Storage cluster for persistency, CXL pools for caching

Karim Manaouil
Informatics
The University of Edinburgh
Edinburgh, Midlothian, United Kingdom
karim.manaouil@ed.ac.uk

Ji Zhang
Huawei Research
Zurich, Switzerland
Dr.jizhang@huawei.com

Yang Zhe
Huawei Research
Beijing, China
yangzhe79@huawei.com

Zhou Xing Wang
Huawei Research
Beijing, China
zhouxingwang@huawei.com

Shai Aviram Bergman
Huawei Zurich Research Center
Zurich, Switzerland
shai.aviram.bergman@huawei.com

Antonio Barbalace
The University of Edinburgh
Edinburgh, Midlothian, United Kingdom
antonio.barbalace@ed.ac.uk

Abstract

The emergence of Compute Express Link (CXL) enables new opportunities for scaling memory across data center nodes, introducing a high-speed, coherent memory pool that multiple systems can access simultaneously. While CXL offers a promising alternative to traditional memory and storage hierarchies, its practical integration with the operating systems's page cache remains an open question. In this paper, we investigate the feasibility of leveraging CXL for storage data caching, specifically by extending the operating system's page cache to span CXL memory pools. We prototype a design using `virtiofs` and a CXL memory expander in a virtualized Linux environment, demonstrating its impact on shared file caching across multiple virtual machines. Our evaluation shows that while CXL enables efficient inter-node data sharing, its bandwidth limitations require careful consideration in cache placement strategies. We discuss the trade-offs between local memory, CXL-backed caches, and storage access patterns, offering insights into the potential of CXL to enhance data-intensive workloads in modern data centers.

CCS Concepts

• **Information systems** → **Storage management**; • **Hardware** → **Emerging architectures**; • **Software and its engineering** → **Memory management**.

This work is licensed under a Creative Commons Attribution International 4.0 License.

HCDS '25, Rotterdam, Netherlands

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1470-2/25/03
https://doi.org/10.1145/3723851.3723859

Keywords

Compute eXpress Link, CXL, Page cache, Data cache, Shared memory, Operating System, virtio, virtiofs

ACM Reference Format:
Karim Manaouil, Ji Zhang, Yang Zhe, Zhou Xing Wang, Shai Aviram Bergman, and Antonio Barbalace. 2025. Storage cluster for persistency, CXL pools for caching. In *4th Workshop on Heterogeneous Composable and Disaggregated Systems (HCDS '25), March 30, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3723851.3723859

1 Introduction

Data Caching. Today's memory-intensive applications, such as large language models and databases, require efficient storage data access to maintain performance. These applications necessitate scaling compute and storage independently across multiple nodes, relying on storage servers and distributed file systems to decouple storage from compute. However, this separation increases data access latency due to network overhead and complicates data consistency management. Additionally, storage devices have still significantly higher access latencies than memory, further limiting performance.

Data centers employ multiple *data caching layers* to mitigate storage access latency. These include dedicated cache tiers in storage cluster management systems like Ceph [14] and application-level caching servers for AI training frameworks [2]. Additionally, each server utilizes an OS-level file and block caches – the page-cache, which stores frequently accessed file segments in RAM to optimize storage access.

The raise of CXL. Compute Express Link (CXL) is a recent advancement in enabling disaggregated memory in data centers. Key use cases include expanding a single server's memory via CXL memory expanders and creating large memory pools that can be shared across multiple servers, with or

without consistency guarantees. Built on PCIe [25], CXL provides byte-addressable access with latencies comparable to remote NUMA memory. While CXL allows applications to maintain large datasets in memory, it will not obviate the need for continuous data exchange with storage.

Currently, CXL switches for shared memory pools remain in the prototype stage, whereas CXL memory expanders for single-server deployments are already available and widely studied. We argue that research should now focus on leveraging shared memory pools. Early efforts, such as MemVerge's Project Gismo [1], have begun exploring this, but are limited to application-level solutions, focus on memory, and don't address storage related issues.

Research Idea. Given the central role of data caches in modern applications and the introduction of CXL, we believe it is the right time to investigate whether CXL shared memory pools can serve as the data cache tier in data center clusters. This approach would eliminate the need for caching servers, reduce data replication, and minimize data transfers, leading to lower power consumption and reduced latencies.

Rather than relying on caching servers, compute servers running a distributed application could collaboratively maintain a storage data cache on CXL shared memory pools. If one compute server accesses a piece of data, that is then stored in the cache, allowing other compute servers to read such data directly from the CXL cache, with memory-level latency, eliminating the need for network communication. This CXL-based cache could either extend the OS's page cache or be implemented as a user-space process, depending on the data center's distributed storage architecture. For performance reasons, this paper focuses on the former approach.

