STYX: Exploiting SmartNIC Capability to Reduce Datacenter Memory Tax

Houxiang Ji, *University of Illinois Urbana-Champaign;* Mark Mansi, *University of Wisconsin-Madison;* Yan Sun, *University of Illinois Urbana-Champaign;* Yifan Yuan, *Intel Labs;* Jinghan Huang and Reese Kuper, *University of Illinois Urbana-Champaign;* Michael M. Swift, *University of Wisconsin-Madison;* Nam Sung Kim, *University of Illinois Urbana-Champaign*

https://www.usenix.org/conference/atc23/presentation/ji

This paper is included in the Proceedings of the 2023 USENIX Annual Technical Conference.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-35-9

Open access to the Proceedings of the 2023 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

STYX: Exploiting SmartNIC Capability to Reduce Datacenter Memory Tax

Houxiang Ji[*], Mark Mansi[†§], Yan Sun[*§], Yifan Yuan[††], Jinghan Huang[*], Reese Kuper[*], Michael M. Swift[†], and Nam Sung Kim[*]

[*]University of Illinois Urbana-Champaign [†]University of Wisconsin-Madison [††]Intel Labs

Abstract

Memory optimization kernel features, such as memory deduplication, are designed to improve the overall efficiency of systems like datacenter servers, and they have proven to be effective. However, when invoked, these kernel features notably disrupt the execution of applications, intensively consuming the server CPU's cycles and polluting its caches. To minimize such disruption, we propose STYX, a framework for offloading the intensive operations of these kernel features to SmartNIC (SNIC). STYX first RDMA-copies the server's memory regions, on which these kernel features intend to operate, to an SNIC's memory region, exploiting SNIC's RDMA capability. Subsequently, leveraging SNIC's (underutilized) compute capability, STYX makes the SNIC CPU perform the intensive operations of these kernel features. Lastly, STYX RDMA-copies their results back to a server's memory region, based on which it performs the remaining operations of the kernel features. To demonstrate the efficacy of STYX, we re-implement two memory optimization kernel features in Linux: (1) memory deduplication (`ksm`) and (2) compressed cache for swap pages (`zswap`), using the STYX framework. We then show that a system with STYX provides a 55–89% decrease in 99^{th}-percentile latency of co-running applications, compared to a system without STYX, while preserving the benefits of these kernel features.

1 Introduction

The modern OS offers various kernel features that can improve the overall utilization and/or performance of systems. Among them, memory optimization kernel features, such as memory deduplication, compressed cache for swap pages, and memory compaction to name a few, are devoted to utilizing the limited DRAM capacity of systems more efficiently. These kernel features are attractive to hyperscalers such as Google, Amazon, Meta, and Microsoft for two key reasons. First, DRAM technology scaling has notably slowed down, resulting in stagnant reduction in cost per GB of DRAM. Second, the DRAM capacity needed for datacenter servers has rapidly grown, not only for applications but also for software packages, profiling, logging, and other supporting functions required for efficient deployment of applications (*i.e.*, datacenter memory tax).

These kernel features have been extensively evaluated and enhanced [6, 19, 22, 27, 31, 40, 51]. They have proven to be effective, but they also incur notable deployment costs. Specifically, they are not frequently invoked, but they often perform memory- and/or CPU-intensive operations. They bring kilobytes to megabytes of usually cold data into the server CPU's caches and then make its cores intensively execute simple but repetitive operations on the data, often after disabling kernel preemption. As a result, they cause significant interference especially with co-running memory-intensive/latency-sensitive applications at the server CPU's cores and caches. This leads to a substantial increase in the high-percentile latency of the applications in datacenters (§3).

In this paper, we propose STYX, using SmarTNIC (SNIC) to efficiently manage the datacenter memorY taX (§4). Specifically, STYX makes use of two common Capabilities of SNIC: **(C1)** the RDMA capability to copy the server's memory regions, on which a kernel feature intends to intensively operate, to SNIC memory (① in Figure 1) and **(C2)** the compute capability to offload the intensive operations of kernel features from the expensive server CPU to the cheap SNIC CPU or accelerators (② in Figure 1). **(C1)** prevents the pollution

[§]Mansi and Sun have contributed equally as second authors.

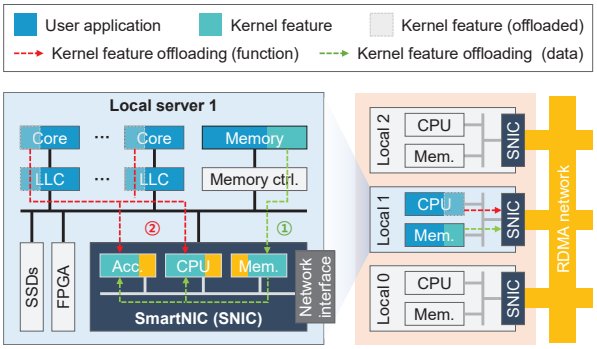


Figure 1: Overview of STYX framework.

of the server CPU's caches that stores application code and data, while **(C2)** frees the server CPU's cores from executing the intensive operations. As such, STYX offers a framework that allows us to deploy kernel features with significantly less disruption to the performance of co-running applications, without making the kernel features less effective.

We choose SNIC as our platform to offload intensive operations of memory optimization kernel features for two **R**easons. **(R1)** SNIC has already been deployed by hyperscalers (*e.g.*, Azure SNIC [15] and Amazon Nitro [4]) to minimize the datacenter tax [24] associated with executing network functions (*e.g.*, compression/decompression, encryption/decryption, and regular expression matching) at high rates. STYX reuses this existing capability without demanding novel or modified hardware. **(R2)** SNIC CPU's cores are not fully utilized as they are typically used to control accelerators in SNIC and orchestrate data transfers between the accelerators and the network controller. Note that STYX is built on a generic RDMA interface. As such, STYX also allows servers with standard RDMA NICs (RNICs) to seamlessly offload the intensive operations of kernel features to other servers with SNICs or RNICs.

To demonstrate the efficacy of STYX, we re-implement two memory optimization Linux kernel **F**eatures as examples: **(F1)** memory deduplication for virtual machines (VMs), also known as kernel same-page merging (`ksm` [14]) and **(F2)** compressed cache for swap pages (`zswap` [21]) (§5). Subsequently, we set up a server with an Intel Xeon CPU-based CPU and an NVIDIA BlueField-2 SNIC [20,37], and take `Redis` [41] driven by `YCSB` [11] as a representative memory-intensive/latency-sensitive application running on datacenter servers (§6). Lastly, we measure the 99^{th}-percentile (p99) response time (or latency) of `Redis` for various cases. (§7).

Specifically, we begin by evaluating the p99 latency values of `Redis` with systems deploying `ksm` (denoted as `sys-ksm`) and `zswap` (`sys-zswap`), and compare them with those of a system that deploys no memory optimization kernel feature (`sys-no-mo`). We show that `sys-ksm` and `sys-zswap` increase the p99 latency values by 4.8–9.7× and 8.1–11.0×, respectively, compared to `sys-no-mo`. Then, we evaluate the p99 latency values of `Redis` with systems deploying STYX-based `ksm` (`sys-styx-ksm`) and `zswap` (`sys-styx-zswap`), and demonstrate that they reduce the p99 latency values to 1.0–1.1× and 1.8–3.8×, respectively, compared to `sys-no-mo`, while preserving the benefits of `ksm` and `zswap`. Finally, we assess the impact of running the STYX-based kernel features on the performance of functions accelerated by SNIC. We choose regular expression matching (`rem`) as a representative function accelerated by a dedicated accelerator in the NVIDIA BlueField-2 SNIC. Even when offloading the intensive operations of `ksm` and `zswap` to the SNIC, STYX increases the p99 latency value of `rem` by only 1.3%.

