CXDVIRT: Low Latency Kernel Module Based CXL-SSD Emulation

Hyunsun Chung[1], Seongho Bong[1], Hong-Yeon Kim[2], Youngjae Kim[1,†]
[1]Sogang University, Republic of Korea, [2]ETRI, Republic of Korea

***Abstract*—CXL-SSDs promise memory-semantic access to NAND-scale capacity through a small in-device DRAM cache, but scarce hardware makes emulation central to CXL-SSD research. Cylon, a state-of-the-art emulator, runs workloads in a VM, intercepting DRAM misses to inject NAND latency. We show this VM machinery adds over 3μs per miss, exceeding high-performance NAND read latency, while repeated VM exits inflate tail miss latencies several-fold. We present CXDVIRT, a kernel-module-based CXL-SSD emulator that removes the VM entirely: the device is exposed through host devdax, hits run natively via the MMU, and a custom page-fault path models NAND latency on misses. CXDVIRT preserves CXL-SSDs' bimodal latency profile while reducing per-miss latency 1.4–1.6× (4.1× at P99.9); with eight threads its miss latency only doubles, against 8–9× for Cylon behind QEMU's global lock. CXDVIRT stays within 5% of remote DRAM without NAND traffic, where Cylon is 3.1–3.6× slower, and enables configurable eviction and prefetch policy studies.**

***Index Terms*—Compute Express Link, CXL-SSD, Emulation**

I. INTRODUCTION

The growth of memory-intensive workloads in AI and HPC alike has driven increased interest in memory expansion related research in academia and industry [1]–[5]. Compute Express Link (CXL) [6] is one of the major proposed memory expansion interconnects, with DRAM-based memory expansion devices being deployed in real-life use cases [7], [8]. With the expansion into CXL, a branch of new CXL memory devices in CXL-SSDs has emerged, proposing memory semantic data access with a small DRAM cache backed with a large pool of NAND in one device [9]–[11]. Ongoing research into CXL-SSD designs aims to realize the potential to deliver near CXL memory speeds at NAND SSD level capacities [12]–[14].

One of the largest roadblocks for CXL-SSD research is the limited accessibility of real-life hardware to the general community to perform evaluations and build new CXL-SSD designs. To answer these concerns, many emulators have been proposed, with various limitations including memory trace reliance, limited full-stack software incorporation, and lack of design extensibility [15], [16]. Cylon represents the state of the art in CXL-SSD emulation, enabling real-time workload execution with an emulated CXL-SSD device using QEMU, a virtual machine (VM) [17]. However, the usage of VMs incurs non-trivial amounts of latency and overhead when interception of memory I/O is required to incorporate NAND latency for DRAM misses. While this overhead is less noticeable with slower NAND, high performance NAND with lower latencies used in CXL-SSD studies [13], [14] may suffer due to overheads being larger than the designed simulated latency.

To overcome this limitation, we present CXDVIRT, a CXL-SSD emulation platform that runs directly on the host operating system via a kernel module. By removing the VM from the equation entirely, CXDVIRT reproduces the configured timing parameters more faithfully.

II. PROBLEM DEFINITION

In this section, we go over the weaknesses of a CXL-SSD emulation environment like Cylon that is reliant on a VM. Namely, we reveal the excessive overhead that VMs bring in median and tail miss latency that harms emulation accuracy.

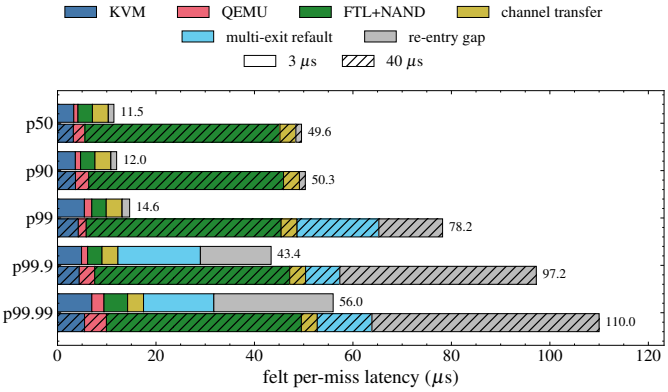


Fig. 1: Breakdown of Cylon's DRAM miss latency across two NAND latency profiles.

To determine the emulation characteristics of the CXL-SSD in Cylon, we run microbenchmarks in the form of a single threaded pointer chase with MIO [18] whose working set is 1.5× the DRAM cache (1.5× WSS), using the evaluation platform as specified in Table II. From these benchmarks, we categorize the contributors to the measured miss latency, and break down the results in tail percentiles in Fig. 1. The results reveal two critical issues of Cylon in emulating a CXL-SSD.

KVM/QEMU overhead in emulation. A key contributor of latency from using Cylon is the VM mechanisms that underlie the emulation platform. Figure 1 breaks down the sources of latency into two categories: KVM and QEMU.

- **KVM**: Time from host machine's kernel operations that occur from both VM exit and entry due to a DRAM miss. Operations included here are the extended page table (EPT) walk, `ioctl` syscall operations, MMIO operations, and x86 instruction execution from the EPT trap.

†Y. Kim is the corresponding author.

- **QEMU**: Actual QEMU operations that pertain to operating the VM. These include the big QEMU lock (BQL) acquisition, memory topology routing, and copy operations from CXL-SSD emulation.