The concept of sharing the page cache is not new; it has been explored in the context of virtualization on a single machine through the `virtiofsd` project [24]. This work extends the concept beyond a single machine, enabling multiple machines to share memory pools over CXL. Additionally, it applies to both virtual machines (VMs) and processes. To the best of our knowledge, this approach has not been proposed nor explored before. Figure 1 illustrates our proposed architecture.

Contributions. This paper explores the feasibility of CXL memory pools as a shared page-cache, integrating a Linux kernel prototype with `virtiofs` and `DAX`. Our evaluation demonstrates that while CXL increases cache capacity and enables inter-node sharing, its bandwidth limitations – compared to local memory, require careful data placement. A hybrid caching approach, which dynamically promotes frequently accessed data to local memory, provides the best balance between performance and profitability. These results underscore the importance of intelligent cache management strategies to optimize the use of CXL in future data center storage architectures.

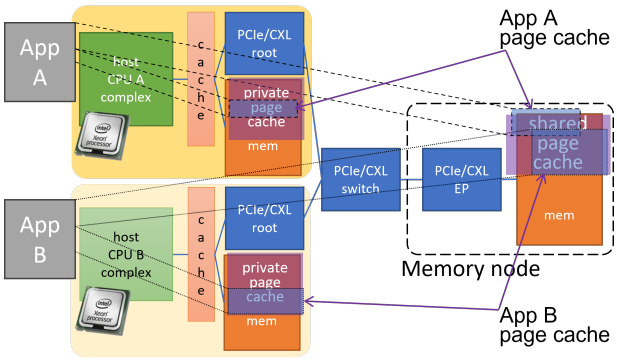

Figure 1: Overall architecture of the proposed shared page cache over CXL/DDR memory for the case of two servers interconnected via a CXL switch with application A and B sharing files.

Limitations. Although several versions of the CXL standard exist, with CXL 1.1 and 2.0 memory extenders, motherboards, and CPUs available commercially, CXL 2.0 switches are still in prototype form, primarily used by large hyperscalers and clients. Additionally, CXL 3.0/3.1 hardware is not yet available. Given the lack of access to CXL shared memory pools and the availability of only CXL memory expanders, our study is based on virtual machines, with the shared page cache residing on the CXL memory expander.

2 Background

Data Center Organization. Data centers, such as public clouds, hyperscalers, and high-performance computing environments, organize their servers into clusters interconnected by high-speed networks. A common practice in modern data centers is the disaggregation of storage from compute [27]. As a result, data centers typically consist of at least one storage cluster and one compute cluster.

CXL. Emerging interconnect technologies, such as OpenCAPI, CCIX, GenZ [8], and CXL [25], are transforming computer architectures by enabling memory expansion over peripheral buses with cache coherence. This shift introduces a new tier of memory – or a new tier of storage that is byte-addressable and nearly as fast as RAM, though with slightly higher latency due to physical remoteness. Similar to traditional data center storage, this new memory tier can be shared among multiple nodes. Among multiple technologies, we focus on CXL, which has gained significant attention, while other efforts have stalled, with major contributors shifting towards CXL. CXL 2.0 introduces shared memory pools, allowing multiple servers to access memory expanders connected via a switch, with software managing concurrent access. CXL 3.0 further enhances this by providing hardware-level cache coherency for shared memory pools.

OS Storage Data Caching. The Linux kernel's page cache constitutes a crucial component of memory management, serving as a caching mechanism that strategically stores frequently accessed disk blocks in main memory to optimize data access. The page cache is a Unified Buffer Cache [26], as it covers both the role of the original BSD page cache and the buffer cache.

3 Motivation

Sharing a single page cache offers several advantages, including data deduplication and improved reusability across different clients. This has been recognized in the virtualization domain, where multiple VMs run on a hypervisor. Each VM maintains its own private page cache, potentially leading to duplicate cached data. Additionally, the hypervisor itself has a page cache that caches data transferred to the VMs. To address this issue, `virtiofsd` [24] allows guest VMs to share a file system in various ways, including mapping the host page cache into the guest VMs, as shown in Figure 2.

This technology aligns with our goal of sharing the kernel-level page cache across different kernel instances, with the page cache residing on CXL memory pools. Our research further expands this approach by investigating its intra-server applicability, and the utilization of both a private *and* shared page-cache to further improve performance.