2 Background

2.1 Memory Optimization Kernel Features

In this section, we provide an overview of two memory optimization kernel features in Linux: `ksm` and `zswap`. Other operating systems, such as Windows, also offer similar features like page combining [7,34] and memory compression [54]).

`ksm`. It is a memory deduplication feature in Linux. It is commonly used with kernel-based virtual machine (KVM) to quickly consolidate more VMs within a given physical memory capacity [35], by sharing pages with the same content among multiple VMs (*e.g.*, pages storing code for OS and common libraries). As it allows for more efficient storage of common data in cache or memory, it also notably improves performance for certain applications and operating systems [47]. It periodically and incrementally scans pages of two or more running processes to identify those with the same memory contents. Then, it merges those identical pages into a single physical copy, updates their page table entries with a copy-on-write (CoW) attribute, and reclaims the memory space previously used by the pages. Both the overhead and benefit of `ksm` are determined by the number of scanned pages per invocation of `ksm`, the frequency of scanning pages, and the maximum number of merged pages.

`zswap`. It serves as a compression backend for the Linux swap daemon (`kswapd`) which includes synchronous direct and asynchronous background paths. `kswapd` takes

the synchronous direct path when the memory allocator fails to allocate pages due to a lack of free memory space. This requires `kswapd` to immediately swap out the least recently used (LRU) pages to the backing swap device. `kswapd` takes the asynchronous background path when the amount of free memory space drops below the `page_low` watermark. This makes `kswapd` begin to swap out pages from the inactive page list, and continues until the amount of free memory space exceeds the `page_high` watermark.

When deployed, `zswap` intercepts the pages from both the paths above, compresses them, and places them in a dynamically allocated memory pool in DRAM (*i.e.*, `zpool`). Meanwhile, when the size of `zpool` reaches the `max_pool_percent` threshold, `zswap` wakes up and takes the LRU page from `zpool`, decompresses and relocates it to the backing swap device, and frees the compressed page from `zpool`. To serve a page fault, `zswap` first checks `zpool` to find whether the page is evicted to the backing swap device. If the page is found in `zpool`, it is simply decompressed and returned by `zswap`. Otherwise, the system follows the standard process for swapping in a page from the backing swap device.

Since `zswap` can notably reduce the need for accessing the slow backing swap device, it may improve the overall performance of the system; the page decompression on the synchronous direct path is part of the performance-critical path for handling page faults, but it is typically faster than retrieving pages from the backing swap device. As such, `zswap` has been evaluated by Google [27] and Meta [51] for potential deployment in production systems.

2.2 SNIC and RDMA

Recently, various SNICs have been developed to offload functions common in network applications such as security, compression, and network function virtualization as part of an effort to reduce the datacenter network processing tax [24]. Generally, an SNIC integrates a traditional network interface controller (NIC) with a CPU, ASIC- and/or FPGA-based accelerators, and memory and IO subsystems. For example, an NVIDIA BlueField-2 SNIC [37] consists of 8 ARM CPU cores with private L1 and shared L2 caches, a cache-coherent on-chip interconnect, DRAM and PCIe controllers, onboard DRAM as main memory, and ASIC-based accelerators for regular expression matching, encryption, and compression. AMD/Xilinx SN1000 [5] integrates an FPGA fabric with an SNIC similar to the NVIDIA BlueField-2 SNIC in a single chip. An SNIC is itself a complete system, recognized as an independent node.

RDMA is supported by most SNICs and standard NICs used in datacenters. As it allows a client to directly access the memory of servers at low latency and high bandwidth, it is now widely used by datacenters [16, 18, 28, 53, 56]. Additionally, an SNIC in a server can access the server's local memory through RDMA. RNIC supports two operating modes: two-sided RDMA and one-sided RDMA. The two-sided RDMA reduces packet processing overhead by delivering requests (or data) from a client directly to server's memory for application processing. One-sided RDMA allows the client to completely bypass the server's CPU and directly read from or write to the server's memory.

3 Impact of Running Kernel Features on Application Performance

A body of prior work has demonstrated that `ksm` and `zswap` have proven to be effective in improving the overall performance of systems [8, 19, 22, 27, 35, 47, 49]. Nonetheless, such benefits come with costs that have often discouraged system administrators from widely deploying them in datacenters. In this section, we analyze the costs associated with deploying `zswap` as an example.

Figure 2 shows a snapshot of (a) consumed CPU cycles, (b) last-level cache (LLC) miss rate, and (c) response latency of `Redis` that are captured when `zswap`-enabled `kswapd` is invoked. See Section 6 for the detailed evaluation setup and methodology. First, `zswap` increases the consumed CPU cycles by 26.4% during the captured time period (Figure 2(a)). Second, it increases the average LLC miss rate from 4.4% to 49.8% (Figure 2(b)). Lastly, it increases the mean, median, 3^{rd} quartile, and p99 latency values of `Redis` serving the same number of requests by 1.5×, 1.2×, 1.8×, and 2.1× respectively (Figure 2(c)).

Although we show the plots only for `zswap` in Figure 2, we observe that `ksm` also exhibits a similar impact on CPU cycle consumption, LLC miss rate and response latency of `Redis`. Subsequently, we discuss the primary sources of such increases in both `zswap` and `ksm` in detail.

`ksm`. It periodically scans pages and calculates a 32-bit checksum for each scanned page to more efficiently identify candidate pages for future merging. Among the candidate pages, `ksm` picks two pages and performs a byte-by-byte comparison to determine if the two pages can be merged. Both the checksum calculation and byte-by-byte comparisons of pages are CPU- and memory-intensive as they bring around 400MB of data to caches and CPU cores (often from DRAM) and do numerous

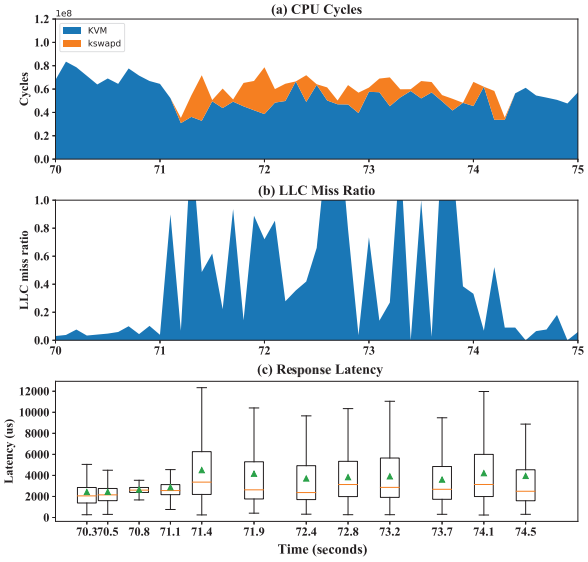


Figure 2: A snapshot of (a) consumed CPU cycles, (b) LLC miss ratio, and (c) response latency before and after invoking `kswapd` while running `Redis`. We gather the response latency values for every 1110 requests and plot them using a box, a vertical line, a triangle, and a horizontal line. The box, triangle, and horizontal line respectively represent the 1^{st} to 3^{rd} quartile range, mean, and median of the latency values for 1110 requests.

arithmetic and comparison operations for that amount of data per invocation of `ksm`.

`zswap`. When invoked, it performs compression and decompression, which are highly compute-intensive and thus significantly consume CPU cycles [23, 39]. For instance, in Figure 2, approximately 45,000 pages are compressed in only 5 seconds, consuming roughly 25–50% cycles of the server CPU's core while `zswap` is running. These pages represent 175MB (*i.e.*, 45,000 × 4KB) of *cold* data that is brought into the server CPU's LLC. Since they are unlikely to be used soon, they end up polluting the server CPU's LLC. Later, when compressed pages in `zpool` are evicted to the backing swap device (§2.1), the pages are decompressed and re-pollutes the server CPU's LLC with *cold* data again.

