation of Intel IAA. By collaborating on comprehensive workload-level studies, we can collectively demonstrate the full spectrum of enhancements that Intel IAA can offer, paving the way for its broader adoption and integration into advanced memory management systems.

References

1. Intel Corporation, "Intel In-Memory Analytics Accelerator (Intel IAA) Architecture Specification," 2023. https://cdrdv2.intel.com/v1/dl/getContent/780887.
2. Intel IAA Compression Accelerator Crypto Driver, [n.d.]. https://docs.kernel.org/next/driver-api/crypto/iaa/iaa-crypto.html.
3. Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. “Software-Defined Far Memory in Warehouse-Scale Computers,” International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) 2019.
4. Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. “TMO: Transparent Memory Offloading in Datacenters,” in proceedings of the 27th ASPLOS 2022, Lausanne, Switzerland, February 2022.
5. zswap, [n.d.]. https://www.kernel.org/doc/Documentation/vm/zswap.txt.
6. THP, [n.d.]. https://docs.kernel.org/admin-guide/mm/transhuge.html.
7. mTHP, [n.d.]. https://lwn.net/Articles/954094/.
8. Intel Corporation, Crypto: Add New Compression Modes for Zlib and Intel IAA, 2024. https://lore.kernel.org/all/cover.1710969449.git.andre.glover@linux.intel.com.
9. Intel Corporation, V4 Zswap Intel IAA Compress Batching, 2024. https://patchwork.kernel.org/project/linux-mm/cover/20241123070127.332773-1-kanchana.p.sridhar@intel.com/.
10. Intel Corporation, [RFC, V4, 00/13] Zswap Intel IAA Compress Batching, 2024. https://patchwork.kernel.org/project/linux-mm/cover/20241018064101.336232-1-kanchana.p.sridhar@intel.com/.
11. Intel Corporation, [RFC, V1, 0/7] Zswap Intel IAA Decompress Batching, 2024. https://patchwork.kernel.org/project/linux-mm/cover/20241018064805.336490-1-kanchana.p.sridhar@intel.com/.
12. Wanos Networks, “Silesia Corpus”, 2017. https://wanos.co/docs/docs/frequently-asked-questions/silesia-corpus/.
13. Lzo, [n.d.]. https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Oberhumer.
14. Lz4, [n.d.]. https://en.wikipedia.org/wiki/LZ4_(compression_algorithm).
15. Deflate, [n.d.]. https://en.wikipedia.org/wiki/Deflate.
16. Zstandard, [n. d.]. https://en.wikipedia.org/wiki/Zstandard.
17. Intel Corporation, "Intel ® Memory Usage Analyzer," 2023. https://github.com/intel/memory-usage-analyzer/tree/main/tests/madvise.
18. bpftrace, [n.d.]. https://github.com/bpftrace/bpftrace.
19. Barry Song, Chuanhua Han, and Tangquan Zheng, “mTHP swap-out and swap-in,” contributions of the Linux Plumbers Conference, 2024. https://lpc.events/event/18/contributions/1780/.
20. SPEC CPU 2017. https://www.spec.org/cpu2017/.
21. Timed Linux Kernel Compilation, [n.d.]. https://openbenchmarking.org/test/pts/build-linux-kernel-1.16.0.

Configurations

Server: 1-node, 2x Intel Xeon 6787P processor, 86 cores, 350W TDP, hyperthreading on, turbo on, total memory 1024 GB (16 x 64 GB, DDR5 6400 MT/s [6400 MT/s]), BIOS BHSDCRB1.IPC.0032.D56.2405160205, microcode 0x81000230, 1x I210 gigabit network connection, 1x 1.7T Micron_7450_MTFDKBG1T9TFR, CentOS Stream 9, 6.13.0-rc1-zswap_iaa_batching_release_v6.13-rc1_1-26-2025+. Software: madvise_test, Compiler: GCC 11.4.1, bpftrace: v0.21.1. Test by Intel at 2500 MHz fixed frequency as of Friday, January 10, 2025, 4:58:52 p.m. Eastern Standard Time.

Notices & Disclaimers