We further motivate our research direction by estimating the potential performance gains via experimentation. Since we do not have access to an actual CXL switch providing shared memory pools, we use a single machine with multiple NUMA nodes and a CXL memory expander, running VMs. This setup serves as a practical approximation.

We collect data using various configurations of `virtiofsd` across different VMs, either on the same or different NUMA nodes. The shared page cache is allocated on the CXL memory expander if not mentioned differently. Our results provide a lower bound on latency and a realistic representation of bandwidth, which is the primary focus of this paper.

3.1 Profiling and Benefits of Caching

Current offerings from public cloud providers [3, 4, 13] for non-premium network-attached storage promise throughput up to 5GB/s (or 40Gbps), which is between what PCIe 3.0 and PCIe 4.0 NVMe SSDs can achieve [15]. Low-spec VMs, however, often rely on network-attached storage with performance similar to SATA drives.

As a baseline, we measured throughput with SSD and SATA devices, both with and without using the page-cache on a single VM (running Linux/KVM). Experiments were conducted using `fio` performing sequential reads, with a block size of 1MB and queue depth of 32. We report the results obtained from two configurations: without caching, by using `fio` with `O_DIRECT` (bypassing both the guest and host page caches), and with caching, by configuring `virtiofsd` with `caching = always`, enabling both host and guest page caches. We verified the caching state by checking the size of the page cache in both the host and guest at the end of each experiment. The experimental setup is illustrated in Figure 2a – only local memory is used, but we considered a single VM.

Run	First	Last
Direct I/O	85.8MB/s	84.8MB/s
With Caching	67.5MB/s	38GB/s

Table 1: SATA disk throughput results for single VM

Run	First	Last
Direct I/O	2.45GB/s	2.45GB/s
With Caching	2.69GB/s	38.5GB/s

Table 2: NVMe disk throughput results for single VM

Device	SATA disk	NVMe disk
Direct I/O	145MB/s	3.40GB/s
With Caching	70GB/s	70GB/s

Table 3: SATA and NVMe disks cumulative throughput results for multiple VMs.

Table 1 presents the results for a SATA disk: without caching, the maximum throughput reaches 85.8MB/s, while with caching it increases to 38GB/s. Table 2 shows the results for an NVMe disk: without caching, the maximum throughput is 2.45GB/s, and with caching, it reaches 38.5GB/s.

Takeaway. As anticipated, enabling the page caches significantly improves throughput, with an approximate three-fold increase for NVMe, and two-orders of magnitude higher for SATA drives.

3.2 Benefits of a Shared Cache

We investigated the achievable sequential throughput when multiple VMs share the same set of files over `virtiofsd`. In this setup, two VMs access the same files using `fio`, with `virtiofsd` configured without DAX support. This configuration resulted in each VM maintaining its own private page cache in DRAM, as shown in Figure 2a. `fio` is configured to create 12 jobs, each accessing sequentially a different 1GB file with a uniform distribution, the group of files is the same for every VM. The results are in Table 3 where we report cumulative throughput for our experiments with each storage technology.

The experiments without caching are constrained by the inherent speed of the storage device used. Compared to results from a single VM, the total achievable throughput is lower due to resource contention. In contrast, the experiments with caching are limited by the speed of local memory.

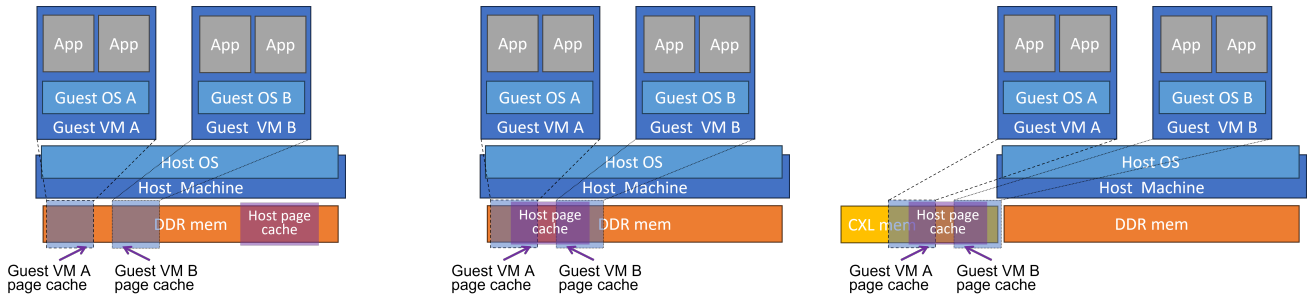

Figure 2: The different configurations benchmarked in the motivation.