4 STYX Framework

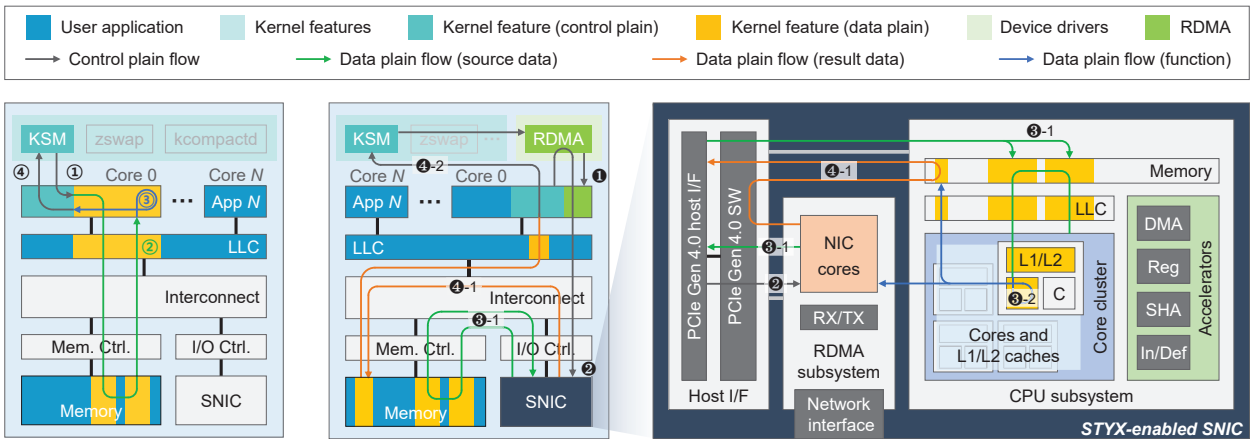
In Section 3, we demonstrated that widely used memory optimization kernel features are often CPU- and memory-intensive, and significantly interfere with co-running applications at the server CPU's cores and caches. To reap the benefits of deploying these kernel features while minimizing interference with the co-running applications, we propose STYX. In this section, we provide an overview of STYX and describe its workflow as a general framework. Subsequently, in Section 5, we delve into the usage of STYX for offloading CPU- and memory-intensive operations of `ksm` and `zswap` to SNIC.

4.1 Overview

We design STYX based on a key observation that memory optimization kernel features, similar to network applications, can be decomposed into control and data planes. We then assign the most CPU- and memory-intensive operations of the kernel features to the data plane, and have the SNIC's CPU handle the data plane. This facilitates STYX to significantly reduce the costs of deploying the kernel features without compromising their benefits.

Figure 3a depicts an abstracted workflow of conventional memory optimization kernel features. When invoked, a kernel feature running on the server's CPU ① determines one or more memory regions that it intends to operate on; ② copies the memory regions to the server CPU's caches; ③ operates on the memory regions (*e.g.*, comparing two pages using `memcmp` in the case of `ksm`); and ④ makes a decision for the next step (*e.g.*, whether merge two pages or not in the case of `ksm`) based on the result of ③. STYX considers ① and ④ as the control plane, while it assigns ② and ③ to the data plane.

In a conventional system, the server's CPU performs both control- and data-plane operations of a given kernel feature. As discussed in Section 3, the data-plane operations significantly pollute the server CPU's caches and intensively consume the server CPU's valuable cycles. In contrast, in an STYX-based system, the SNIC's CPU performs data-plane operations instead, while the server's CPU still performs the control-plane operations. Specifically, the control plane on the server's CPU first determines memory regions that the data plane on the SNIC's CPU will operate on, RDMA-copies the memory regions from the server's memory to the SNIC's memory, and then makes the data plane on the SNIC's CPU operate on the memory regions. After the data plane on the SNIC's CPU completes operations on the RDMA-copied memory regions, it RDMA-copies the results back to the server's designated memory region. Finally, as the conventional system does, the control plane on the server's CPU decides the next step based on the results.



(a) Conventional workflow

(b) Workflow of kernel feature offloading via STYX framework

Figure 3: Workflow of a memory optimization kernel feature in conventional and STYX-enabled systems. For brevity, only the cache and memory regions of the kernel feature's data plane are illustrated.

```
struct STYX_descriptor {
  func_type func;
  // memory regions RDMA-copied to SNIC
  u64* addrs; // starting addresses
  int* lens;  // lengths
  int  num;   // number
  // RDMA resources
  void* completion_queue;
  void* send_queue;
  void* recv_queue;
  ...
};
```

Listing 1: Data structure of `STYX_descriptor`.

4.2 Workflow

Figure 3b illustrates a high-level workflow of STYX framework, which is built on top of kernel-space RDMA verbs. It comprises four steps: ❶ setup, ❷ submission, ❸ remote execution, and ❹ completion.

❶ **Setup.** STYX first determines the functions that will be offloaded to SNIC. A function comprises data-plane operations within a specific kernel feature. For instance, `memcmp`, which performs a byte-by-byte comparison of two memory regions, can be such a function in `ksm`. Next, STYX establishes a communication interface between the server and the SNIC by setting up RDMA connections between them. Specifically, STYX allocates necessary RDMA resources, such as completion and work queues, in the kernel space on the server and the user space on the SNIC (❶ in Figure 3(b)). To avoid contention for the RDMA resources among functions, STYX sets up one RDMA connection for each function. Finally, for each function, STYX creates two descriptors, each called `STYX_descriptor`, on the server and the SNIC, respectively, and then associates the descriptors with the corresponding RDMA connection. `STYX_descriptor` is a data structure described in Listing 1, which stores the following information: a function identifier, pointers to the starting addresses of memory regions, the lengths and number of the memory regions, and pointers to the RDMA resources. It is designed to provide a uniform and generic interface for a server to provide necessary information for an SNIC that will execute a specific function on behalf of the server.

❷ **Submission.** Before a kernel feature executes a function, STYX on the server updates the descriptor associated with the function with the starting addresses and lengths of memory regions that it has determined to work on. Next, STYX uses two-sided RDMA to offload the function. Specifically, STYX on the server sends an RDMA `send` request based on the updated descriptor, making the SNIC RDMA-copy the memory regions from the server's memory to the SNIC's memory (❷ in Figure 3(b)). Lastly, STYX calls RDMA `recv`. This suspends the execution of the kernel feature until the SNIC sends the results of the function back to the server through RDMA `send`, and allows the kernel feature to yield the server CPU's core to other application processes.

Alternatively, STYX can employ one-sided RDMA. In this approach, STYX on the server posts the updated starting addresses and lengths to the server's designated memory region (*i.e.*, the descriptor on the server) registered to the SNIC. At the same time, STYX on the SNIC continuously polls the memory region using RDMA `read`. Once STYX on the SNIC obtains the updated addresses and

lengths, it proceeds with RDMA-copying the memory regions from the server's memory to the SNIC's memory using RDMA `read`. This one-side RDMA-based approach can allow STYX to free up the server's CPU for other application processes faster than the two-sided RMDA-based approach. However, it requires the SNIC's CPU to poll the registered memory region using RDMA `read` requests, resulting in consuming PCIe interconnect bandwidth and SNIC CPU's cycles.

