My CXL Pool Obviates Your PCIe Switch

Yuhong Zhong
Columbia University

Daniel S. Berger
Microsoft Azure
University of Washington

Pantea Zardoshti
Microsoft Azure

Enrique Saurez
Microsoft Azure

Jacob Nelson
Microsoft Research

Antonis Psistakis
University of Illinois
Urbana-Champaign

Joshua Fried
MIT CSAIL

Asaf Cidon
Columbia University

ABSTRACT

Pooling PCIe devices across multiple hosts offers a promising solution to mitigate stranded I/O resources, enhance device utilization, address device failures, and reduce total cost of ownership. The only viable option today are PCIe switches, which decouple PCIe devices from hosts by connecting them through a hardware switch. However, the high cost and limited flexibility of PCIe switches hinder their widespread adoption beyond specialized datacenter use cases.

This paper argues that PCIe device pooling can be effectively implemented in software using CXL memory pools. CXL memory pools improve memory utilization and already have positive return on investment. We find that, once CXL pools are in place, they can serve as a building block for pooling any kind of PCIe device. We demonstrate that PCIe devices can directly use CXL memory as I/O buffers without device modifications, which enables routing PCIe traffic through CXL pool memory. This software-based approach is deployable on today's hardware and is more flexible than hardware PCIe switches. In particular, we explore how disaggregating devices such as NICs can transform datacenter infrastructure.

CCS CONCEPTS

• **Software and its engineering** → **Operating systems**; • **Computer systems organization** → **Cloud computing**; • **Hardware** → *Emerging technologies.*

HOTOS 25, May 14–16, 2025, Banff, AB, Canada

ACM ISBN 979-8-4007-1475-7/25/05.
https://doi.org/10.1145/3713082.3730393

KEYWORDS

Compute Express Link, CXL, PCIe switch, resource pooling, datacenter, cloud computing

ACM Reference Format:
Yuhong Zhong, Daniel S. Berger, Pantea Zardoshti, Enrique Saurez, Jacob Nelson, Antonis Psistakis, Joshua Fried, and Asaf Cidon. 2025. My CXL Pool Obviates Your PCIe Switch. In *Workshop in Hot Topics in Operating Systems (HOTOS 25), May 14–16, 2025, Banff, AB, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3713082.3730393

1 INTRODUCTION

PCIe devices — such as NICs, SSDs, and accelerators — account for a significant portion of server cost and power consumption [2, 30, 78]. Even optimized platforms like AWS and Azure use servers that physically connect a dozen SSDs over PCIe [12, 59]. All servers use at least one high-bandwidth NIC. This tight coupling of PCIe devices to servers requires providers to over-provision PCIe devices for peak demands and to offer redundancy for handling device failures. This leads to low utilization of NICs [67, 71] and local SSDs [66, 71]. Unused I/O resources are stranded within a host and cannot be used for I/O-intensive workloads running on other hosts. A common approach to address resource stranding is to pool resources *across multiple hosts* [26, 57, 63, 70]. Pooling resources within half a rack can already significantly improve utilization [25, 57].

PCIe pooling enables multiple hosts to use any PCIe device in a shared pool. This would bring multiple benefits:

- **Utilization.** Pooling PCIe devices mitigates stranded I/O resources and improve device utilization, which reduces datacenter total cost of ownership (TCO) [57, 71].
- **Failover.** When a PCIe-attached device fails, the host using it can fail over to other devices in the pool automatically. Pooling also reduces the number of redundant devices required for fault tolerance in a rack.

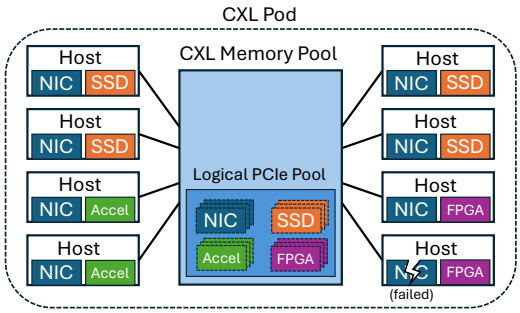


Figure 1: A CXL memory pool enables hosts to access any PCIe device in a CXL pod, forming a logical pool of PCIe devices.

- **Load Balancing.** To prevent high load and high latency from PCIe device saturation, pools can dynamically adjust the number of hosts using a PCIe device by migrating workloads to less-utilized devices.
- **Peak Performance.** During demand spikes, a host can harvest all the PCIe devices in the pool to achieve higher aggregated performance [84].
- **Enabling Hardware Innovation.** When new accelerators (e.g., smart SSDs and FPGAs) are deployed in public clouds, they often face low utilization. Pooling addresses this by allowing cloud providers to deploy a small number of accelerators (e.g., 1:16 ratio) while ensuring all hosts in the target racks can access them.

However, practical solutions for pooling PCIe devices have been limited and PCIe switches are the only viable option today. One might think to use RDMA, since cloud providers already utilize RDMA to disaggregate SSDs [17, 27, 58]. However, in practice, RDMA latency is too high; all cloud providers still offer host-local SSDs in addition to remote SSDs. Anecdotally, AWS has tried to remove local SSDs from their offers, but brought them back due to customer demand[1]. Pooling the NICs themselves over RDMA is unfeasible.

PCIe switches have been *the only* generic solution to pool PCIe devices [1, 43]. Hosts and PCIe devices connect to a common PCIe switch which allows any host to utilize any device in the pool [8, 11, 14, 15].