Takeaway. As expected, enabling caching when sharing data improves throughput by a factor of 2.5 to 500.

3.3 Shared Cache on CXL

Having established that caching provides benefits regardless of storage technology, we experimented with `virtiofsd` DAX [24]. Using DAX, VMs can directly map the host's memory into its address space via the use of a memory window. This mode enables guest VMs to directly access the host page-cache, thus obviating the need for an additional copy between the host and the VMs.

We first evaluate `virtiofsd` DAX with the DAX window mapped to local DDR memory, and then to CXL memory. These configurations are depicted in Figure 2b and Figure 2c, respectively. We report the experimental results in Table 4, with throughput results cumulative across VMs.

Mapping the host page-cache into guest VMs eliminates the overhead associated with managing multiple page caches, and in these experiments – though not in our proposal – the overheads introduced by virtualization.

Moving the page-cache to CXL shared memory pools frees up more local memory for applications running on servers, and enables file sharing among multiple servers once CXL switches become available. However, our measurements show that bandwidth is constrained by the capabilities of the CXL memory expander. Based on this, CXL memory expanders or shared memory pools cannot be used as-is to support the page cache for all applications, as it may lead to significant performance degradation. *Takeaway.* CXL extends the page cache capacity of VMs, which would otherwise be limited by their memory size, and enables inter-machine sharing. However, the bandwidth is constrained by the specific CXL device, and the latency is higher compared to locally attached memory.

4 Design

While CXL offers lower latency than storage, whether networked or directly attached, our experiments showed that

Device	SATA disk	NVMe disk
DDR	70GB/s	70GB/s
CXL	17.4GB	17.4GB/s

Table 4: SATA and NVMe disks cumulative throughput with `virtiofsd` DAX on DDR or on CXL.

using CXL for the page cache may not match the performance of local memory. However, local memory is limited in capacity, whereas CXL memory offers larger capacity and the potential for sharing across multiple machines [2, 14]. This creates the opportunity for an inter-machine page cache over CXL, providing remote memory access without the overhead of a full network-attached caching server. In this work, we propose a more efficient approach than the naïve method of maintaining the entire shared page cache on CXL.

Efficiency: Low Overheads. Sharing the page cache across multiple machines over CXL reduces the number of data copies that need to traverse the network, thereby minimizing network bandwidth consumption and data access latency compared to adding a dedicated data cache server. To avoid the overhead of inter-process communication, we propose sharing the existing OS page cache across multiple machines over CXL, leveraging the fact that the operating system already manages a data cache.

While implementing this data cache in user-space would avoid changes to the OS kernel, we suggest implementing it in kernel-space for seamless integration with existing applications. Additionally, our experience with `virtiofsd` indicates that using DAX to share the host page cache alleviates the need for an additional data copy between the host page cache and the VM's page cache, leading to higher throughput.

To further enhance reliability and parallelism, data can be cached on multiple (switch-attached) CXL memory expanders simultaneously, allowing for striped/interleaved access to improve parallel read/write performance.

Efficiency: Locality. For applications with frequent data accesses, relying on CXL could reduce performance over time,

as we've shown in Section 3. To address this, we propose a dynamic mechanism for caching data promotion and demotion at runtime. This mechanism will promote frequently accessed file chunks from CXL to local memory and demote less frequently accessed chunks from local memory to CXL.

By doing so, the majority of local memory can be utilized for application processing rather than acting as a storage cache, maximizing the available local memory for tenants' applications. As illustrated in Figure 1, local memory remains private to each server but can be offloaded to CXL, enabling shared access across multiple machines.

Research Questions. The proposed design raises several research questions regarding the re-engineering of existing OS kernels to implement an extensible page cache over CXL. While prior works [18, 28] (see Section 8) have explored extending the Linux cache for tiered memory, they did not address several critical aspects, including:

(a) The ability for multiple OS instances across different servers to access the shared cache concurrently.
(b) The independent failure of servers, CXL switches, or CXL memory expanders.
(c) The need for OS-level communication to manage the shared cache.
(d) Coordination for flushing data to persistent storage.
(f) The dynamic movement of file chunks between CPU-attached memory and CXL memory based on application behavior.
(g) How to achieve such data management automatically and transparently.

Additionally, future work may explore these aspects within the context of a hierarchy or mesh of CXL switches.

