Memory Disaggregation: Advances and Open Challenges

Hasan Al Maruf, Mosharaf Chowdhury
SymbioticLab, University of Michigan

Abstract

Compute and memory are tightly coupled within each server in traditional datacenters. Large-scale datacenter operators have identified this coupling as a root cause behind fleet-wide resource underutilization and increasing Total Cost of Ownership (TCO). With the advent of ultra-fast networks and cache-coherent interfaces, *memory disaggregation* has emerged as a potential solution, whereby applications can leverage available memory even outside server boundaries.

This paper summarizes the growing research landscape of memory disaggregation from a software perspective and introduces the challenges toward making it practical under current and future hardware trends. We also reflect on our seven-year journey in the SymbioticLab to build a comprehensive disaggregated memory system over ultra-fast networks. We conclude with some open challenges toward building next-generation memory disaggregation systems leveraging emerging cache-coherent interconnects.

1 Introduction

Modern datacenter applications – low-latency online services, big data analytics, and AI/ML workloads alike – are often memory-intensive. As the number of users increases and we collect more data in cloud datacenters, the overall memory demand of these applications continue to rise. Despite their performance benefits, memory-intensive applications experience *disproportionate* performance loss whenever their working sets do not completely fit in the available memory. For instance, our measurements across a range of memory-intensive applications show that if half their working sets do not fit in memory, their performance can drop by 8× to 25× [22].

Application developers often sidestep such disasters by over-allocating memory, but pervasive over-allocation inevitably leads to datacenter-scale memory underutilization. Indeed, memory utilization at many hyperscalers hovers around 40%–60% [1, 22, 25, 40]. Service providers running on public clouds, such as Snowflake, report 70%–80% underutilized memory on average [48]. Since DRAM is a significant driver of infrastructure cost and power consumption [33], excessive underutilization leads to high TCO.

At the same time, increasing the effective memory capacity and bandwidth of each server to accommodate ever-larger working sets is challenging as well. In fact, memory bandwidth is a bigger bottleneck than memory capacity today as the former increases at a slower rate. For example, to increase memory bandwidth by 3.6× in their datacenters, Meta had to increase capacity by 16× [33]. To provide sufficient memory capacity and/or bandwidth to memory-intensive applications, computing and networking resources become stranded in traditional server platforms, which eventually causes fleet-wide resource underutilization and increases TCO.

Memory disaggregation addresses memory-related rightsizing problems at both software and hardware levels. Applications are able to allocate memory as they need without being constrained by server boundaries. Servers are not forced to add more computing and networking resources when they only need additional memory capacity or bandwidth. By exposing all unused memory across all the servers as a memory pool to all memory-intensive applications, memory disaggregation can improve both application-level performance and overall memory utilization. Multiple hardware vendors and hyperscalers have projected [9, 10, 28, 33] up to 25% TCO savings without affecting application performance via (rack-scale) memory disaggregation.

While the idea of leveraging remote machines' memory is decades old [15, 18, 20, 29, 31, 35], only during the past few years, the latency and bandwidth gaps between memory and communication technologies have come close enough to make it practical. The first disaggregated memory[1] solutions (Infiniswap [22] and the rest) leveraged RDMA over InfiniBand or Ethernet, but they are an order-of-magnitude slower than local memory. To bridge this performance gap and to address practical issues like performance isolation, resilience, scalability, etc., we have built a comprehensive set of software solutions. More recently, with the rise of cache-coherent Compute Express Link (CXL) [3] interconnects and hardware protocols, the gap is decreasing even more. We are at the cusp of taking a leap toward next-generation software-hardware co-designed disaggregated memory systems.

This short paper is equal parts a quick tutorial, a retrospective on the Infiniswap project summarizing seven years' worth of research, and a non-exhaustive list of future predictions based on what we have learned so far.

2 Memory Disaggregation

Simply put, memory disaggregation exposes memory capacity available in remote locations as a pool of memory and shares it across multiple servers over the network. It decouples the available compute and memory resources, enabling independent resource allocation in the cluster. A server's lo-

[1] Remote memory and far memory are often used interchangeably with the term disaggregated memory.