Despite being technically available, deploying PCIe switches is costly and inflexible [43]. The total cost of using PCIe switches in a rack, including the expenses for PCIe switches, switch software, host adapter cards, and cabling, easily reaches $80,000 [6]. Realistic deployments require redundant switches for fault tolerance and firmware updates, further increasing costs. Such high costs can easily outweigh the cost savings of pooling. Additionally, as a hardware solution, PCIe switches are inflexible. Different vendors have varying requirements for the topology between hosts and PCIe devices, as well as the types of PCIe devices they can support [8, 11, 14]. For example, Liquid's SmartStack is specifically designed to pool GPUs and does not support other device types [14], and GigaIO's FabreX has separate pooling appliances for storage devices and accelerators [9, 10].

In this paper, we argue that *PCIe device pooling can be implemented in software* on top of Compute eXpress Link (CXL) memory pools. CXL memory pools have gained industry interest to improve memory utilization and scale up in-memory databases [13, 20, 25, 33, 36, 40, 54, 55, 57, 64]. Hardware vendors have also started offering CXL memory with pooling capabilities[2] [13, 20, 64]. A CXL pod [85] consists of multiple hosts within a rack, allowing these hosts to dynamically allocate memory from the pool based on demands. Recent work shows how to build CXL pods with hardware available today for about $600 per host [32] *without* using expensive CXL switches (§3). This enables positive return-on-investment (ROI) for memory-pooling use cases [31].

Once CXL pods are deployed, they can do much more than pooling memory — they can serve as a building block for implementing PCIe device pooling in software (Figure 1). PCIe devices and hosts access CXL memory just like any other memory. So, devices can directly read from or write to I/O buffers on CXL memory. CPUs from multiple hosts can transparently access these I/O buffers.

Implementing an efficient data path for PCIe device pooling in software brings multiple challenges. First, CXL increases access latency by 2-3× compared to local DDR5 memory [73, 76]. We need to carefully assess the performance impact of placing I/O datapaths in CXL memory. Second, although the CXL 3.0 specification introduces hardware coherence flows across hosts [5], CXL memory pool devices available today are not cache-coherent across hosts. We must therefore implement our own software coherence.

We argue that these challenges can be overcome with hardware available today. A small fraction of memory from the CXL pool serves as (software-coherent) shared memory accessible to multiple hosts. We show that a PCIe device attached to one CPU can DMA to/from shared memory and that an application on another CPU — to which the device is not connected via PCIe — can access IO buffer data in shared memory. Latency and bandwidth overheads are within 5%.

To enable remote hosts to access PCIe device memory (e.g., send MMIO commands or ring MMIO doorbells), we implement a sub-μs-scale host-to-host communication mechanism

[1] E.g., M6i supported local NVMe drives, M7i did not support local NVMe drives and was introduced in 2023 [28]. AWS later introduced I7ie [29] with local NVMe drives.

[2] Although advanced pooling features (e.g., Dynamic Capacity Device (DCD), Back Invalidation (BI)) are only standardized in the recent CXL 2.0 and 3.0 specification [4, 5], the widely adopted CXL 1.1 specification already provides the necessary functionalities for CXL memory pools (§3)

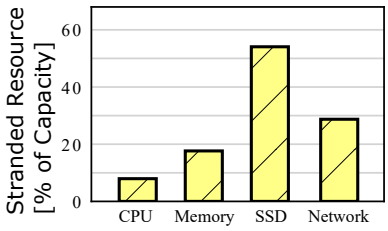


Figure 2: Percentages of stranded CPU cores, memory capacity, SSD storage, and NIC bandwidth in Microsoft Azure datacenters.

based on CXL shared memory. To the best of our knowledge, this is the first paper to demonstrate shared-memory communication latency using a real CXL memory pool. Prior work assumes cross-host cache coherence and *emulates* CXL pools using local memory [60, 62, 83]. This building block is used to forward device memory operations from remote hosts to the *local host* where the devices are physically attached.

Our software design of PCIe device pooling offers unique benefits in terms of costs and flexibility: First, CXL memory pools are inherently cheaper than PCIe switches due to switch-less pod designs [32]. Second, PCIe pooling reuses the exact same hardware used for memory pooling use cases. We can essentially enable PCIe pooling at no extra cost once CXL memory pools are deployed. Third, this software-based approach can dynamically support dynamic assignment of PCIe devices to hosts including potential fail-over scenarios. Fourth, the software solution can be easily extended to support new PCIe devices without hardware changes.

In this paper, we first motivate PCIe device pooling (§2) and provide background on CXL memory pools (§3). We then sketch our design of software-based PCIe pooling with CXL (§4). Lastly, we discuss how PCIe device pooling will change the datacenter infrastructure and other open questions, including how the CXL link bandwidth affects different PCIe pooling use cases (§5).

2 WHY IS PCIE POOLING A GOOD IDEA?

2.1 Stranded I/O Resources

Without pooling, PCIe devices are only accessible to a single host. When a host depletes one type of resource (e.g., CPU cores, memory, SSD storage, or NIC bandwidth), the remaining unutilized I/O resources are effectively *stranded*, as additional workloads cannot be scheduled.

An key contributor to stranding is the heterogeneity of workloads, e.g., of Virtual Machine (VM) types in public clouds [3, 7, 19], which leads to a multi-dimensional bin-packing problem [38, 41, 72]. A host accepts new VMs until it fills up along one dimension (e.g., memory), leaving the other dimensions stranded (e.g., SSD capacity or NIC bandwidth). For example, Microsoft Azure reports stranding across multiple resources (Figure 2) [71]. We find that SSD capacity and NIC bandwidth are the two most stranded resources with 54% and 29% being stranded on average, respectively.