5 Initial Prototype/Implementation

This paper takes the first steps towards answering the research questions identified in the previous Section. We begin by exploring the feasibility of a design that implements page cache sharing over CXL; *with a local cache (on DDR memory) for hot data and a shared disaggregated cache (on CXL memory) for shared, cold data.* We aim to assess whether this approach offers superior performance compared to alternative designs, such as using DDR or CXL for the entire page cache. To evaluate the proposed design, we developed a Linux kernel patch (open sourced, see below) which is applied to the virtualization setup described in Section 6.

The patch enables allocation of files from the appropriate memory node based on whether the file is shared or private. Shared files are allocated from the CXL memory node, while private files are allocated from the local memory node where `virtiofsd` is running. Private files simulate promoted, hot file data in this prototype. In subsequent experiments, we vary the number of private files while keeping the access distribution (uniform or zipfian) fixed, to assess how different ratios of CXL and local memory impact caching performance based on the workload's access pattern.

Memory interface	Bandwidth	Latency
DDR	104GB/s	105 – 200ns
CXL	17.5GB/s	278 – 363ns

Table 5: Direct attached DDR and CXL memory bandwidth and latency range comparison.

6 Testbed Setup

Hardware. All experiments were conducted on a dual-socket AMD EPYC 4th Gen 9224 processor, providing a total of 48 cores and 4 NUMA nodes with 768GB of RAM (192GB per NUMA node), using the Gigabyte MZ73 motherboard. A 128GB CXL 1.1 Samsung CXL memory expansion module is directly attached to one of the CPU sockets, while the other CPU socket is connected to a 4TB 5400RPM SATA Seagate Barracuda HDD and a 512GB NVMe Samsung 970P.

We measured the latency and bandwidth of both DRAM and CXL memory expanders using the Intel Memory Latency Checker (`mlc`). The results are summarized in Table 5. The bandwidth represents the maximum achievable bandwidth, and the latency values provided reflect a range for each interface, due to varying access latencies across different NUMA nodes linked to distinct memory controllers in the NUMA topology (including crossing the CPU socket). For example, the end-to-end latency when accessing the memory expander, attached to the second NUMA node on the second CPU socket, from the first NUMA node of the first CPU socket, is around 363ns.

We measured the bandwidth of the various disks on our system using `fio`. The SATA HDD achieves a bandwidth of 120MB/s, while the NVMe device reaches 3.8GB/s. As previously mentioned, due to the lack of access to a CXL switch, our experiments are designed to simulate the CXL setup shown in Figure 1, using the VM configuration outlined in Figure 4. Given the server's multiple NUMA nodes and a CXL memory expander, we ran the VMs on the same CPU socket (but across different NUMA nodes), with the CXL memory expander attached to the other CPU socket. This configuration was intended to approximate the latency (363ns) typically associated with a CXL switch (434ns, as reported in [23]).

Unless specified otherwise, the host page-cache is allocated on the CXL memory expander. CPU hyperthreading was disabled, CPU frequency scaling was set to performance mode, and 1GB huge pages were used to back the VMs. Each VM was configured with 32GB of RAM and 12 CPUs.

Software All experiments are conducted using Linux kernel v6.13-rc6 for both the guest and host systems. The host runs Debian Trixie 13, while the guest VMs use Ubuntu 22.04. To

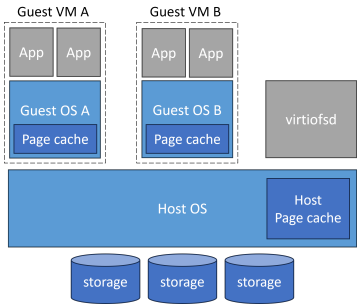

Figure 3: virtiofs in action

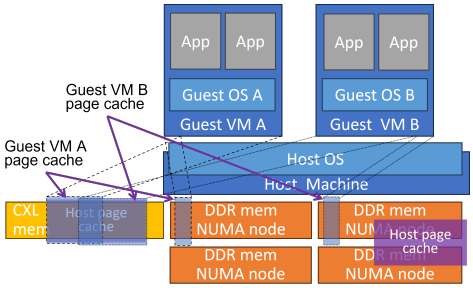

Figure 4: Testbed configuration

minimize variability in results, frequency scaling and auto-numa were disabled. We utilize `virtiofsd` version 7.0.0 and FUSE kernel interface version 7.36. `virtiofsd` is a shared file system that enables VMs to access a host directory tree, providing local file system semantics and performance. It also supports mapping a portion of the host page-cache into the guest via DAX windows.

Unless stated otherwise, our primary benchmarking tool was the Flexible I/O Tester (fio) version 3.28, configured with 12 threads, each accessing 12GB files. All results presented are averages of 10 runs.