Combined, these two aspects of latency consistently add up to over $3\mu s$ in all latency setups, appearing in both median and tail latency values at a minimum of $4.15\mu s$. This overhead alone exceeds the read latency of high performance NAND such as Samsung Z-NAND [19]. Therefore, a persistent accuracy issue across Cylon's emulation can be identified, clearly emphasizing the shortcomings of VM-reliant emulation for CXL-SSDs with low NAND latencies.

Additional latency from VM exit overhead. Outside of expected (FTL, NAND, channel) and direct VM machinery latency, another major overhead concern is the VM exit overheads. Such overheads show mostly in tail latency, from P99 to the extreme tails in P99.99. We identify the following additional contributors to latency from the VM exit.

- **Multi-exit refault**: The waiting done to incur NAND latency and the EPT update are independent of each other. Even if the wait is complete, if the EPT is not marked for direct access, Cylon re-executes the load operation and performs additional VM exits.
- **Re-entry gap**: Unavoidable gaps of time between each VM exit and re-execution from host contention are also a contributing factor in latency. This is made more evident in the tail latency regions where multi-exits are more common. An access that misses the direct mapping on its first exit waits out the remainder partly in repeated exits and partly in such gaps.

Such high overheads in extreme tail latencies from VM usage are documented in prior studies, and pose concerns for using VM-based emulation in CXL-SSD studies [20]. Current CXL-SSD designs with a small DRAM cache in comparison to the large NAND capacity make the accurate portrayal of DRAM misses critical [10], [11]. A small DRAM cache makes misses more likely to occur, thus elevating the impact of extreme tail latencies. Cylon's failure to portray the configured latencies reveals a gap in current emulation fidelity, thus motivating a new approach for CXL-SSD emulation.

III. Design of CXDVirt

A. Component Overview

Fig. 2 outlines the components that drive CXDVirt's functionality. They are divided into components for initializing the actual CXL-SSD device via the kernel module, and components for the operation and features of the CXL-SSD.

Components for CXL device registration. In order to identify the device created by CXDVirt's kernel module as a CXL memory device, the relevant device metadata for the PCIe device creation must be created. This metadata is stored in a small reserved space of DRAM used for device emulation on the host, and includes the CXL endpoint information, PCIe BAR regions, the MSI-X table, and CXL component and device registers.

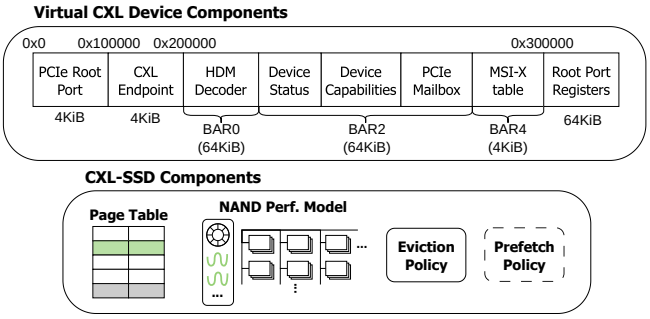


Fig. 2: Components of CXDVirt to ensure functionality.

Components for CXL-SSD operation management. To drive the NAND read, write, and erase latencies of I/O between the DRAM and NAND region, CXDVirt employs a NAND performance model. Additional components include the eviction and prefetch policies that a CXL-SSD device design may want to implement. CXDVirt offers a variety of eviction policies that determine what pages to evict from DRAM to NAND, such as FIFO, LIFO, and CLOCK. For prefetch policies, CXDVirt offers a general next N page background prefetch policy for exploiting spatial locality.

CXDVirt is flexible and modifiable with customizable eviction and prefetch policies to suit a specific workload's needs. To support these features, CXDVirt keeps per-page metadata for the CXL-SSD as seen in Table I. Each page's entry holds its state, dirty bit, CLOCK bit, and pin count, its position in the FIFO/LIFO order, and a wait queue for accesses waiting on its state transitions. Bitmaps are used to track each page's residency, validity, allocation, and prefetch status. The data stored here is not part of the DRAM region exposed as usable memory space, and is considered as a separate reserved memory region. Future CXL-SSD designs with CXDVirt can share this memory region with the host system to facilitate communication with the device for hardware-software co-design studies.

TABLE I: CXDVirt per-page metadata for the 96 GiB device (23.8 M tracked 4 KiB pages).

Structure	Per page	Total
Page entry (state, CLOCK bit, pin count, dirty bit, FIFO/LIFO order, wait queue)	48 B	1,091.4 MiB
Bitmaps (resident, valid, alloc, prefetch)	5 bits	14.2 MiB
Total	48.6 B	1,105.6 MiB

B. Operation Flow of CXDVirt

CXDVirt's key method of enabling the CXL-SSD emulation is the kernel's page table and page-fault handler machinery. CXDVirt uses the entire reserved physical memory region as capacity for the purposes of emulation.

Fig. 3 illustrates this with the overall architecture and execution flow. CXDVirt starts with the user application accessing the emulated CXL-SSD device via the devdax driver (❶).