By pooling resources among N servers, the effective bin's shape becomes more flexible, so fewer resources become fully saturated in one dimension while sitting idle in others. If demands across servers are somewhat independent, a spike in one server's demand for a resource (e.g., lots of SSD usage) may be offset by lower demand in another server. So, in aggregate, fewer resources remain stranded.

As a rough estimate, queueing theory typically shows a square-root improvement in resource overprovisioning when demands are aggregated over N hosts. Specifically, if demands across servers were independent, then the fraction of stranded resources would decrease with $\sqrt{N}$ [47, 80]. So, pooling across even just $N = 8$ servers would reduce SSD stranding from 54% to 19% and NIC stranding from 29% to 10% in Microsoft Azure. If scheduling dependencies (e.g., availability zones and other constraints [16, 18, 41]) lead to correlated high demands being colocated in a rack, pooling could be less effective — however, prior industry data has not observed such strong correlations [31].

2.2 PCIe Device Failures

Many cloud servers use only a single NIC to keep costs low. If this NIC fails, the entire server becomes unreachable and needs to wait for repair [59]. Providers can design servers with redundant PCIe devices (e.g., NICs) to ensure availability during hardware failures. However, PCIe devices are expensive and power-intensive. Additionally, since device failures are not very common [59], redundancy exacerbates stranding.

Pooling PCIe devices solves this problem by sharing redundant devices across multiple hosts. If a device fails, the pool can allocate a replacement to the affected host. This reduces the number of redundant devices needed, lowers costs, and improves device utilization.

3 CXL MEMORY POOLS

CXL standardizes interconnect protocols between processors and memory devices. We focus on CXL.mem which enables cores and devices to transparent load/store (and DMA) to device memory [5, 73]. CXL establishes a link based on the PCIe physical layer, ports, and cables. It achieves low latency with custom link and transaction layers. All major server CPUs (Intel, AMD, ARM) support CXL today.

Latency and bandwidth. The idle load-to-use latency of CXL memory is about 2-3× the latency of local DDR5 memory. For example, recent measurements show 2.15× idle latency on a Leo CXL Smart Memory Controller [73]. In

terms of bandwidth, a CXL 2.0/PCIe-5.0 ×8 link matches the bandwidth of a DDR5-4800 channel at a typical 2:1 read to write ratio (30 GB/s). CPUs can interleave at 256 Byte granularity across multiple CXL links to achieve higher bandwidth. On Intel Xeon 6 (Granite Rapids), we can interleave across 64 CXL lanes per CPU socket [49, 51, 53, 69, 77], providing ≈ 240 GB/s.

CXL memory pools. A CXL memory pool is a set of CXL memory devices that connect to multiple hosts, allowing them to dynamically allocate memory from the pool. The set of hosts connected to a CXL pool is called a CXL pod [45].

CXL memory pools can be constructed either by CXL devices with multiple ports or by CXL switches. Devices with multiple CXL ports, or *multi-headed devices (MHD)*, can connect to multiple hosts via dedicated CXL links [13, 20, 32, 33, 36, 40, 57, 61, 64]. These devices are readily available today from AsteraLabs [52], Marvell [64], and UnifabriX [20]. A single MHD today has up to 20 CXL ports [20] and multiple MHDs can be combined to scale to larger CXL pod sizes [32].

On the other hand, CXL switches allow CXL devices (including devices with a single port) to be connected to a CXL switch, which in turn connects to multiple hosts. Switched designs resemble tree network topologies in CXL 2.0 [50] and Clos network topologies in CXL 3.0 [65]. A single switch typically offers 128-256 CXL lanes [21]. These lanes would be shared among connected CPUs, CXL controllers, and other devices. For example, 128 lanes could connect six CPUs with x16 CXL ports and two single-ported CXL controllers with x16 ports. In CXL 3.0, multiple switching levels can be used to build large-scale CXL pods with 4096 hosts [73].

The tradeoff for switched designs is higher power, cost, and latency. Prior work observes that CXL switch costs can exceed the benefits that can be achieved from pooling at scale [31, 56]. CXL switch latency is also fundamentally higher than direct CXL connections, as switches require serialization and deserialization of CXL packets twice on every path from CPU to CXL memory controller [31, 57]. Current switches add more than 250*ns* of latency [48], resulting in idle load-to-use latency of roughly $500 - 600ns$.

In short, CXL switches are more costly and have higher latency overhead, so MHD-based pods will likely be deployed earlier [31, 32, 56].

Shared CXL memory. A CXL memory pool can be used as shared memory across all the hosts in a CXL pod. Shared CXL memory has the potential to speed up RPCs, key-value stores, and distributed databases [25, 44, 45, 60, 62, 75, 79, 83],

Cache coherence. The CXL 3.0 specification introduces Back-Invalidate (BI), a hardware flow to implement cache coherence across multiple hosts [5, 73]. With BI, a CXL 3.0 memory device can implement a snoop filter to track host caching and issue snoop requests to change the cache state of a host. However, implementing BI involves both processor-side and device-side changes, which greatly increases hardware complexity and costs. Neither CPUs nor CXL memory pool devices support BI today.