❸ **Remote Execution.** After receiving an RDMA `send` request from the server, the SNIC starts to RDMA-copy the server's memory regions to the SNIC's memory region pointed by the RDMA `recv` request using a single scatter-gather transfer (❸-1 in Figure 3(b)). The completion of serving the RDMA `recv` request wakes up STYX on the SNIC to execute a function associated with the RDMA connection. Subsequently, STYX on the SNIC creates a worker thread to execute the function which operates on the RDMA-copied memory region (❸-2 in Figure 3(b)). Note that executing such a function may interfere with network applications co-running on the SNIC. Nonetheless, the SNIC CPU serves as a control plane for network applications executed by the SNIC accelerators (§1), and STYX utilizes unused or underutilized SNIC CPU cores. Therefore, STYX negligibly affects the performance of network applications running on the SNIC (§7.6).

❹ **Completion.** After completing the remote execution of the function, STYX on the SNIC posts an RDMA `send` request to send the results (*e.g.*, the checksum of a page in the case of `ksm`) to STYX on the server (❹-1 in Figure 3(b)). After STYX on the server receives the result through RDMA `recv` previously invoked at ❷, it makes the kernel feature resume the execution and read the results from the server memory (❹-2 in Figure 3(b)). At the same time, STYX on the SNIC invokes RDMA `recv`, which makes STYX on the SNIC sleep until it receives RDMA `send` from the server.

Similar to what is discussed in ❷, STYX on the SNIC can send the result to the server through one-sided RDMA `write` to a memory region registered on the server. However, this demands the server's CPU to keep polling the memory region until the completion signal is detected. This not only wastes the server CPU's cycles but also prevents the kernel feature from yielding the server CPU to application processes.

5 Offloading Kernel Features with STYX

In Section 4, we provided a high-level workflow of STYX as a general framework for offloading intensive operations (or functions) of memory optimization kernel features to SNIC. In this section, we will further elaborate on implementations of STYX-based `ksm` and `zswap`, as well as optimizations tailored for each kernel feature.

5.1 `ksm`

`ksm` is a memory-deduplication feature that merges pages with the same content. We identify the two most resource-intensive functions to offload to SNIC: (1) page comparison and (2) checksum calculation. The page comparison gives the relative address of the first byte that differs in two pages. This is used to determine whether the pages can be merged and the relative order of the two pages. The checksum calculation provides a word-size hash value calculated based on the page content and indicates whether a page has been changed between scan passes by the `ksm` daemon. Algorithm 1 describes one pass of STYX-based `ksm` where the page comparison and the checksum calculation are performed by `STYX_compare` and `STYX_checksum`, respectively.

Since we offload two functions to SNIC, STYX creates two RDMA connections and two descriptors during the setup phase (❶). `STYX_compare` requires two for the number of memory regions as it compares two pages (or memory regions). On the other hand, since

Algorithm 1: `ksm` with STYX offloading

```
1  Init stable_tree and unstable_tree
2  while pages for this pass > 0 do
3      cand_page = next page in the pass
4      for page ∈ stable_tree do
5          if STYX_compare(cand_page, page) then
6              merge(cand_page, page)
7              goto line 2
8      new_cksum = STYX_checksum(cand_page)
9      old_cksum = cand_page.cksum
10     cand_page.cksum = new_cksum
11     if new_cksum == old_cksum then
12         for page ∈ unstable_tree do
13             if STYX_compare(cand_page, page)
                 then
14                 merged_page = merge(cand_page,
                     page)
15                 cow_protect(merged_page)
16                 remove(page, unstable_tree)
17                 insert(merged_page, stable_tree)
18                 goto line 2
19         insert(cand_page, unstable_tree)
20 End of pass, sleep()
```

STYX_checksum calculates a checksum for a given page, it needs one for the number of memory regions. For the length of a memory region, both STYX_compare and STYX_checksum use the size of a page in byte. During the submission phase (❷), STYX updates the starting addresses of the memory regions in the descriptor for STYX_compare with the pointers to two chosen pages. For STYX_checksum, STYX takes the pointer to a selected page and updates the descriptor accordingly. Subsequently, STYX RDMA-copies these memory regions from the server memory to the SNIC memory. During the remote execution phase (❸), STYX on the SNIC performs a byte-by-byte comparison and an xxHash-based checksum calculation [10] on the RDMA-copied page(s). It then returns the relative address of the first byte that differs in the two pages (STYX_compare) and the word-size checksum (STYX_checksum), respectively, to STYX on the server. Receiving the results from STYX on the SNIC, STYX on the server decides whether it will merge the two pages or not, and updates the checksum value for the scanned page during the completion phase (❹).

5.2 zswap

zswap serves as a compression backend for kswapd, and it was incorporated into the Linux kernel starting from version 3.5. As described in Section 2.1, there are synchronous direct and asynchronous background paths.

Algorithm 2: kswapd with STYX offloading

```
1  while kswapd_enabled do
2      if free_page < page_low then
3          kswapd_running = true;
4          while kswapd_running do
5              page = page_to_swap_out()
6              if zpool > max_zpool_size then
7                  if STYX_decompression(LRU_page,
                     dst) fails then
8                      kernel_decompress(LRU_page,
                         dst);
9                  write_to_backing_swap_device(dst);
10                 free_zpool_space(LRU_page);
11             if STYX_compression(page, dst) fails
                 then
12                 kernel_compress(page, dst);
13             write_to_zpool(dst);
14             if free_page > page_high then
15                 kswapd_running = false;
16     else
17         kswapd_sleep();
```

STYX is capable of offloading functions from both paths to SNIC. Nonetheless, as an optimization, we choose to offload only the asynchronous background path which is taken when (1) the amount of free memory space falls below the page_low watermark, and (2) the size of zpool reaches the max_pool_percent threshold. Specifically, when (1) happens, STYX-based zswap makes SNIC compress pages and place the compressed pages in zpool until the amount of free memory space is above the page_high watermark. When (2) occurs, it makes SNIC decompress the LRU page from zpool and relocate it to the backing swap device. We propose this optimization because the latency involved in RDMA-copying pages to the SNIC memory over the PCIe interconnects (*i.e.*, $\sim 5\mu$s) may slow down the time-sensitive synchronous direct path, and degrade overall system performance. Algorithm 2 describes kswapd modified to support STYX-based zswap where the page compression and decompression are performed by STYX_compression and STYX_decompression, respectively.

Since we offload two functions to SNIC, the setup and submission phases for STYX-based zswap are exactly the same as STYX-based ksm except that zswap has only one memory region to RDMA-copy to the SNIC memory for both the functions. Finally, after the remote execution of STYX_compression and STYX_decompression, STYX on the SNIC will return the compressed and decompressed pages, respectively, to STYX on the server.

6 Methodology and Implementation

System Setup. We set up a server with an Intel Xeon Gold CPU and an NVIDIA BlueField-2 SNIC. The detailed hardware and software configurations of the server are listed in Table 1. Note that we lock the CPU frequency at 2.1 GHz and disable hyper-threading (HT) for more consistent performance over multiple measurement runs. VMs are pinned to specific CPU cores to reduce performance variations and interference caused by dynamic voltage/frequency scaling (DVFS) [12], HT [32], and VM scheduling.

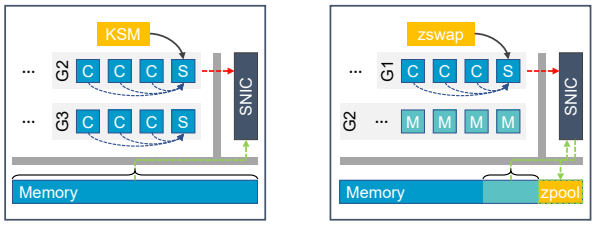
Workload. We run Redis [41] with Yahoo! Cloud Serving Benchmark (YCSB) [11] on the system. Redis is an in-memory data store and is used as a distributed, in-memory key–value database, cache, and message broker, with an optional durability feature. YCSB is a benchmarking framework to evaluate the performance of various in-memory key-value stores. It comes with four workloads: (a) update heavy, (b) read heavy, (c) read only, and (d) read latest that consist of (a) 50% read and 50% update, (b) 95% read and 5% update, (c) 100% read, and

Table 1: Hardware and Software configurations.