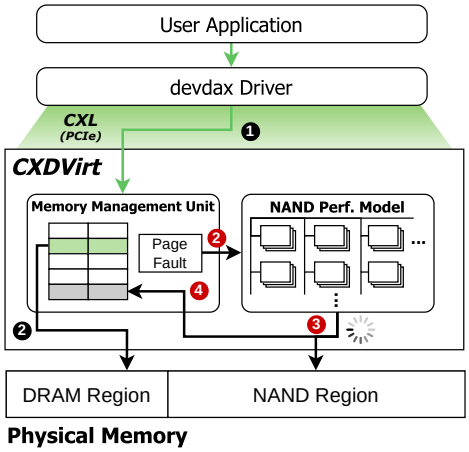


Fig. 3: Overview of CXDVIRT's execution flow.

Here, the execution flow diverges on whether the data is in DRAM (DRAM hit) or in NAND (DRAM miss), determined by using the page table. If a DRAM hit occurs, the memory management unit (MMU) directly translates and accesses the location of the data in DRAM (❷). This enables data access at pure memory access speed without any additional interception.

On the other hand, a DRAM miss triggers a custom page fault implemented using the devdax device (❷). Instead of looking for data in storage like a normal page fault, the page fault here is used as an interception point for memory accesses. By intercepting at this point, CXDVIRT uses a NAND performance model to determine the incurred NAND I/O time in the CXL-SSD. After waiting for the calculated NAND latency, access to the data in physical memory finally occurs (❸). At the same time, the page table is updated to mark said page is now present in memory, allowing for native memory access times (❹) for future access to the same page.

Flush/retrieve between regions. To avoid the overhead of data movement and `memcpy`, CXDVIRT sets no static DRAM or NAND regions, nor does it move data between the two regions in both flush and retrieve operations. Rather, the distinction of the data's location is determined by its page table entry (PTE). Flush from DRAM to NAND removes the PTE to incur a page fault on its next access. NAND writes are only issued if the page is dirty, and can be issued either by the foreground faulting thread or CXDVIRT's background I/O workers depending on whether the flush is to be performed synchronously or asynchronously. Data retrieval from NAND to DRAM installs the PTE to allow direct data access via the MMU. NAND reads are issued in the foreground, or in the background if the retrieval was from prefetch.

C. Page state management in CXDVIRT

A state machine is used to represent the status of pages currently stored in CXDVIRT as seen in Fig. 4. Each page has information on its state saved and accessed in its page metadata entries (§III-A, Table I). Each page is in exactly one of four states as seen below.

- **NOT_CACHED**: Page is located in the NAND region, and requires a NAND read to the DRAM region for usage.

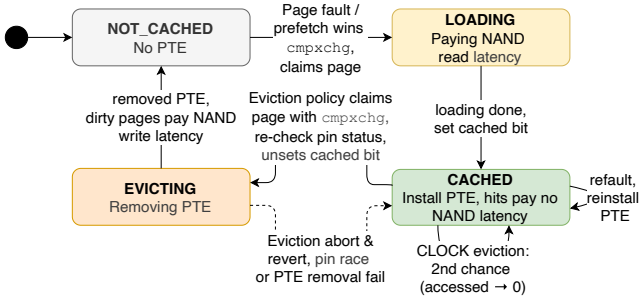


Fig. 4: State machine diagram in CXDVIRT.

- **CACHED**: Page is located in the DRAM region.
- **EVICTING**: Page is in the process of being written from DRAM to NAND if said page is dirty.
- **LOADING**: Page is in the process of being read from NAND to DRAM.

Each state change is claimed with a per-page atomic compare-and-swap, with no device-wide lock held while a miss waits out its NAND latency. Therefore, misses to different pages can proceed in parallel, whereas Cylon serves every miss under QEMU's global lock. Concurrent accesses to a page in LOADING or EVICTING sleep on its wait queue, so each fill is charged once, and a pin count keeps eviction from removing a page whose PTE is being installed.

IV. EVALUATION

A. Implementation & evaluation setup

TABLE II: Specifications of evaluation platform.

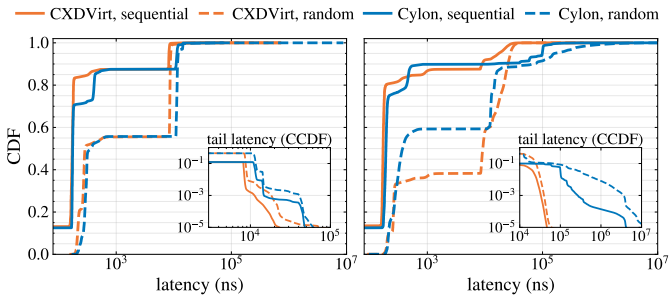
	Cylon	CXDVIRT
CPU	Dual Xeon Gold 5218R, 40 C/80 T	
NAND	Z-NAND [19] SLC, 8 ch × 16 way	
NAND timing	$t_R/t_{PROG}/t_{BERS}$ = 3/100/1000 μs	
OS	Ubuntu 22.04.5	Ubuntu 20.04.6
Kernel	Linux 6.4.6	Linux 6.18.5
Workloads	VM, 8 vCPUs	host
Memory	96 GiB	282 GiB

We implement CXDVIRT[1] through a modified version of NVMeVirt [20], with the evaluation system as specified in Table II. PCI device creation and NAND performance modeling are taken from NVMeVirt, with modifications made for CXL device realization and CXL-SSD device semantics. A custom kernel is implemented to expose the required functions to the CXDVIRT module for the CXL device creation and page table manipulation. A 96GiB region on a remote NUMA node serves as the CXL-SSD's physical storage medium, of which 4.8GiB acts as the DRAM cache unless otherwise stated; said remote placement enables incurring non-local memory access latency to the backing medium. All workload sizes are given as working set size (WSS) ratio to the DRAM cache capacity.