4 DESIGNING PCIE POOLS IN SOFTWARE

We propose the following design goals to make our design practical and effective in unlocking the benefits of pooling:

(1) **Immediately Deployable.** CXL memory pool devices available today do not have hardware cache coherence across hosts. Thus, pooling should work without coherence.
(2) **Device Compatibility.** Our design should be compatible with most PCIe devices (e.g., NICs, SSDs, accelerators).
(3) **Load Shifting.** To realize load balancing and failovers, our design should support adjusting device-to-host mappings and shift loads between devices dynamically.

Our design consists of two components: (i) **the datapath**, which decouples PCIe devices from hosts by allowing hosts to access remote PCIe devices not directly connected to them via PCIe (§4.1); and (ii) **the pooling orchestrator**, which manages the mapping between PCIe devices and hosts (§4.2).

Our design assumes containerized environments. The datapath is implemented in the host userspace I/O stacks, and the pooling orchestrator runs as a special management container on one of the hosts in the CXL pod.

Throughout this section, we use NICs as example PCIe devices as they exhibit lower latency and higher bandwidth compared to SSDs, making them more challenging to pool. Nevertheless, our design is compatible with other PCIe devices, including SSDs and accelerators such as FPGAs.

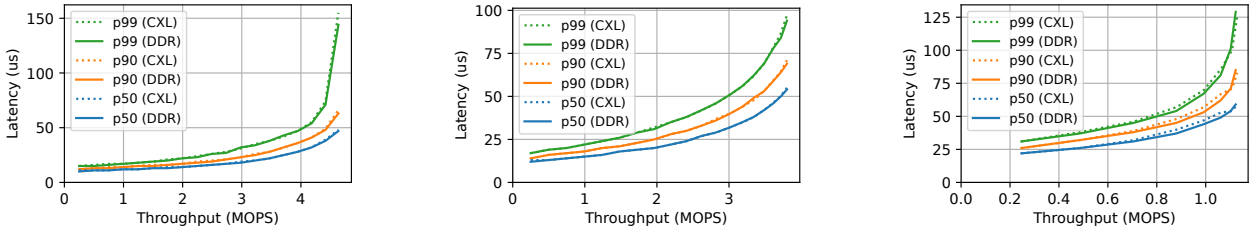
4.1 PCIe Datapath Over CXL

A CXL pool offers a shared memory range for connected hosts and their PCIe devices, which enables us to route PCIe traffic through the CXL pool across host boundaries.

Placing buffers in shared CXL memory. Routing PCIe traffic through a CXL memory pool requires placing I/O-related buffers in the CXL pool, so that remote PCIe devices can access these buffers via DMA.

For non-cache-coherent CXL pools, the datapath should explicitly maintain coherence in software. When modifying I/O buffers, the data should always be written to the CXL memory rather than staying in the CPU caches. Otherwise, other hosts might retrieve stale data from the CXL memory.

Performance implications of CXL. Although the latency overhead incurred by CXL could affect the I/O buffers placed in CXL memory pools, this overhead is negligible compared to the end-to-end I/O latency of NICs and SSDs.



(a) 75 B packets. (b) 1500 B packets. (c) 9000 B packets.

Figure 3: Latency-throughput graph of the UDP microbenchmark with 100 Gbps NICs. On the server side, TX and RX buffers are allocated either from the CXL memory pool (dotted lines) or local DDR5 memory (solid lines).

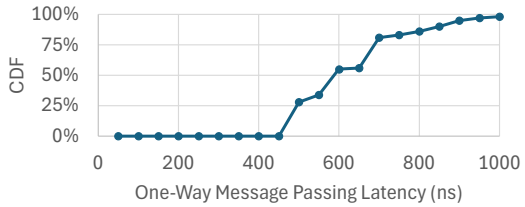


Figure 4: Latency distribution of message passing.

To quantify if placing I/O buffers in CXL affects I/O latency, we set up a two-socket server equipped with a 100 Gbps Mellanox ConnectX-5 NIC and connect both of its CPUs to a MHD-based CXL pod with each a PCIe-5.0 ×8 link. A second host serves as load generator (client) and connects via another 100 Gbps ConnectX-5 NIC to a common 100 Gbps switch. We use Junction [37] as the network stack for both the server and the client. We modify Junction on the server side to allocate TX and RX buffers (not the TX/RX queues) from the CXL memory pool. The NIC connects to socket0 and uses one ×8 CXL link. Junction runs on socket1 and uses the other ×8 CXL link.

We perform a UDP microbenchmark with varying payload sizes and compare the round-trip network latency of our modified Junction to an unmodified version that allocates memory locally and runs on socket0. Figure 3 shows that, although CXL has higher access latency, placing TX/RX buffers in CXL has negligible effects on the network latency. Maximum throughput is also not affected because the two PCIe-5.0 ×8 links provide enough bandwidth to saturate 100 Gbps NICs. Scaling to two 400 Gbps NICs may be possible on a Xeon 6 CXL configuration (Section 3). However, our proposal may work better for general-purpose computing than high-bandwidth HPC or ML use cases.

Event signaling and host-to-host communications. Signaling events such as request arrivals often requires accessing PCIe device memory. As a host cannot directly access a remote device's memory via MMIO, the datapath should provide a communication channel to forward MMIO operations to the host where the device is physically attached.

We prototype a shared-memory communication channel in shared CXL memory. The channel is implemented as a ring buffer, with each message slot sized at 64 B to match the cacheline granularity. It manages cache coherence in software by using non-temporal stores to send messages.

We measure its latency using a ping-pong test. The sender and receiver each connect to the CXL memory pool using a PCIe-5.0 ×16 link. Figure 4 shows that shared-memory channels in CXL achieve sub-μs latencies without cache coherence. The median latency is around 600 ns, slightly above the theoretical minimum latency for message passing, which equals the total latency of one CXL write and one CXL read.