Intel Xeon 6138P Server
CPU: 16 Skylake cores @ 2.1GHz w/ HT disabled, 32KB L1, 1MB L2, and 1MB L3 caches per core **Memory:** 5-Ch. w/ 5 16GB DDR4-2666 DRAM modules **OS:** Ubuntu 18.04.6 LTS, Linux kernel 5.4
NVIDIA BlueFeild-2 SNIC
Network: ConnectX-6 Dx w/ two 25 Gbps Ethernet ports , RDMA over converged Ethernet V2 **CPU:** 8 ARM A72 cores @ 2.5GHz, 640 KB L1 per core, 4 MB L2 caches per 2 cores, and 6 MB L3 cache **Memory:** 1 Ch. w/ 16GB DDR4-1600 DRAM module **Accelerators**: regular expression matching, compression, and cryptography **OS**: Ubuntu 20.04.2 LTS, Linux kernel 5.4
Kernel Feature
ksm: `sleep_between_scan`=20ms, `free_mem_thres`=20 `pages_to_scan` ∈ [64, 1250] # adjusted by *ksmtuned* **zswap**: `compressor_type` = lzo, `max_pool_percent` = 20 `zpool_management` = zbud
Virtual Machine
Hypervisor: QEMU-KVM 2.11.1 **VM**: Ubuntu Cloud 18.0, 1 Core, 4GB memory

(d) 95% read and 5% insert, respectively.

Methodology. Figure 4 depicts the evaluation environments for `ksm` and `zswap`. `ksm` aims to reduce memory usage in virtualized environments where multiple VMs are running similar workloads. We set up 16 VMs and pin each VM to a physical core. Then, we organize the VMs into 4 groups, each comprising 3 VMs for `Redis` clients and 1 VM for a `Redis` server. To trigger `zswap`, we set up a background workload designed to allocate



(a) Setup for **`ksm`** (b) Setup for **`zswap`**

Figure 4: Experimental setup for evaluating STYX-based `ksm` and `zswap`. The blue-color boxes indicate the Redis server ('S') and client ('C'). The lighter-blue blocks ('M') represent the cores running a background workload.

and free memory space periodically. We need such a background workload because `Redis` is in-memory data store, and it should be configured not to incur any page faults. Otherwise, p99 latency is dominated by handling page faults. We use `cgroup` [1] to protect the pages used by `Redis` from being swapped out. In the experimental setup for `zswap`, `Redis` servers and clients run on the physical cores directly without VMs.

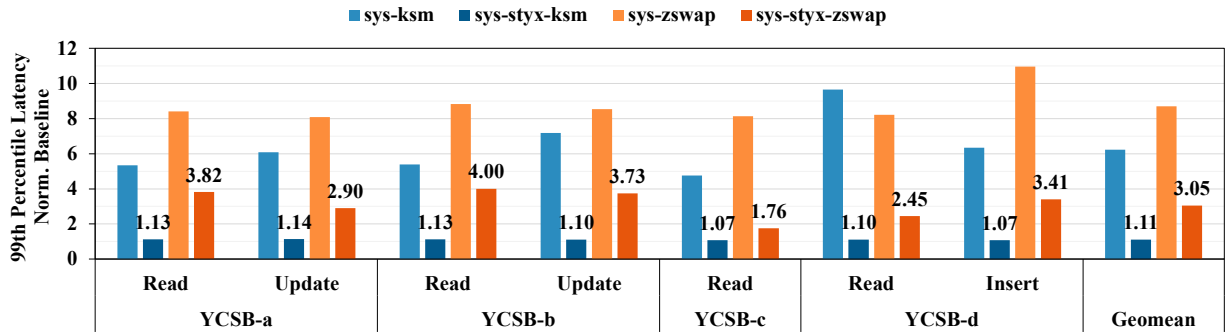
7 Evaluation

7.1 Latency

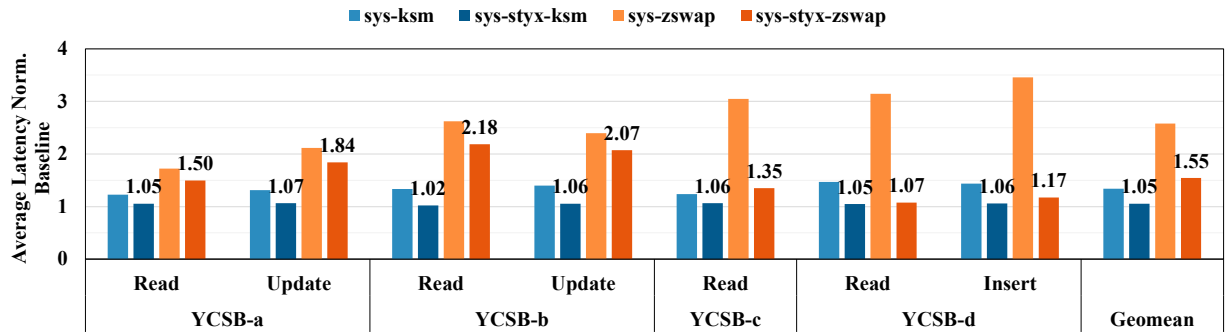
In this section, we choose p99 latency as a key performance metric for our evaluation because many important datacenter applications need to meet certain high-percentile latency requirements [36, 38]. Figure 5a shows the p99 latency values of `Redis` on systems that deploy `ksm` (`sys-ksm`), `zswap` (`sys-zswap`), STYX-based `ksm` (`sys-styx-ksm`), and STYX-based `zswap` (`sys-styx-zswap`), normalized to those of a system without deploying any memory optimization kernel feature (`sys-no-mo`). Overall, it demonstrates that STYX can significantly reduce the p99 latency increased by deploying `ksm` and `zswap`.

Specifically, on average (geometric mean), `sys-ksm` and `sys-zswap` give 6.24× and 8.70× higher p99 latency values than `sys-no-mo`, respectively. In contrast, `sys-styx-ksm` and `sys-styx-zswap` offer 1.11× and 3.05× higher p99 latency values than `sys-no-mo`, respectively. That is, `sys-styx-ksm` and `sys-styx-zswap` reduce the p99 latency increase by 5.62× and 2.85×, compared to `sys-ksm` and `sys-zswap`, respectively. Note that `sys-styx-ksm` offloads most of the intensive operations to SNIC, practically eliminating the p99 latency increase of `sys-styx-ksm`. On the other hand, `sys-styx-zswap` offloads only the intensive operations of the asynchronous background path to SNIC. That is, the intensive operations of the synchronous direct path still affect the p99 latency of `Redis`, contributing to more than 3× increase in p99 latency.

In addition, Figure 5b shows the average latency values of `Redis` on those systems. The average latency of `Redis` is also an important performance metric, as it is inversely proportional to the throughput. Note that `Redis` often throttles serving requests to prevent response latency from increasing too much when the system receives more requests than it can handle efficiently [3]. On average, `sys-ksm` and `sys-zswap` give 1.34× and 2.58× higher average latency than `sys-no-mo`, respectively. In contrast, `sys-styx-ksm` and `sys-styx-zswap`



(a) p99 latency



(b) Average latency

Figure 5: Latency values of `Redis` with `YCSB` workloads running on `sys-ksm`, `sys-styx-ksm`, `sys-zswap` and `sys-styx-zswap`, normalized to `sys-no-mo`.

offers only 1.05× and 1.55×, higher average latency than `sys-no-mo`, respectively. That is, `sys-styx-ksm` and `sys-styx-zswap` reduce the average latency increase by 22% and 40%, compared to `sys-ksm` and `sys-zswap`, respectively.