7 Evaluation

Here, we present the results of a prototype that simulates the design outlined in Section 4.

7.1 Experimental Protocol

To evaluate our design, which incorporates both a private and shared page cache implemented in the Linux kernel, we use the aforementioned setup and develop a Python-based workload generator leveraging `fio`.

The generator allows us to control two key parameters: the ratio of private files (allocated in local memory) to shared files (allocated in CXL memory), and the file access distribution, which can be either uniform or zipfian with adjustable skewness controlled via the `theta` parameter. The generator then produces a `fio` configuration file based on these parameters, ordering the files from private to shared and from the most accessed to the least accessed. The `fio` flow parameter is utilized to implement the zipfian distribution by assigning relative priorities to the jobs. Subsequently, another script is used to test all combinations of file access distributions and private file ratios (referred to as `pratio`). For each combination, two fresh VMs are created, and the corresponding `fio` job is run three times consecutively to ensure the cache reaches a steady state. The bandwidth results obtained from these experiments are presented in the following section.

7.2 Results

We evaluate three access distributions: uniform (`theta=0`), skewed (`theta=1.2`), and highly skewed (`theta=1.5`), where a skewed distribution indicates that a small subset of files receives the majority of accesses.

Initially, we assess the achievable bandwidth with local memory caching by setting all files as private (`pratio=100`), ensuring they are stored in the guest's page cache. As shown in Figure 5, a highly skewed access pattern (`theta=1.5`) achieved the highest performance, with a bandwidth of 32GB/s, as the working set fit entirely within local memory. However, with a uniform access distribution, the bandwidth drops significantly due to thrashing, as the working set (72GB) exceeded the guest's 32GB memory capacity, forcing Linux to perform page frame reclamation.

Next, we examined the effect of caching all files on CXL (`pratio=0`), revealing a bandwidth cap of approximately 16GB/s across all access distributions, constrained by the CXL memory expander. While CXL mitigates thrashing by offering a larger cache, it does not match the performance of local memory. To explore the balance between local and shared caching, we vary the private/shared cache ratio (Figure 6). Results confirm that a hybrid approach maximizes bandwidth while minimizing thrashing. Selectively promoting frequently accessed data to local memory improves performance, but excessive promotion—especially with uniform access patterns—result in performance degradation. Efficient cache management is essential for optimizing data locality and balancing memory utilization.

These results emphasize the importance of selective promotion: caching a larger portion of the dataset in local memory enhances performance, but only when the working set remains within the system's memory capacity. When this capacity is exceeded, as observed with `pratio=70` under a uniform access distribution, excessive promotion leads to performance degradation. Achieving an optimal balance between local and shared caching is crucial for maximizing efficiency. Our findings demonstrate that a hybrid approach, combining local and shared CXL-backed caches, strikes an effective balance between maximizing bandwidth and preventing cache thrashing. While CXL facilitates inter-node

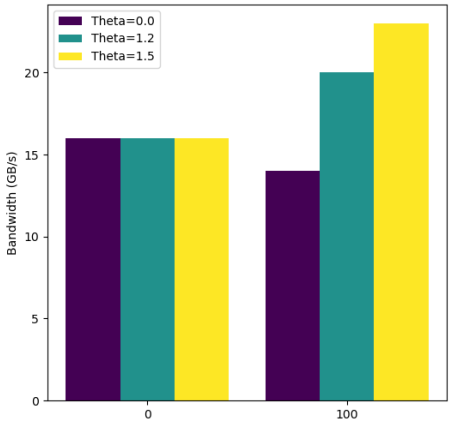

Figure 5: Bandwidth when running with only private files - exclusively caching in guest's local memory (`pratio` 100 on X-axis 100) vs. bandwidth when running with only shared files - exclusively caching in CXL memory (`pratio` 0 on X-axis 0).

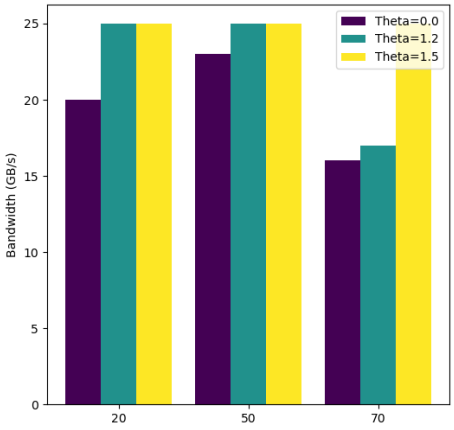

Figure 6: Bandwidth when running with mixed private and shared files, varying the ratio of private files

data sharing and extends cache capacity, its lower bandwidth compared to local memory makes pure CXL-based caching suboptimal for high-throughput applications. These insights highlight the need for dynamic cache management strategies, where frequently accessed data is promoted to local memory, and less critical data remains in the shared CXL cache. These observations suggest that future caching mechanisms should incorporate intelligent policies for data promotion and eviction, adapting to workload access patterns and memory constraints.