4.2 Pooling Orchestrator

The pooling orchestrator manages a PCIe device pool. It handles control plane operations, including allocating PCIe devices to hosts, monitoring resource usage and health status of each PCIe device, and migrating workloads between devices to balance load or handle device failures.

Each host runs a pooling agent that monitors and configures the PCIe device. The orchestrator and the agents communicate using shared-memory channels in the shared CXL memory. Both roles are good candidates to offload to accelerators such as SmartNICs.

When allocating a PCIe device to a host, the orchestrator first checks if the host has a local PCIe device that is below a load threshold. If not, the orchestrator selects the least-utilized device in the pod to balance load.

If a PCIe device fails (e.g., due to NIC link failures) or becomes overloaded, the corresponding agent will report the issue to the orchestrator using the shared-memory CXL channel. The orchestrator can then migrate workloads from the affected device to other devices.

5 DISCUSSION

The adoption of software-based PCIe pooling introduces numerous potential research opportunities.

Datacenter networks without ToRs. Traditional datacenter designs have had a single top-of-rack (ToR) switch. All servers in the rack connect to this ToR. Links fan out from that ToR to multiple aggregation switches. The ToR to aggregation links are typically oversubscribed [39, 67, 74].

If a ToR fails, the entire rack becomes unreachable. Datacenter operators have thus started to deploy dual ToRs, which avoids the single point of failure but increases cost [68, 81].

This creates an interesting opportunity: can we eliminate the ToRs altogether? Instead of oversubscribing at the ToR level, we can provision sufficient NICs within each CXL pod to provide equivalent oversubscription, and then directly connect these NICs to multiple switches within the aggregation layer. This allows us to work around both ToR failures and NIC failures. This would require high CXL pod reliability.

Highly-available CXL pods. There are multiple ways to build CXL pods (Section 3). MHD-based pods typically use multiple MHDs and thus inherently offer high redundancy. A recent Microsoft white paper formalizes this with so-called dense topologies that offer λ redundant paths within a CXL pool [32]. Many industry proposals offer $\lambda = 4$ or even $\lambda = 8$ [36, 40, 46, 57].

In the future, CXL 3.0 switches will explicitly support multi-path CXL topologies with Port Based Routing [73].

While CXL pod hardware may offer redundancy, correctly implementing redundancy and fail-over is a hard implementation challenge. However, we believe that software implementations would likely still be significantly easier than hardware implementations based on PCIe switches.

Soft accelerator disaggregation. The architecture community has proposed and continues to propose an ever increasing set of accelerators. Typically, they come in the form of PCIe cards [35]. Deploying an accelerator that is used by a large number of people (e.g., compression) is easy. However, accelerators increasingly target highly specific workloads such as homomorphic encryption [82]. If a user wants to use an accelerator, they need to run on a server with this accelerator. Specialized accelerators may get infrequent use and thus may sit idle most of the time, which is a significant cost overhead. Disaggregating accelerators through PCIe switches [9, 10] is complex and expensive. Our approach allows exposing these accelerators as a disaggregated resource accessed via a CXL-connected pool.

PCIe pooling can complement RDMA-based storage disaggregation. Storage disaggregation over RDMA is common in clouds. However, in storage clusters, SSDs and NICs remain tightly coupled to hosts. Each host is provisioned with SSDs and NICs to handle peak demands. Since storage clusters often exhibit skewed access patterns, NICs are mostly underutilized. PCIe pooling can address this issue by pooling NICs between multiple storage nodes, improving resource efficiency in RDMA-based disaggregation.

Better host load balancing. PCIe pooling can also facilitate load balancing for resources other than PCIe devices. Typically, the challenge for moving load to another server is that TCP connections are assigned at setup and cannot be moved [34, 42]. A long-lived connection on a server whose load changes in the background may experience high latency. Recent work has explored TCP connection migration to move requests to new servers after they are being processed, but this work requires programmable switches or other network middleboxes [34, 42]. Our virtual NIC approach could implement the transformations required to migrate connections seamlessly within the CXL pod.

CXL link bandwidth. Recently platforms (e.g., Intel Xeon 6) provide 64 CXL 2.0 / PCIe5 lanes per CPU socket [49, 51, 53, 69, 77]. Fully disaggregating a 200 Gbps NIC and a 400 Gbps NIC requires only 8 and 16 CXL lanes, respectively. Therefore, pooling NICs to improve NIC utilization and reduce the NIC-to-host ratio can be supported by allocating enough CXL lanes per host to interleave between multiple MHDs. Similarly, disaggregating six NVMe SSDs (current AWS servers offer six local NVMe SSDs [24], and datacenter SSDs today often provide 5 GB/s bandwidth [22, 23]) would require 30 GB/s bandwidth, which can be satisfied with 8 CXL lanes.

However, enabling a single host to saturate a large number of NICs to maximize its peak network performance does pose higher requirements for the CXL link [84]. For example, to drive the combined bandwidth of eight 400 Gbps NICs in a CXL pod, the host would need at least 100 CXL 2.0 lanes, making this use case less realistic.

Operational implications. Datacenters often need to update host OSes and platform firmware, which requires rebooting individual hosts. To enable such maintenance, PCIe device pooling should support hot-adding and hot-removing hosts from a CXL pod, allowing updates to be rolled out incrementally on a per-host basis. For instance, when reconfiguring a host in public clouds, the VM scheduler should live migrate all VMs off the host and notify the pooling orchestrator to hot-remove it. Upon removal, the orchestrator should prevent new device allocations to the host and migrate its existing PCIe device assignments to other active hosts.