B. Microbenchmarks: Device latency characteristics

Overall performance characteristics: Fig. 5 shows the distribution of Cylon and CXDVIRT latencies running MIO

[1]https://github.com/hschung1652/cxdvirt



(a) 1 thread, 1.7× WSS (b) 8 threads, 2.5× WSS

Fig. 5: Access latency CDFs for sequential and random pointer chasing across Cylon and CXDVIRT.

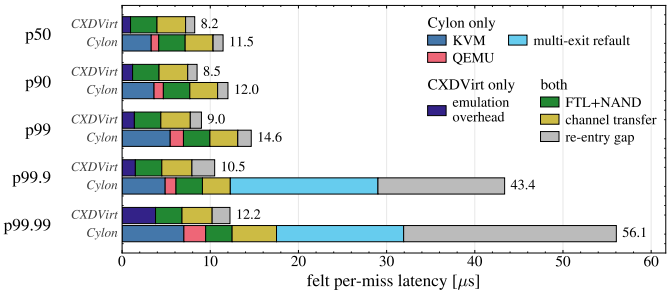


Fig. 6: DRAM miss latencies of CXDVIRT and Cylon across a variety of percentiles.

in a variety of settings. Cylon and CXDVIRT both show the bimodal hit/miss profile expected of a CXL-SSD, but CXDVIRT's misses sit closer to the configured device latency (t_R plus the 4 KiB channel transfer) of 6.2μs. At one thread, CXDVIRT's curves show only the two steps with the miss step 1μs wide at 8–9μs, while Cylon's sequential curves add a third step for the access after each miss and place their misses at 10–12μs. Under eight threads, contention widens CXDVIRT's misses to 8–34μs, whereas Cylon's spread from 12μs to as much as 1.7 ms. On average, a miss at eight threads takes 1.9–2.0× as long as at one thread on CXDVIRT, but 8.0–9.2× on Cylon. The complementary CDF (CCDF) inset graphs take a closer look at tail latencies, which reflect DRAM miss distributions. Here, CXDVIRT is lower at every level from 10^{-1} down to 10^{-4} at one thread, and above P90 at eight threads. At eight threads, Cylon's accesses take 4.6–25× longer at P99 and 8–91× longer at P99.9.

Another point of interest is the lead in Cylon's random curve in Fig. 5(b) until P88. This is not due to lower emulation latency, as Cylon's misses take 6× longer on average, but to its miss path through QEMU. With uniformly random access, at most $\frac{1}{\text{WSS}}$ of accesses can hit, which is 40% in (b). CXDVIRT's plateau sits just below this at 0.38, while Cylon's rises well above it to 0.59. Exceeding the bound requires uneven thread progress, which is caused by QEMU's global lock. Cylon serves each miss, NAND wait included, while holding this lock, and 71% of the time the lock then passes to the same vCPU's next miss. The lock holder's misses are thus served back to back in about 13μs, forming the steep rise from 0.59 to 0.88, while the other vCPUs' misses wait and form the slope above P88. Running ahead also keeps the holder's region cached, which raises Cylon's hit plateau: its threads hit 52–74% and spend 62–177 s in the run, while every CXDVIRT thread hits 38% and spends 33 s.

DRAM miss breakdown: Fig. 6 shows a closer study of DRAM miss latencies from a single-threaded sequential pointer chase workload at 1.5× WSS. The results reveal why CXDVIRT is able to achieve lower DRAM miss latencies over Cylon as seen in Fig. 5. Up to P99 of DRAM misses, CXDVIRT maintains a 1.4× to 1.6× reduction in latency. Entering the extreme tail widens this gap further, with 4.1× and 4.6× for P99.9 and P99.99 respectively. This reduction is due to CXDVIRT's ability to eliminate the VM-specific overheads identified in §II. Against the configured miss latency of 6.2μs, the application observes a 2.0–2.7μs overhead on CXDVIRT up to P99 and 4.1μs at P99.9, against 5.3–8.5μs and 37.2μs on Cylon. CXDVIRT therefore only leaves the cost of the page fault itself between the configured and the observed latency, making the device model the the main determinant of what an application observes.

C. Macrobenchmarks: real-life workloads with CXDVIRT

For real-life workloads, we run the write-intensive Splash-4 OCEAN workload with eight threads, and the read-intensive YCSB-C workload with Redis in a single thread across three WSS ratios [21]–[23]. For the OCEAN workload, the DRAM region's size is adjusted to meet the specified WSS ratio to the workload's size of 13.9GiB. A remote NUMA DRAM baseline was also performed to compare with the 0.35 WSS result, which represents a CXL-SSD workload that fits entirely within the DRAM region with no NAND operations involved.