8 Related Work

The Linux kernel introduced both frontswap [19] and cleancache [28] to support transcendental memory [20]. Specifically, cleancache was designed to reduce DRAM consumption by reading disk pages from transcendental memory. This concept is similar to our approach, with transcendental memory being replaced by CXL shared memory. However, unlike the transcendental memory, our approach necessitates handling of multiple servers that may request the same page concurrently, and allows byte-addressable accesses by the hosts directly to the CXL data. Like P2CACHE [18], we address the challenge of using tiered memory to extend the page cache. However, our approach focuses on CXL memory expanders rather than persistent memory. Additionally, while P2CACHE is designed for a single server, our approach targets multiple servers that share a common page cache, introducing additional requirements for coordination and functionality.

Prior research demonstrates the sharing of memory and compute resources in distributed systems via low-latency interconnects, including distributed data caches [9]. Our work pioneers shared caching in disaggregated architectures, advancing this vision by unifying caching across nodes through CXL memory pools. Distributed operating systems by definition extend OS services among all machines on which the distributed OS is running, including the page cache. However, none of the recent (research) distributed OSes we are aware of [5–7, 10–12, 17, 22, 29] implement a distributed page cache, but Kerrighed [21] and Hare [16]. Hare is the most similar attempt to this work using the shared memory to save the buffer cache. However, none of these works

9 Conclusion

Leveraging CXL memory pools to coherently share data across multiple computing nodes presents numerous opportunities, such as reducing access times to storage data by maintaining a copy within memory pools. However, for practical implementation, several questions must be addressed: Where should CXL be positioned within the system, and how can it seamlessly integrate with existing OS components, such as file systems, the page cache, and applications? The goal is to enhance performance and reduce latency by utilizing memory sharing to eliminate data copying, minimize network traffic, and accelerate data transfers.

This paper represents an initial exploration of the design space for using CXL to accommodate storage data – the first such effort, to the best of our knowledge. The paper culminates in a functional prototype, which provides indicative performance metrics. This prototype embodies one of the

proposed approaches: a file cache maintained on a CXL memory pool, shared across multiple OS kernels. This reflects our belief that, in the future, data persistency will continue to be handled by storage clusters, while data caching may migrate from individual nodes to CXL memory pools. At a high level, this investigation aims to offer insights into the potential of CXL to transform computing systems. Our patch, scripts, and results are open source and available at https://github.com/systems-nuts/virtiofs-cxl-experiments.

References

[1] [n. d.]. Memverge Gismo. https://memverge.com/wp-content/uploads/MemVerge-Gismo-FMS2023.pdf.

[2] Alluxio. [n. d.]. *Data Caching Strategies for Data Analytics and AI: Data+AI Summit 2023 Session Recap.* https://www.alluxio.io/blog/data-caching-strategies-for-data-analytics-and-ai-dataai-summit-2023-session-recap

[3] Amazon AWS. [n. d.]. *Amazon EC2 Instance types.* https://aws.amazon.com/ec2/instance-types/

[4] Microsoft Azure. [n. d.]. *Azure managed disk types.* https://learn.microsoft.com/en-us/azure/virtual-machines/disks-types

[5] Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. 2017. Breaking the Boundaries in Heterogeneous-ISA Datacenters. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. ACM, New York, NY, USA, 645–659. doi:10.1145/3037697.3037738

[6] Antonio Barbalace, Binoy Ravindran, and David Katz. 2014. Popcorn: a replicated-kernel OS based on Linux. In *Proceedings of the Linux Symposium, Ottawa, Canada.*

[7] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. 2015. Popcorn: Bridging the Programmability Gap in Heterogeneous-ISA Platforms. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, Article 29, 16 pages. doi:10.1145/2741948.2741962

[8] Brad Benton. 2017. Ccix, gen-z, opencapi: Overview & comparison. In *OpenFabrics Workshop.*

[9] Shai Bergman, Onur Mutlu, Wu Yong, Keji Huang, and Ji Zhang. 2024. Composable Storage Servers: A Storage Paradigm for Disaggregated Systems. In *2024 International Conference on Networking, Architecture and Storage (NAS)*. 1–4. doi:10.1109/NAS63802.2024.10781363