6 ACKNOWLEDGMENTS

We thank the reviewers for their helpful comments. We also thank Rodrigo Fonseca and Stefan Saroiu for their feedback. This work was supported by NSF award CNS-2143868.

REFERENCES

[1] A New Twist on Pci-Express Switching for the Datacenter. https://gigaio.com/2019/10/a-new-twist-on-pci-express-switching-for-the-datacenter/.

[2] AI Server Cost Analysis. https://semianalysis.com/2023/05/29/ai-server-cost-analysis-memory-is/.

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

[4] Compute Express Link (CXL) 2.0 Specification. https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-2.0-Specification.pdf.

[5] Compute Express Link (CXL) 3.0 Specification. https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-3.0-Specification.pdf.

[6] Counting the Cost of Under-Utilized GPUs And Doing Something About It. https://gigaio.com/2020/11/counting-the-cost-of-under-utilized-gpus-and-doing-something-about-it/.

[7] General-purpose machine family for Compute Engine. https://cloud.google.com/compute/docs/general-purpose-machines.

[8] GigaIO: FabreX AI Memory Fabric Platform. https://gigaio.com/products/fabrex-system-overview/.

[9] GigaIO – Accelerator Pooling Appliance. https://gigaio.com/products/accelerator-pooling-appliance/.

[10] GigaIO — Storage Pooling Appliance. https://gigaio.com/products/storage-pooling-appliance/.

[11] H3 Composable AI Solutions. https://www.h3platform.com/solution/composable-ai.

[12] Instance store temporary block storage for ec2 instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html. Accessed: 2025-01-13.

[13] Leo CXL® Smart Memory Controllers. https://www.asteralabs.com/products/leo-cxl-smart-memory-controllers/.

[14] Liqid SmartStack. https://www.liqid.com/products/gpu-on-demand.

[15] Microchip Switchtec PCIe Switches. https://www.microchip.com/en-us/products/interface-and-connectivity/pcie-switches.

[16] Move Azure virtual machines into Availability Zones. https://learn.microsoft.com/en-us/azure/site-recovery/move-azure-vms-avset-azone.

[17] NVMe over Fabrics (oF) Specification. https://nvmexpress.org/specification/nvme-of-specification/.

[18] Regions and Zones. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[19] Sizes for virtual machines in Azure. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/overview.

[20] UnifabriX launches memory pooling system. https://www.unifabrix.com/resources/unifabrix-launches-memory-pooling-system.

[21] XConn CXL2.0/PCIe5.0 switch. https://www.xconn-tech.com/product.

[22] 990 EVO 2 TB SSD NVMe M.2 product specifications, 2025. Available at https://www.samsung.com/us/computing/memory-storage/solid-state-drives/990-evo-nvme-ssd-2tb-mz-v9e2t0b-am/, accessed 4/16/25.

[23] Solidigm D5-P5430 product specifications, 2025. Available at https://www.solidigm.com/products/data-center/d5/p5430.html#configurator, accessed 4/16/25.

[24] Specifications for Amazon EC2 general purpose instances, 2025. Available at https://docs.aws.amazon.com/ec2/latest/instancetypes/gp.html#gp_network, accessed 4/16/25.

[25] Minseon Ahn, Thomas Willhalm, Norman May, Donghun Lee, Suprasad Mutalik Desai, Daniel Booss, Jungmin Kim, Navneet Singh, Daniel Ritter, and Oliver Rebholz. An examination of cxl memory use cases for in-memory database management systems using sap hana. *Proceedings of the VLDB Endowment*, 17(12):3827–3840, 2024.

[26] Pradeep Ambati, Inigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing SLOs for Resource-Harvesting VMs in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 735–751. USENIX Association, November 2020.

[27] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49–67, Boston, MA, April 2023. USENIX Association.

[28] Jeff Barr. New seventh-generation general purpose amazon ec2 instances (m7i-flex and m7i). https://aws.amazon.com/blogs/aws/new-seventh-generation-general-purpose-amazon-ec2-instances-m7i-flex-and-m7i/, 2023. Accessed: 2025-01-13.

[29] Jeff Barr. Now available: Storage optimized amazon ec2 i7ie instances. https://aws.amazon.com/blogs/aws/now-available-storage-optimized-amazon-ec2-i7ie-instances/, 2024. Accessed: 2025-01-13.

[30] Luiz André Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 2009.

[31] Daniel S Berger, Daniel Ernst, Huaicheng Li, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Lisa Hsu, Ishwar Agarwal, Mark D Hill, et al. Design tradeoffs in cxl-based memory pools for public cloud platforms. *IEEE Micro*, 43(2):30–38, 2023.

[32] Daniel S. Berger, Yuhong Zhong, Pantea Zardoshti, Shuwei Teng, Fiodar Kazhamiaka, and Rodrigo Fonseca. Octopus: Scalable low-cost cxl memory pooling, 2025.

[33] Prakash Chauhan, Chris Petersen, Brian Morris, and Jerome Glisse. Hyperscale tiered memory expander specification for compute express link. Available at https://www.opencompute.org/documents/hyperscale-tiered-memory-expander-specification-for-compute-express-link-cxl-1-pdf, 2023. Open Compute Project, Revision 1, Effective October 27, 2023.