7.2 LLC Miss Rates

To analyze how STYX reduces the negative impact of deploying `ksm` and `zswap` on p99 latency of `Redis`, we measure the LLC miss rates of the server CPU every 1 second while evaluating `sys-no-mo`, `sys-styx-*` and `sys-*`. We report the p99 LLC miss rates from approximately 160 1s intervals instead of the average LLC miss rates, because intervals with high LLC miss rates are likely responsible for p99 latency values of `Redis`.

Table 2 summarizes the p99 LLC miss rates across all `YCSB` workloads at their highest throughput values that the systems can provide. This shows that the memory optimization kernel features can significantly increase the p99 LLC miss rates, bringing large amounts of cold data into the server CPU's caches when invoked. Specifically, `sys-ksm` and `sys-zswap` give 7.33× and 1.70× higher p99 LLC miss rates than `sys-no-mo`, respectively, in some intervals. In contrast, `sys-styx-ksm` and `sys-styx-zswap` offer 3.78× and 1.28× higher p99 LLC miss rates than `sys-no-mo`, respectively. That is, `sys-styx-ksm` and `sys-styx-zswap` reduce the p99 LLC miss rate increase by 48% and 25%, respectively. Such a benefit comes from the fact that `sys-styx-ksm` and `sys-styx-zswap` RDMA-copy the server's memory regions that `ksm` and `zswap` work on to the SNIC memory instead of the server CPU's caches.

Table 2: p99 LLC miss rates of three systems (`sys-no-mo`, `sys-*`, and `sys-styx-*`) for different `YCSB` workloads.

	a	b	c	d	GeoMean
`no-mo`	9.7%	7.1%	7.3%	8.0%	8.0%
`ksm`	60.4%	56.9%	59.8%	57.5%	58.6%
`styx-ksm`	40.4%	26.5%	27.2%	28.4%	30.2%
`no-mo`	18.5%	21.4%	22.2%	21.7%	20.9%
`zswap`	34.7%	41.3%	33.9%	32.6%	35.5%
`styx-zswap`	25.1%	27.8%	29.8%	24.7%	26.8%

Table 3: CPU utilization of two systems (sys-* and sys-styx-*) for different YCSB workloads.

	a	b	c	d	GeoMean
ksm	26.0%	26.0%	25.9%	25.9%	26.0%
styx-ksm	7.1%	7.3%	6.8%	6.7%	7.0%
zswap	23.5%	19.8%	20.5%	17.8%	20.3%
styx-zswap	13.0%	8.9%	11.8%	8.4%	10.4%

Note that the p99 LLC miss rate of sys-no-mo for ksm is much lower than that of zswap. This is because of the background workload designed to incur page faults for zswap. Besides, in the case of zswap, STYX offloads only intensive operations of the asynchronous background path to SNIC. That is, intensive operations of the synchronous direct path still pollute the server CPU caches when the background workload incurs page faults. Lastly, even in the case of ksm, STYX does not offload all the operations, either.

7.3 CPU Cycle Consumption

In addition to reducing cache pollution, STYX conserves the server CPU's cycles, as it offloads the CPU-intensive operations to the SNIC CPU. To assess the impact of running ksm and zswap on consuming the server CPU's cycles, we identify the number of 1-millisecond intervals that both a kernel feature and Redis co-run on a server CPU's core while measuring the number of server CPU's core cycles consumed by the kernel feature during these intervals. To get the average CPU utilization shown in Table 3, we sum up all the server CPU's core cycles consumed by the kernel feature and then divide it by the total number of the server CPU's core cycles during the intervals in which the kernel feature and Redis co-run.

Table 3 shows that STYX considerably reduces the consumption of the server CPU's core cycles. On average, sys-styx-ksm and sys-styx-zswap reduce the server CPU's core cycles consumed by ksm and zswap from 26% to 7% and from 20% to 10%, respectively. The server CPU's cycles saved by offloading intensive operations of ksm and zswap to SNIC can be used for Redis, which minimizes disruption of Redis operations during these co-running intervals.

7.4 Offloading Latency

Table 4 shows the breakdown of the latency values of ksm and zswap functions offloaded to the NVIDIA BlueField-2 SNIC. By analyzing the breakdown, we can identify the offloading steps that may provide further optimization opportunities. We do not include the latency breakdown of the setup step (❶), because it is called once and the latency cost is amortized over time. The submission step (❷) takes ∼0.5 microseconds, *e.g.*, only ∼2% of the total latency of offloading the functions to SNIC. This latency primarily comes from the time to send an RDMA send request to SNIC.

Table 4: The breakdown of the offloading latency values of each function and the percentage values of function execution time in total kernel feature execution time per invocation. f1 and f2 correspond to comparison and checksum of ksm, respectively. For f1, we measure the latency of comparing two pages with the same content, which gives the longest latency. f3 and f4 represent compression and decompression of zswap, respectively.

		f1	f2	f3	f4
styx-ksm/zswap	❷ (μs)	0.51	0.49	0.52	0.49
	❸ (μs)	14.61	12.93	20.26	16.97
	❹ (μs)	5.04	4.97	5.21	5.13
	% in Tot.	57.2	32.3	25.4	8.3
ksm/zswap	% in Tot.	36.9	19.5	12.3	6.1

The remote execution step (❸) takes a total of 13–20 microseconds depending on the offloaded functions. Specifically, it spends 5–7 microseconds for RDMA-copying memory regions from the server memory to the SNIC memory. It spends the remaining 8–15 microseconds for the SNIC CPU to execute the functions of ksm and zswap. As the RDMA-copy latency is responsible for a dominant fraction of the remote execution step, we may consider making SNIC's on-chip accelerators execute these functions to further reduce the remote execution latency. However, the accelerators in the NVIDIA BlueField-2 SNIC are connected to the on-chip PCIe interconnect. Thus, it still takes a notable amount of time to offload functions from the SNIC CPU to the accelerators (*e.g.*, ∼7 milliseconds for the compression accelerator), which involves another DMA transfer within SNIC.

The completion step (❹) consumes a notable amount of time spent by interrupt handling and process context switching between application and kernel feature processes. During this step, the SNIC CPU remains active after submitting the RDMA request until receiving an acknowledgment from the server CPU. This waiting time is included as part of the latency of the completion step.

7.5 Effectiveness of Kernel Features

It takes a longer time to offload functions of the kernel features to SNIC than to run them directly on the server CPU. This in turn increases the overall execution time

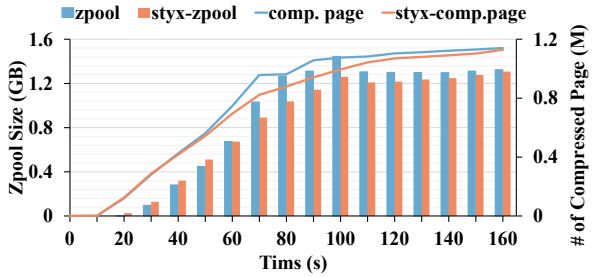


Figure 6: The number of the compressed pages (line) and the size of `zpool` (area) over time in `sys-zswap` and `sys-styx-zswap`.

of the kernel features and may affect the kernel feature's effectiveness. We take `zswap` as an example and plot: (1) the number of compressed pages and (2) the size of `zpool` over time in Figures 6. Although the compression operation performed by the SNIC in STYX does result in longer latency, the overall effectiveness and performance of `zswap` are not greatly affected. First, we observe that the size of `zpool` is not directly proportional to the number of compressed pages. This is because the compression ratio of pages varies across pages. Figure 6 shows that STYX-based `zswap` can provide a comparable compression rate close to what `zswap` can, based on the number of compressed pages. Figure 6 also shows that with STYX, the rate of growth of `zpool` is only 2% lower than that of the standard `zswap` implementation. This is explained by run-to-run performance variations.