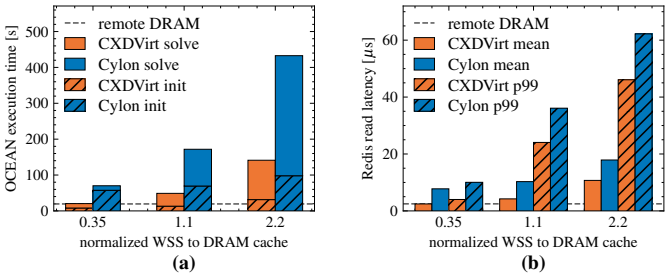


Fig. 7: Results of CXDVIRT and Cylon for (a) Splash-4 OCEAN and (b) Redis YCSB-C.

Fig. 7(a) shows the Splash-4 results in total execution time, where CXDVIRT maintains a 3.06× to 3.51× execution time reduction over Cylon. Cylon's overhead is clearly visible at 0.35 WSS, where no NAND operations occur due to the small WSS. While CXDVIRT stays within 5% of the remote DRAM backing the emulation, Cylon faces a 3.6× increase, indicating emulation machinery overhead. This overhead continues up to 2.2 WSS, where CXDVIRT sees 1.1–2.6μs per miss overhead excluding remote-DRAM time and modeled NAND latency against 15.8–17.3μs on Cylon.

Fig. 7(b)'s Redis results show a similar pattern to Splash-4. At 0.35 WSS, CXDVIRT meets remote DRAM latency, while Cylon is 3.1× slower, its mean at least 5μs above CXDVIRT's

at all three ratios. This extra overhead is especially noticeable in P99 latencies from DRAM miss latency, with a 1.5× and 1.35× increase at 1.1 and 2.2 WSS ratios respectively. The overhead also makes Cylon 1.9× less sensitive to WSS growth: from 0.35 to 2.2 WSS, CXDVIRT reports mean latency rising 4.38× while Cylon only 2.31×.

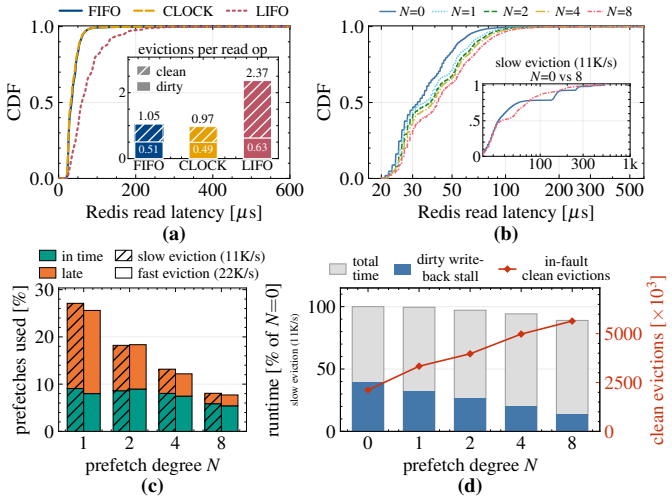


Fig. 8: Eviction (a) & prefetch (b)–(d) policy study results.

Eviction & prefetch policy study: As a case study, we conduct eviction and prefetch policy studies and show the difference in performance characteristics in Fig. 8. Both studies perform the Redis YCSB-C benchmark with a single thread and maintain a WSS ratio of 2.65 to stress the DRAM cache.

Fig. 8(a) covers the eviction policy of the study. The CDF in Fig. 8(a) shows that while FIFO and CLOCK show near identical latency characteristics, LIFO shows significantly worse results, with 1.8–1.9× higher read latency medians over FIFO and CLOCK. The bar graph shows dirty evictions stay at 0.49–0.63 per read for all three policies, indicating that workload, more than the eviction policy, determines the dirty rate of pages. LIFO's 3.2–3.6× higher clean eviction rate also shows that its cost in read latency is in repeated refetches over extra writeback traffic.

Fig. 8(b)–(d) covers the prefetch study, showing next-N prefetch as ineffective for the Redis workload: read latency rises monotonically with N at every percentile in Fig. 8(b). The extra retrievals amplify evictions by 6.1× with the Redis thread paying both the prefetch and the eviction machinery. Prefetch helps only once background eviction is slowed, and only in the upper tail with a 42% reduction at N=8's P80. This gain is not better hit rate, as Fig. 8(c) shows fewer than 10% of prefetches arrive in time, and the two eviction rates agree to within 1.5 points on both prefetch accuracy and timeliness.

Fig. 8(d) resolves the question of how slower eviction takes advantage of prefetch. With background eviction lagging behind, evictions in demand faults are more common, where it is susceptible to a dirty page writeback. Meanwhile, prefetched pages are often clean, so in-fault evictions can pick clean prefetched pages as victims instead of dirty ones, reducing the amount of writeback stall. Hence, the latency reduction comes from preventing writebacks, not from prefetched data being used. The case study shows the level of detail CXDVIRT offers for CXL-SSD machinery studies, and resolves subtle policy questions for a device with the configured timing parameters.

V. Discussion & Future Work

Following Cylon and prior CXL emulation studies [17], [24], [25], CXDVIRT represents CXL memory timings with remote NUMA access latencies—a platform limitation and a source of potential evaluation inaccuracy. Where available, real CXL memory can instead serve as the backing medium, capturing true physical CXL characteristics while incurring emulated NAND I/O latencies on DRAM misses. CXDVIRT's lower emulation overhead makes it a compelling basis for such a solution, which we leave to future work.