[34] Inho Choi, Nimish Wadekar, Raj Joshi, Joshua Fried, Dan RK Ports, Irene Zhang, and Jialin Li. Capybara: μsecond-scale live tcp migration. In *Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 30–36, 2023.

[35] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, Karthik Gururaj, and Glenn Reinman. Accelerator-rich architectures: Opportunities and progresses. In *Proceedings of the 51st annual design automation conference*, pages 1–6, 2014.

[36] Mohamad El-Batal. Seagate Composable Memory Appliance (CMA) Architecture. https://www.youtube.com/watch?v=KCgE0WejXl0, June 2024.

[37] Joshua Fried, Gohar Irfan Chaudhry, Enrique Saurez, Esha Choukse, Inigo Goiri, Sameh Elnikety, Rodrigo Fonseca, and Adam Belay. Making kernel bypass practical for the cloud with junction. In *21st USENIX*

Symposium on Networked Systems Design and Implementation (NSDI 24), pages 55–73, Santa Clara, CA, April 2024. USENIX Association.

[38] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.*, 44(4):455–466, August 2014.

[39] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[40] Minho Ha, Junhee Ryu, Jungmin Choi, Kwangjin Ko, Sunwoong Kim, Sungwoo Hyun, Donguk Moon, Byungil Koh, Hokyoon Lee, Myoungseo Kim, et al. Dynamic capacity service for improving cxl pooled memory efficiency. *IEEE Micro*, 43(2):39–47, 2023.

[41] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Protean: VM allocation service at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 845–861. USENIX Association, November 2020.

[42] Yutaro Hayakawa, Michio Honda, Douglas Santry, and Lars Eggert. Prism: Proxies without the pain. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 535–549, 2021.

[43] Wentao Hou, Jie Zhang, Zeke Wang, and Ming Liu. Understanding routable PCIe performance for composable infrastructures. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 297–312, Santa Clara, CA, April 2024. USENIX Association.

[44] Yibo Huang, Haowei Chen, Newton Ni, Vijay Chidambaram, Dixin Tang, and Emmett Witchel. Tigon: A distributed database for a CXL pod. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, Boston, MA, July 2025. USENIX Association.

[45] Yibo Huang, Newton Ni, Vijay Chidambaram, Emmett Witchel, and Dixin Tang. Pasha: An efficient, scalable database architecture for cxl pods. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2024.

[46] Ronen Hyatt. The quest for bandwidth and capacity: Memory edition, 2023. https://www.hpcuserforum.com/wp-content/uploads/2023/09/Ronen-Hyatt_UnifabriX_The-Quest-for-Bandwidth-and-Capacity-Memory-Edition_Sept-2023-HPC-UF.pdf.

[47] A. J. E. M. Janssen and Johan S. H. van Leeuwaarden. Refining square-root safety staffing by expanding erlang c. *Operations Research*, 59(6):1512–1522, 2011. Provides refinements to the square-root safety staffing rule using corrected diffusion approximations for the Erlang C formula.

[48] JP Jiang. CXL Switch for Scalable & Composable Memory Pooling/Sharing. FMS presentation available at https://www.xconn-tech.com/products, 2024.

[49] Michael Kalodrich. New supermicro x14 systems, 2024. Accessed: 2024-12-18.

[50] Jim Kao. CXL 2.0 Switch for a Composable Memory System. https://computeexpresslink.org/wp-content/uploads/2024/09/Xconn_CXL-2.0-Switch-for-a-Composable-Memory-System_FMS-2024_FINAL.pdf, October 2024.

[51] Patrick Kennedy. Lenovo has a cxl memory monster with 128x 128gb ddr5 dimms, 2024. Accessed: 2024-12-18.

[52] Astera Labs. Leo cxl smart memory controllers. Available at https://www.asteralabs.com/products/leo-cxl-smart-memory-controllers/, December 2023. Product Brief.

[53] Lenovo. Lenovo thinksystem sr860 v3 server, 2024. Accessed: 2024-12-18.

[54] Alberto Lerner and Gustavo Alonso. Cxl and the return of scale-up database engines. *Proc. VLDB Endow.*, 17(10):2568–2575, August 2024.

[55] Alberto Lerner and Gustavo Alonso. Cxl and the return of scale-up database engines. *arXiv preprint arXiv:2401.01150*, 2024.

[56] Philip Levis, Kun Lin, and Amy Tai. A case against cxl memory pooling. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, page 18–24, New York, NY, USA, 2023. Association for Computing Machinery.

[57] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo 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, page 574–587, New York, NY, USA, 2023. Association for Computing Machinery.

[58] Qiang Li, Qiao Xiang, Yuxin Wang, Haohao Song, Ridi Wen, Wenhui Yao, Yuanyuan Dong, Shuqi Zhao, Shuo Huang, Zhaosheng Zhu, et al. More than capacity: Performance-oriented evolution of pangu in alibaba. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 331–346, 2023.

[59] Jialun Lyu, Marisa You, Celine Irvene, Mark Jung, Tyler Narmore, Jacob Shapiro, Luke Marshall, Savyasachi Samal, Ioannis Manousakis, Lisa Hsu, et al. Hyrax:{Fail-in-Place} server operation in cloud platforms. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 287–304, 2023.

[60] Teng Ma, Zheng Liu, Chengkun Wei, Jialiang Huang, Youwei Zhuo, Haoyu Li, Ning Zhang, Yijin Guan, Dimin Niu, Mingxing Zhang, and Tao Ma. HydraRPC: RPC in the CXL era. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 387–395, Santa Clara, CA, July 2024. USENIX Association.