Note that both `zswap` saves the disk I/O as it compresses the pages into swap memory and avoids the direct swap out. We monitor the disk I/O at the runtime of the workloads and see the overall disk I/O consumption is 39% lower when `zswap` is enabled. The disk I/O reduction is attributed to the swap-out compressing cache in `zswap`. Upon deploying STYX, the disk I/O throughput remains 35% lower than the `sys-no-mo` case where no `zswap` feature runs. We see that both `sys-zswap` and `sys-STYX-zswap` achieve comparable disk I/O saving. In summary, STYX-based `zswap` preserves the benefits of `zswap` while notably reducing the disruption of co-running applications caused by `zswap`.

7.6 SNIC Application Performance

Since SNIC has its own designated roles, we need to analyze the impact of running STYX on performance of applications that SNIC is designed to accelerate. Specifically, we choose regular expression matching (`rem`) as an application; since it has been extensively used for various network security applications and the NVIDIA BlueField-2 SNIC provides a dedicated accelerator. Table 1 gives an overview of the NVIDIA BlueField-2 SNIC. We take the `DPDK-Pktgen` tool [52] on a remote server to send network packets to the SNIC. We configure the SNIC and `DPDK-Pktgen` to exercise the maximum 25Gbps network bandwidth, and vary the size of packets to observe the utilization of the SNIC CPU's cores.

As the packet size increased from 128 bytes to 1024 bytes, the number of SNIC CPU's cores required to achieve the maximum `rem` throughput decreases from 5 to 1. Smaller packet sizes demand more packets per second to use the full network bandwidth, which in turn requires more cores. In our current implementation, STYX on the SNIC utilizes only ∼30% of a single core of the SNIC CPU, which is obtained after running the most CPU-intensive function, page compression in `zswap`. That is, the SNIC can handle STYX with little impact on the performance of `rem`. Our experiment shows that the SNIC running only `rem` gives a p99 latency of 13.83 microseconds, while the SNIC running both `rem` and STYX offers a p99 latency of 13.85 microseconds. It also confirms that the SNIC running both `rem` and STYX do not decrease the maximum throughput of `rem`.

8 Related Work

Exploring the improved compute efficiency of heterogeneous computing, many past proposals have focused on offloading CPU-intensive operations of user-space programs from the CPU to xPUs and FPGA. In contrast, relatively less attention has been given to offloading CPU- and memory-intensive operations of kernel-space programs from the CPU so far. Nevertheless, it has become increasingly important, especially for datacenter servers to cost-effectively reduce the high datacenter tax.

Some past proposals aim to make kernel features run more efficiently. One pioneering proposal is `Pageforge` [45], which implements a hardware mechanism in the memory controller to execute the page comparison operations of `ksm`. It also exploits the Error Correcting Code (ECC) engine in the memory controller to perform the checksum calculation operations of `ksm`. Although `Pageforge` is effective, it requires hardware changes in the memory controller. Lin *et al.* propose to accelerate checksum calculation using GPU [30]. `XLH` [35] enhances the page scan in `ksm`, utilizing hints from guest I/O in a VM environment. It allows `ksm` to identify mergeable pages earlier and merge more pages. Nonetheless, it does not reduce either the consumption

of the server CPU's cycles or the pollution of its caches. `ezswap` [25] estimates the compression ratios of pages in advance, compresses only highly-compressible pages, and store them in `zpool`. Classic `zswap` blindly chooses pages to swap out and compress based on the LRU policy. Song *et al.* propose an efficient way to enhance `zswap` by skipping the compression of incompressible pages [46]. These optimizations are orthogonal to our work and can be employed together with STYX.

Abeyrathne *et al.* [2] demonstrated the potential of offloading kernel functions to FPGA by utilizing the advanced features of the latest Xilinx FPGA with the provided kernel modules. STYX instead deals with the problems by elaborative and creative designs without any limitation on the FPGA model. Roulin *et al.* [42] examined the migration of the user-space network switch daemon to the kernel space. This setup aims to grant complete control of the routing ASIC to the Linux kernel, thereby reducing the overhead of kernel-space and user-space communication. However, this approach does not offer a general solution to the communication between the OS kernel and the offloading device, as it is specifically designed for network switch APIs.

There also have been many studies conducted on SNIC to explore its capacity in various ways. Offloading various functions, such as distributed services and intrusion detection, to SNICs is a promising approach to mitigate resource consumption on servers, enhance the performance of specialized operations, and improve overall energy efficiency [9, 13, 15, 17, 29, 48, 50]. Specifically, `LineFS` [26] offloads distributed file system. `FlexTOE` [44] offloads TCP to SmartNIC with flexibility and high performance. `Xenic` [43] uses the LiquidIO 3 SNIC [33] for fast distributed transactions. `Pigasus` [55] uses an FPGA-based SNIC to accelerate intrusion detection and prevention systems. STYX focuses on harnessing the capabilities of SNICs to effectively mitigate the datacenter memory tax.

9 Conclusion

In this paper, we first showed that memory optimization kernel features intensively consume the server CPU's cycles and pollute its caches when they are invoked. This in turn leads to a significant increase in the p99 latency of memory-intensive/latency-sensitive datacenter applications. Second, we proposed STYX as a solution to minimize the consumption of server CPU's cycles and the pollution of its cache caused by these kernel features. STYX accomplished these by leveraging the RDMA and compute capabilities of modern SNIC, and offloading the intensive operations of these kernel features to SNIC. Lastly, we demonstrated the effectiveness of STYX after re-implementing two memory optimization kernel features in Linux: `ksm` and `zswap` using the STYX framework and running memory-intensive/latency-sensitive applications. We showed that the systems with STYX-based `ksm` and `zswap` achieved 5.6× and 2.9× lower p99 latency values than the systems with classic `ksm` and `zswap`, while preserving the benefits of `ksm` and `zswap`.

Acknowledgments

We thank Jiacheng Ma and Ipoom Jeong for their technical discussion and support. This work was supported in part by Samsung Electronics, the IBM-Illinois Discovery Accelerator Institute and PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Nam Sung Kim has a financial interest in Samsung Electronics and NeuroRealityVision.

References

[1] Control Groups (Cgroups) - Linux kernel documentation. https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html. 2022.

[2] Pabudi T Abeyrathne, S Devapriya Dewasurendra, and Dhammika Elkaduwa. Offloading specific performance-related kernel functions into an FPGA. In *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE'21)*, 2021.

[3] Alibaba Cloud Database Team. Improving redis performance through multi-thread processing. /https://www.alibabacloud.com/blog/improving-redis-performance-through-multi-thread-processing_594150/, 2018.

[4] Amazon Web Services. Aws re:invent 2018: Powering next-gen ec2 instances: Deep dive into the nitro system (cmp303-r1). https://www.youtube.com/watch?v=e8DVmwj3OEs, 2018.

[5] AMD/Xilinx. Alveo SN1000 smartnic accelerator card. https://www.xilinx.com/products/boards-and-kits/alveo/sn1000.html, 2023.

[6] Andrea Arcangeli, Izik Eidus, and Chris Wright. Increasing memory density by using KSM. In *Proceedings of the linux symposium*, 2009.