VI. Conclusion

We present CXDVIRT, a new CXL-SSD evaluation platform that circumvents the overhead of VMs. Through CXDVIRT, we hope to make CXL-SSD research more accessible by enabling the wider community to contribute new optimized designs of both general and specialized use cases for the emerging memory device.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-00564249 and RS-2024-00453929) and by the Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. RS-2025-02215256).

References

[1] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and Memory Wall," *IEEE Micro*, vol. 44, no. 03, 2024.

[2] Y. Kwon and M. Rhu, "Beyond the memory wall: a case for memory-centric hpc system for deep learning," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[3] X. Yang, Q. Hu, J. Li, F. Li, Y. Zhu, Y. Zhou, Q. Lin, J. Dai, Y. Kong, J. Zhang, *et al.*, "Beluga: A cxl-based memory architecture for scalable and efficient llm kvcache management," *Proceedings of the ACM on Management of Data*, vol. 4, no. 1, pp. 1–29, 2026.

[4] J. Wahlgren, G. Schieffer, M. Gokhale, and I. Peng, "A quantitative approach for adopting disaggregated memory in hpc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[5] X. Wang, J. Liu, J. Wu, S. Yang, J. Ren, B. Shankar, and D. Li, "Performance characterization of cxl memory and its use cases," in *Proceedings of the 2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1048–1061, 2025.

[6] D. Das Sharma, R. Blankenship, and D. Berger, "An introduction to the compute express link (cxl) interconnect," *ACM Comput. Surv.*, vol. 56, July 2024.

[7] Samsung Electronics, "CMM-D: CXL Memory Module." https://semiconductor.samsung.com/cxl-memory/cmm-d/, 2024. Accessed: 2026-08-19.

[8] SK hynix Inc., "SK hynix Completes Customer Validation of CXL 2.0-based DDR5." https://news.skhynix.com/sk-hynix-completes-customer-validation-of-cxl-based-ddr5/, 2025. Accessed: 2026-08-19.

[9] M. Jung, "Hello bytes, bye blocks: PCIe storage meets compute express link for memory expansion (CXL-SSD)," in *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2022.

[10] G. Sun, M. Nik, S. Murugesapandian, D. Li, K. Huen, X. Yao, C. Kahn, A. Beygi, V. Cargnini, and R. Kachare, "Cmm-h pmem: Performance characterization of cxl large capacity persistent memory," *IEEE Computer Architecture Letters*, vol. 25, no. 1, pp. 93–96, 2026.

[11] M. Soltaniyeh, G. Sun, X. Yao, A. Beygi, R. Kachare, D. Zhao, H. Huen, A. Chang, S. Murugesapandian, and C. Kahn, "Revisiting memory hierarchies with cmm-h: Use device-side caching to integrate dram and ssd for a hybrid cxl memory," in *Proceedings of the 17th ACM Workshop on Hot Topics in Storage and File Systems*, HotStorage '25, (New York, NY, USA), p. 45–51, Association for Computing Machinery, 2025.

[12] H. Zhang, Y. Xue, Y. E. Zhou, S. Li, and J. Huang, "SkyByte: Architecting an Efficient Memory-Semantic CXL-based SSD with OS and Hardware Co-design," in *Proceedings of the 2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025.

[13] M. Kwon, D. Gouk, J. Jang, J. Baek, H. You, S. Ji, H. Jung, J. Moon, S. Kang, S. Lee, and M. Jung, "From block to byte: Transforming pcie solid-state devices with compute express link memory protocol and instruction annotation," *IEEE Micro*, vol. 45, no. 6, pp. 46–55, 2025.

[14] D. Oh, M. Kwon, J. Kim, E. Na, J. Moon, H. Choi, S. Jang, H. Choi, H. Jung, S. Lee, and M. Jung, "Compute express link topology-aware and expander-driven prefetching: Unlocking solid-state drive performance," *IEEE Micro*, vol. 45, no. 6, pp. 36–45, 2025.

[15] H. Chung, J. Park, T. Noh, S. Ahn, K. Kim, M. Zhao, and Y. Kim, "Opencxd: An open real-device-guided hybrid evaluation framework for cxl-ssds," in *Proceedings of the 2025 33rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 1–8, 2025.

[16] Y. Wang, Z. Wang, F. Meng, Y. Wang, Y. Ou, L. Wu, W. Hong, X. Ge, and J. Cao, "A Full-System Simulation Framework for CXL-Based SSD Memory System," *arXiv preprint arXiv:2501.02524*, 2025.

[17] D. Yoon, H. Idden, J. Liu, B. Inceisci, S. H. Noh, and H. Li, "Cylon: Fast and accurate Full-System emulation of CXL-SSDs," in *24th USENIX Conference on File and Storage Technologies (FAST 26)*, (Santa Clara, CA), pp. 313–327, USENIX Association, Feb. 2026.

[18] J. Liu, H. Hadian, Y. Wang, D. S. Berger, M. Nguyen, X. Jian, S. H. Noh, and H. Li, "Systematic cxl memory characterization and performance analysis at scale," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, (New York, NY, USA), p. 1203–1217, Association for Computing Machinery, 2025.

[19] TechPowerUp, "Samsung SZ1735 800 GB." https://www.techpowerup.com/ssd-specs/samsung-sz1735-800-gb.d2275, 2021. Accessed: 2026-08-05.