[61] Grant Mackey. You don't know 'jack': Cxl fabric orchestration and management. Available at https://files.futurememorystorage.com/proceedings/2024/20240806_CXLT-102-1_Mackey.pdf, 2024. Presented by Jackrabbit Labs.

[62] Suyash Mahar, Ehsan Hajyjasini, Seungjin Lee, Zifeng Zhang, Mingyao Shen, and Steven Swanson. Telepathic datacenters: Fast rpcs using shared cxl memory, 2024.

[63] Hasan Al Maruf, Yuhong Zhong, Hongyi Wang, Mosharaf Chowdhury, Asaf Cidon, and Carl Waldspurger. Memtrade: Marketplace for disaggregated memory clouds. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(2), May 2023.

[64] Inc. Marvell Technology. Structera a 2504 memory-expansion controller. Available at https://www.marvell.com/content/dam/marvell/en/public-collateral/assets/marvell-structera-a-2504-near-memory-accelerator-product-brief.pdf, 2024. Product Brief, P/N MV-SLA25041-A0-HF350AA-C000.

[65] Danny Moore and Debendra Das Sharma. CXL 3.0: Enabling composable systems with expanded fabric capabilities. https://computeexpresslink.org/wp-content/uploads/2023/12/CXL_3.0-Webinar_FINAL.pdf, October 2022.

[66] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. Ssd failures in datacenters: What? when? and why? In *Proceedings of the 9th ACM International on Systems and Storage Conference*, pages 1–11, 2016.

[67] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 66–85, 2022.

[68] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 691–706, 2024.

[69] ASRock Rack. Gnrd8-2l2t preliminary ceb specifications, 2024. Accessed: 2024-12-18.

[70] Benjamin Reidys, Jinghan Sun, Anirudh Badam, Shadi Noghabi, and Jian Huang. BlockFlex: Enabling storage harvesting with Software-Defined flash in modern cloud platforms. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 17–33, Carlsbad, CA, July 2022. USENIX Association.

[71] Benjamin Reidys, Pantea Zardoshti, Íñigo Goiri, Celine Irvene, Daniel S. Berger, Haoran Ma, Kapil Arya, Eli Cortez, Taylor Stark, Eugene Bak, Mehmet Iyigun, Stanko Novaković, Lisa Hsu, Karel Trueba, Abhisek Pan, Saravan Bansal, Chetan Rajmohan, Jian Huang, and Ricardo Bianchini. Coach: Exploiting temporal patterns for all-resource oversubscription in cloud platforms. In *Proceedings of the 30th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '25, New York, NY, USA, 2025. Association for Computing Machinery.

[72] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 69–87, Carlsbad, CA, October 2018. USENIX Association.

[73] Debendra Das Sharma, Robert Blankenship, and Daniel S. Berger. An introduction to the Compute Express Link (CXL) interconnect. *ACM Computing Surveys (CSUR)*, 2024.

[74] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM computer communication review*, 45(4):183–197, 2015.

[75] Joshua Suetterlein, Joseph Manzano, and Andres Marquez. Synchronization for CXL based memory. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '24, page 178–185, New York, NY, USA, 2024. Association for Computing Machinery.

[76] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 105–121, New York, NY, USA, 2023. Association for Computing Machinery.

[77] KAYTUS Systems. Kr2280v3 platform intel amd, 2024. Accessed: 2024-12-18.

[78] Jaylen Wang, Daniel S. Berger, Fiodar Kazhamiaka, Celine Irvene, Chaojie Zhang, Esha Choukse, Kali Frost, Rodrigo Fonseca, Brijesh Warrier, Chetan Bansal, Jonathan Stern, Ricardo Bianchini, and Akshitha Sriraman. Designing cloud servers for lower carbon. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 452–470, 2024.

[79] Zhao Wang, Yiqi Chen, Cong Li, Yijin Guan, Dimin Niu, Tianchan Guan, Zhaoyang Du, Xingda Wei, and Guangyu Sun. CTXNL: A software-hardware co-designed solution for efficient cxl-based transaction processing. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, page 192–209, New York, NY, USA, 2025. Association for Computing Machinery.

[80] Ward Whitt. Understanding the efficiency of multi-server service systems. *Management Science*, 38(5):708–723, 1992. Analyzes the efficiency of multi-server systems and discusses the implications of the square-root staffing rule.

[81] Ying Xie and Guohan Lu. Dualtor evolution: Active-active dualtor. https://drive.google.com/file/d/1RtYYk1JHv7WnyABrUVj7FiRMZu2KEYtL/view, October 2023. Accessed: 2025-01-13.

[82] Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Ximeng Liu, and Kai Chen. Sok: Fully homomorphic encryption accelerators. *ACM Computing Surveys*, 56(12):1–32, 2024.

[83] Mingxing Zhang, Teng Ma, Jinqi Hua, Zheng Liu, Kang Chen, Ning Ding, Fan Du, Jinlei Jiang, Tao Ma, and Yongwei Wu. Partial failure resilient memory management system for (cxl-based) distributed shared memory. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 658–674, New York, NY, USA, 2023. Association for Computing Machinery.

[84] Xu Zhang, Ke Liu, Yisong Chang, Ke Zhang, and Mingyu Chen. Dfabric: Scaling out data parallel applications with cxl-ethernet hybrid interconnects, 2024.

[85] Zhiting Zhu, Newton Ni, Yibo Huang, Yan Sun, Zhipeng Jia, Nam Sung Kim, and Emmett Witchel. Challenges and opportunities for systems using cxl memory. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–2, 2024.