[7] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE symposium on security and privacy (SP'16)*, 2016.

[8] Wenqi Cao and Ling 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'18)*, 2018.

[9] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. A cloud-scale acceleration architecture. In *Proceedings of the 49th IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*, 2016.

[10] Yann Collet. xxHash: Extremely fast hash algorithm. https://github.com/Cyan4973/xxHash, 2016.

[11] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing (SoCC'10)*, 2010.

[12] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing (ICAC'11)*, 2011.

[13] Dong Du, Qingyuan Liu, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. Serverless computing on heterogeneous computers. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, 2022.

[14] Izik Eidus and Hugh Dickins. Kernel Samepage Merging. https://docs.kernel.org/next/admin-guide/mm/ksm.html, 2009.

[15] Daniel Firestone, Andrew Putnam, Hari Angepat, Derek Chiou, Adrian Caulfield, Eric Chung, Matt Humphrey, Kalin Ovtcharov, Jitu Padhye, Doug Burger, Dave Maltz, Albert Greenberg, Sambhrama Mundkur, Alireza Dabagh, Mike Andrewartha, Vivek Bhanu, Harish Kumar Chandrappa, Somesh Chaturmohta, Jack Lavier, Norman Lam, Fengfen Liu, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Kushagra Vaid, and David A. Maltz. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, 2018.

[16] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI'21)*, 2021.

[17] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C. Snoeren. Smartnic performance isolation with fairnic: Programmable networking for the cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'20)*, 2020.

[18] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over commodity Ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM'16)*, 2016.

[19] Fan Guo, Yongkun Li, Yinlong Xu, Song Jiang, and John C. S. Lui. SmartMD: A high performance deduplication engine with mixed pages. In *2017 USENIX Annual Technical Conference (USENIX ATC'17)*, 2017.

[20] Jinghan Huang, Jiaqi Lou, Yan Sun, Tianchen Wang, Eun Kyung Lee, and Nam Sung Kim. Analyzing energy efficiency of a server with a smartnic under slo constraints. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'23)*, 2023.

[21] Jennings, Seth. The zswap compressed swap cache. https://lwn.net/Articles/537422/, 2013.

[22] Gangyong Jia, Guangjie Han, Joel JPC Rodrigues, Jaime Lloret, and Wei Li. Coordinate memory deduplication and partition for improving performance in cloud computing. *IEEE Transactions on Cloud Computing*, 7(2):357–368, 2015.

[23] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. Live virtual machine migration with adaptive, memory compression. In *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009.

[24] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a warehouse-scale computer. In *Proceedings of the 42nd IEEE/ACM International Symposium on Computer Architecture (ISCA'15)*, 2015.

[25] Jongseok Kim, Cheolgi Kim, and Euiseong Seo. *ezswap*: Enhanced compressed swap scheme for mobile devices. *IEEE Access*, 7:139678–139691, 2019.

[26] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM (SOSP'21)*, 2021.

[27] 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. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*, 2019.

[28] Yanfang Le, Mojtaba Malekpourshahraki, Brent Stephens, Aditya Akella, and Michael M. Swift. On the impact of cluster configuration on RoCE application design. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking (APNet'19)*, 2019.

[29] Junru Li, Youyou Lu, Qing Wang, Jiazhen Lin, Zhe Yang, and Jiwu Shu. AlNiCo: SmartNIC-accelerated contention-aware request scheduling for transaction processing. In *2022 USENIX Annual Technical Conference (USENIX ATC'22)*, 2022.

[30] Wei-Cheng Lin, Chia-Heng Tu, Chih-Wei Yeh, and Shih-Hao Hung. GPU acceleration for kernel samepage merging. In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'17)*, 2017.

[31] Karmen MacKendrick. Fragmentation and memory. In *Fragmentation and Memory*. Fordham University Press, 2022.

[32] Deborah T Marr, Frank Binns, David L Hill, Glenn Hinton, David A Koufaty, J Alan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1), 2002.

[33] Marvell. Marvell LiquidIO III inline dpu based smartnic. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf, 2022.

[34] Microsoft. Cache and Memory Manager Improvements. https://learn.microsoft.com/en-us/windows-server/administration/performance-tuning/subsystem/cache-memory-management/improvements-in-windows-server, 2022.

[35] Konrad Miller, Fabian Franz, Marc Rittinghaus, Marius Hillenbrand, and Frank Bellosa. XLH: More effective memory deduplication scanners through cross-layer hints. In *2013 USENIX Annual Technical Conference (USENIX ATC'13)*, 2013.

[36] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, et al. Scaling memcache at facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, 2013.

[37] NVIDIA Corporation. NVIDIA BlueField-2 DPU: Data center infrastructure on a chip. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf, 2022.

[38] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango:

Achieving high cpu efficiency for latency-sensitive datacenter workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019.

[39] Laura Promberger, Rainer Schwemmer, and Holger Fröning. Characterization of data compression across cpu platforms and accelerators. *Concurrency and Computation: Practice and Experience*, page e6465, 2022.

[40] Shashank Rachamalla, Debadatta Mishra, and Purushottam Kulkarni. Share-o-meter: An empirical analysis of KSM based memory sharing in virtualized systems. In *20th Annual International Conference on High Performance Computing (HiPC'13)*, 2013.

[41] Redislabs. Redis. https://redis.io, 2021.

[42] Andy Roulin. Advancing the state of network switch asic offloading in the linux kernel. Technical report, 2018.

[43] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. Xenic: Smartnic-accelerated distributed transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM (SOSP'21)*, 2021.

[44] Rajath Shashidhara, Timothy Stamler, Antoine Kaufmann, and Simon Peter. FlexTOE: Flexible TCP offload with Fine-Grained parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*, 2022.

[45] Dimitrios Skarlatos, Nam Sung Kim, and Josep Torrellas. Pageforge: A near-memory content-aware page-merging architecture. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*, 2017.

[46] Taejoon Song, Myeongseon Kim, Gunho Lee, and Youngjin Kim. Prediction-guided performance improvement on compressed memory swap. In *2022 IEEE International Conference on Consumer Electronics (ICCE'22)*, 2022.

[47] J.P. Stevenson, M.A. Horowitz, D.R. Cheriton, P.M. Hanrahan, and Stanford University. Department of Electrical Engineering. *Fine-grain In-memory Deduplication for Large-scale Workloads*. 2013.

[48] Maroun Tork, Lina Maudlej, and Mark Silberstein. Lynx: A smartnic-driven accelerator-centric architecture for network servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[49] Carl A Waldspurger. Memory resource management in VMware ESX server. *ACM SIGOPS Operating Systems Review*, 2002.

[50] Zeke Wang, Hongjing Huang, Jie Zhang, Fei Wu, and Gustavo Alonso. FpgaNIC: An FPGA-based versatile 100gb SmartNIC for GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC'22)*, 2022.

[51] 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 ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, 2022.

[52] Keith Wiles. Pktgen-DPDK. https://pktgen-dpdk.readthedocs.io/en/latest/contents.html, 2021.

[53] Jiachen Xue, Muhammad Usama Chaudhry, Balajee Vamanan, T. N. Vijaykumar, and Mithuna Thottethodi. Dart: Divide and specialize for fast response to congestion in RDMA-based datacenter networks. *IEEE/ACM Transactions on Networking*, 28(1), 2020.

[54] Pavel Yosifovich, David A Solomon, and Alex Ionescu. *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*. 2017.

[55] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C. Hoe, Vyas Sekar, and Justine Sherry. Achieving 100gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, 2020.

[56] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale RDMA deployments. In *Proceedings of the 2015 ACM SIGCOMM Conference (SIGCOMM'15)*, 2015.