[20] S.-H. Kim, J. Shim, E. Lee, S. Jeong, I. Kang, and J.-S. Kim, "NVMeVirt: A versatile software-defined virtual NVMe device," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, (Santa Clara, CA), pp. 379–394, USENIX Association, Feb. 2023.

[21] E. J. Gómez-Hernández, J. M. Cebrian, S. Kaxiras, and A. Ros, "Splash-4: A modern benchmark suite with lock-free constructs," in *Proceedings of the 2022 IEEE International Symposium on Workload Characterization (IISWC)*, (Austin, TX (USA)), pp. 51–64, IEEE Computer Society, Nov. 2022.

[22] "YCSB." https://github.com/brianfrankcooper/YCSB. Last accessed: 2026-08-20.

[23] "Redis." https://redis.io. Last accessed: 2026-08-20.

[24] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: Cxl-based memory pooling systems for cloud platforms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, (New York, NY, USA), p. 574–587, Association for Computing Machinery, 2023.

[25] J. Hu, H. Li, and M.-C. Yang, "Aquifer: Hierarchical memory pooling with cxl and rdma for microvm snapshots," *arXiv preprint arXiv:2606.24079*, 2026.

Appendix: Artifact Description

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 A characterization of VM-based CXL-SSD emulation (Cylon): the KVM and QEMU path adds over 3 μs to every DRAM miss (at least 4.15 μs at each percentile shown), and repeated VM exits inflate tail miss latency (Fig. 1).

C_2 CXDVirt, a kernel-module CXL-SSD emulator that serves DRAM-cache hits through the MMU and services misses in a custom page-fault handler under a NAND performance model (Section III).

C_3 Lower and more stable miss latency than Cylon: 1.4–1.6× lower through P99 and 4.1–4.6× lower in the extreme tail (Fig. 6); at eight threads miss latency doubles, against 8–9× for Cylon (Fig. 5).

C_4 Without NAND traffic, CXDVirt stays within 5% of remote DRAM on Splash-4 OCEAN and Redis YCSB-C, where Cylon is 3.1–3.6× slower (Fig. 7).

C_5 Configurable eviction and prefetch policies, demonstrated by a policy study (Fig. 8).

B. Computational Artifacts

A_1 The CXDVirt emulator: directory `emulator/`.

A_2 The experiment workflows and the paper's measurements: per-figure drivers, the Cylon instrumentation patch, benchmark patches and configurations, and plotting code: directory `reproduction/`.

Both are in one public repository: `https://github.com/hschung1652/cxdvirt`. Paths below are relative to each artifact's directory.

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_2, C_5	Section III, Table I
A_2	C_1, C_3, C_4, C_5	Figs. 1, 5, 6, 7, 8

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

A_1 is the system of C_2, and the device under test in every CXDVirt measurement; its eviction and prefetch policies are those studied in C_5.

Expected Results

The device appears as `/dev/dax0.0` with a bimodal latency: a resident page is accessed at host-DRAM speed, and a miss is charged at least the configured NAND read time. The bundled self-test checks both, and that the module's count of charged NAND reads equals the number of read misses: the properties on which C_2 rests.

Expected Reproduction Time (in Minutes)

Setup: kernel build and installation, 60–120 min; module and library build, 1 min. Execution: device bring-up, 1 min; self-test, 1 min. Analysis: 1 min.

Artifact Setup (incl. Inputs)

Hardware: An x86-64 machine with two or more NUMA nodes, on which a custom Linux kernel can be installed. The device is a physical carve-out of one node's memory (96 GiB in the paper), and its I/O workers run on four isolated cores of that node. The paper used two Intel Xeon Gold 5218R CPUs (20 cores each) with 282 GiB of DDR4.

Software: Linux 6.18.5 (`kernel.org`) with the supplied patch and `.config`; NVMeVirt (`github.com/snu-csl/nvmevirt`, commit `a6a4252`) with the supplied patch; ndctl v78 or newer (`github.com/pmem/ndctl`). The paper used Ubuntu 20.04.6.

Datasets / Inputs: None.

Installation and Deployment: Compilers: gcc 9.4.0 and GNU Make 4.2.1; the module is built with the compiler that built the running kernel.

1) Build and boot the patched kernel with the carve-out reserved and the device's I/O cores isolated (`memmap=96G$<addr>`, `isolcpus`, `nohz_full`, `intremap=off`, `nokaslr`); `README.md` gives the exact line.
2) `bin/get_nvmevirt.sh` fetches NVMeVirt and applies the patch.
3) `bin/cxdvirt build` builds the module and `daxmalloc`.
4) `bin/setup.sh`, as root, loads the module and brings up `/dev/dax0.0`.

Artifact Execution

T_1 (kernel) → T_2 (build) → T_3 (load the module) → T_4 (`bin/cxdvirt selftest`). Changing a load-time parameter, such as the cache size or policy, means reloading the module (T_3); the parameters are listed in `docs/PARAMETERS.md`.

Artifact Analysis (incl. Outputs)

The self-test reports PASS or FAIL per condition; `bin/cxdvirt stats` reports the device counters, documented in `docs/`.

B. Computational Artifact A_2

Relation To Contributions

A_2 produces every figure in the evaluation: it drives A_1 for the CXDVirt series and an instrumented Cylon for C_1 and the Cylon series.