[10] Sharath K. Bhat, Ajithchandra Saya, Hemedra K. Rawat, Antonio Barbalace, and Binoy Ravindran. 2015. Harnessing Energy Efficiency of heterogeneous-ISA Platforms. In *Proceedings of the Workshop on Power-Aware Computing and Systems* (Monterey, California) *(HotPower '15)*. ACM, New York, NY, USA, 6–10. doi:10.1145/2818613.2818747

[11] Shenghsun Cho, Han Chen, Sergey Madaminov, Michael Ferdman, and Peter Milder. 2020. Flick: Fast and Lightweight ISA-Crossing Call for Heterogeneous-ISA Environments. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 187–198. doi:10.1109/ISCA45697.2020.00026

[12] Ho-Ren Chuang, Karim Manaouil, Tong Xing, Antonio Barbalace, Pierre Olivier, Balvansh Heerekar, and Binoy Ravindran. 2023. Aggregate VM: Why Reduce or Evict VM's Resources When You Can Borrow Them From Other Nodes?. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 469–487. doi:10.1145/3552326.3587452

[13] Huawei Cloud. [n. d.]. *Elastic Cloud Server.* https://support.huaweicloud.com/intl/en-us/ecs/index.html

[14] Ceph Storage Cluster Documentation. [n. d.]. *Cache Tiering.* https://docs.ceph.com/en/reef/rados/operations/cache-tiering/

[15] William Gayde. [n. d.]. *PCIe 4.0 vs. PCIe 3.0 SSDs Benchmarked.* https://www.techspot.com/review/1893-pcie-4-vs-pcie-3-ssd/

[16] Charles Gruenwald, Filippo Sironi, M. Frans Kaashoek, and Nickolai Zeldovich. 2015. Hare: a file system for non-cache-coherent multicores. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, Article 30, 16 pages. doi:10.1145/2741948.2741959

[17] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. 2015. K2: A Mobile Operating System for Heterogeneous Coherence Domains. *ACM Trans. Comput. Syst.* 33, 2, Article 4 (June 2015), 27 pages. doi:10.1145/2699676

[18] Zhen Lin, Lingfeng Xiang, Jia Rao, and Hui Lu. 2023. P2CACHE: Exploring Tiered Memory for In-Kernel File Systems Caching. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 801–815. https://www.usenix.org/conference/atc23/presentation/lin

[19] LWN.net. 2010. Cleancache and Frontswap. Online. https://lwn.net/Articles/386090/.

[20] LWN.net. 2011. Transcendent Memory in a Nutshell. Online. https://lwn.net/Articles/454795/.

[21] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, Gaël Utard, R. Badrinath, and Louis Rilling. 2003. Kerrighed: A Single System Image Cluster Operating System for High Performance Computing. In *Euro-Par 2003 Parallel Processing*, Harald Kosch, László Böszörményi, and Hermann Hellwagner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1291–1294.

[22] Edmund B Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt. 2009. Helios: heterogeneous multiprocessing with satellite kernels. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 221–234.

[23] Brian Pan. [n. d.]. *CXL Memory Sharing System Architecture and Solution Demo.* https://files.futurememorystorage.com/proceedings/2024/20240807_CXLT-202-1_Pan.pdf

[24] Community Project. [n. d.]. *virtiofs.* https://virtio-fs.gitlab.io/

[25] Debendra Das Sharma. 2022. Compute Express Link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*. 5–12. doi:10.1109/HOTI55740.2022.00017

[26] Chuck Silvers. [n. d.]. *UBC: An Efficient Unified I/O and Memory Caching Subsystem for NetBSD.* https://www.usenix.org/legacy/publications/library/proceedings/usenix2000/freenix/full_papers/silvers/silvers_html/

[27] Susnjara Stephanie and Smalley Ian. [n. d.]. *What is a data center?* https://www.ibm.com/think/topics/data-centers

[28] The Linux Kernel Doc. 2018. Cleancache. Online. https://www.kernel.org/doc/html/v4.20/vm/cleancache.html.

[29] Tong Xing, Antonio Barbalace, Pierre Olivier, Mohamed L. Karaoui, Wei Wang, and Binoy Ravindran. 2022. H-Container: Enabling Heterogeneous-ISA Container Migration in Edge Computing. *ACM Trans. Comput. Syst.* 39, 1–4, Article 5 (July 2022), 36 pages. doi:10.1145/3524452