Expected Results

On other platforms absolute values move but the relations hold: in Fig. 1 the KVM and QEMU stages add several microseconds to every Cylon miss (C_1); in Fig. 5 CXDVIRT's miss latency grows about 2× from one to eight threads and Cylon's 8–9×, and in Fig. 6 CXDVIRT is lower at every percentile (C_3); in Fig. 7 CXDVIRT is near remote DRAM at 0.35× WSS while Cylon is not (C_4); in Fig. 8 LIFO is slower than FIFO and CLOCK, and prefetch helps only under slow eviction (C_5).

Expected Reproduction Time (in Minutes)

Setup: fetching and building, about 5 min (26 s and 40 s measured on eight cores); Cylon's host kernel, FEMU and guest image, about 180–300 min, mostly two kernel builds. Execution: CXDVIRT experiments, about 600 min, 430 min of them Fig. 8; Cylon experiments, about 150 min including VM boots. Analysis: 15 min.

Artifact Setup (incl. Inputs)

Hardware: As for A_1, with at least eight cores and 96 GiB of free memory on the other node for the Cylon guest. The Cylon experiments boot Cylon's Linux 6.4.6 host kernel instead.

Software:

- Splash-4, commit `59949ac`: `github.com/OdnetninI/Splash-4`
- Redis 8.10, commit `5279a8d`: `github.com/redis/redis`
- YCSB 0.17.0: `github.com/brianfrankcooper/YCSB`
- Cylon, commit `4c5e196`, which provides FEMU and the MIO benchmark: `github.com/MoatLab/Cylon`
- a Java runtime (OpenJDK 11 in the paper) and Python 2.7, for YCSB's launcher
- Python 3.8–3.11 with the packages pinned in `plots/requirements.txt`; optionally LaTeX, for the figures' typesetting

Splash-4, Redis and YCSB are used with the supplied patches or configurations, and Cylon with the supplied instrumentation patch and host kernel `.config`.

Datasets / Inputs: The workloads generate their inputs. The paper's measurements for Figs. 5, 7 and 8 ship in `plots/data/` for comparison; the per-access traces behind Figs. 1 and 6 are not included and are regenerated by those figures' targets.

Installation and Deployment: Compilers as for A_1, and meson (1.11.1 in the paper) for ndctl. The NVMeVirt base of A_1 and the benchmarks are fetched at their pinned commits by `scripts/fetch.sh`, and the Cylon tree with `--cylon`; `scripts/build.sh all` builds them, A_1 through its own `bin/cxdvirt build`. `scripts/README.md` gives the full procedure, including Cylon's host kernel, VM and per-boot device setup.

Artifact Execution

T_1 (fetch and build) → T_2 (experiments) → T_3 (figures). In T_2, `scripts/run_experiment.sh` runs one target per figure and emulator, plus one for the remote-DRAM baseline, and writes to `results/`; T_3 is `scripts/make_figures.sh`.

The Cylon targets do not start or stop the VM. Each Cylon configuration is launched by hand with the settings its target prints; QEMU's threads are then pinned and the guest's CXL device is set up and warmed, and the guest is shut down before the next launch. A target refuses a VM launched with other settings or not set up in the current boot.

Each target sets its figure's parameters, or for Cylon prints them. All runs use the 96 GiB device. Except in Fig. 7(a), CXDVIRT's DRAM cache is 4914 MB, matching Cylon's default buffer of 4915 MB, and each workload's footprint sets its working-set size (WSS) relative to the cache:

- Fig. 1: MIO pointer chase, 7400 MB (1.5×), one thread; Cylon only, NAND read time 3 and 40 μs, eviction write-back off.
- Fig. 5: MIO pointer chase, 8390 MB at one thread (1.7×) and 8×1534 MB at eight (2.5×); CLOCK (software bit), slow eviction.
- Fig. 6: CXDVIRT, `felt_probe` pointer chase, 7400 MB (1.5×), one thread, as Fig. 5, drawn from the pass with no write-back; Cylon, the 3 μs run of Fig. 1.
- Fig. 7(a): Splash-4 OCEAN `-n8194 -p8` (13.9 GiB) against caches of 40578, 12912 and 6456 MB (0.35, 1.1 and 2.2×); CLOCK (software bit), read-allocate, eviction 4096 per 1 ms.
- Fig. 7(b): Redis YCSB-C, 1M, 3M and 6M records (0.35, 1.1 and 2.2×), one client thread; CLOCK (software bit), fast eviction.
- Fig. 7, remote DRAM: each workload with its heap on the remote node and no device.
- Fig. 8(a): Redis YCSB-C, 8M records (2.65×), one client thread; FIFO, LIFO and CLOCK (hardware bit), fast eviction.
- Fig. 8(b)–(d): as Fig. 8(a), with CLOCK and next-N prefetch, $N \in \{0, 1, 2, 4, 8\}$, under slow and fast eviction.

Slow eviction drains 64 victims per 10 ms in the background, fast eviction 1024 per 1 ms. "Software bit" is CLOCK without the hardware Accessed bit, which is how Cylon's CLOCK behaves.

Artifact Analysis (incl. Outputs)

Each figure is drawn from `results/`, and the numbers quoted with Fig. 7 are printed beside the paper's values. With `--reference`, the published Figs. 5, 7 and 8 are drawn from `plots/data/` instead.