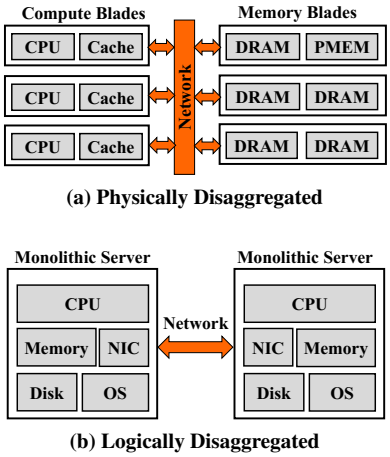

Figure 1: Physical vs. logical memory disaggregation architectures.

cal and remote memory together constitute its total physical memory. An application's locality of memory reference allows the server to exploit its fast local memory to maintain high performance, while remote memory provides expanded capacity with an increased access latency that is still orders-of-magnitude faster than accessing persistent storage (e.g., HDD, SSD). The OS and/or application runtime provides the necessary abstractions to expose all the available memory in the cluster, hiding the complexity of setting up and accessing remote memory (e.g., connection setup, memory access semantics, network packet scheduling, etc.) while providing resilience, isolation, security, etc. guarantees.

2.1 Architectures

Memory disaggregation systems have two primary cluster memory architectures.

Physical Disaggregation. In a physically-disaggregated architecture, compute and memory nodes are detached from each other where a cluster of compute blades are connected to one or more memory blades through network (e.g., PCIe bridge) [30] (Figure 1a). A memory node can be a traditional monolithic server with low compute resource and large memory capacity, or it can be network-attached DRAM. For better performance, the compute nodes are usually equipped with a small amount of memory for caching purposes.

Logical Disaggregation. In a logically-disaggregated architecture, traditional monolithic servers hosting both compute and memory resources are connected to each other through the network (e.g., Infiniband, RoCEv2) (Figure 1b). This is a popular approach for building a disaggregated memory system because one does not need to change existing hardware architecture; simply incorporating appropriate software to provide a remote memory interface is sufficient. In such a setup, usually, each of the monolithic servers has their own OS. In some cases, the OS itself can be disaggregated across multiple hosts [44]. Memory local to a host is usually prioritized for running local jobs. Unutilized memory on remote machines can be pooled and exposed to the cluster as remote [14, 22, 27, 32, 34, 42, 44].

Table 1: Selected memory disaggregation proposals.

Abstraction	System	Hardware Transparent	OS Transparent	Application Transparent
Virtual Memory Management (VMM)	Global Memory [19]	Yes	No	Yes
	Memory Blade [30]	No	No	Yes
	Infiniswap [22]	Yes	Yes	Yes
	Leap [32]	Yes	No	Yes
	LegoOS [44]	Yes	No	Yes
	zSwap [25]	Yes	No	Yes
	Kona [14]	Yes	No	Yes
	Fastswap [12]	Yes	No	Yes
	Hydra [27]	Yes	Yes	Yes
Virtual File System (VFS)	Memory Pager [31]	Yes	Yes	No
	Remote Regions [11]	Yes	Yes	No
Custom API	FaRM [17]	Yes	Yes	No
	FaSST [24]	Yes	Yes	No
	Memtrade [34]	Yes	Yes	No
Programming Runtime	AIFM [42]	Yes	Yes	No
	Semeru [49]	Yes	Yes	No

Hybrid Approach. Cache-coherent interconnects like CXL provides the opportunity to build a composable heterogeneous server memory systems that combine logical and physical disaggregation approaches. Multiple monolithic servers, compute devices, memory nodes, or network specialized devices can be connected through fabric or switches where software stacks can provide the cache-line granular or traditional virtual memory-based disaggregated memory abstraction.

2.2 Abstractions and Interfaces

Interfaces to access disaggregated memory can either be transparent to the application or need minor to complete re-write of applications (Table 1). The former has broader applicability, while the latter might have better performance.

Application-Transparent Interface. Access to remote disaggregated memory without significant application rewrites typically relies on two primary mechanisms: disaggregated Virtual File System (VFS) [11], that exposes remote memory as files and disaggregated Virtual Memory Manager (VMM) for remote memory paging [22, 27, 32, 44]. In both cases, data is communicated in small chunks or pages (typically, 4KB). In case of remote memory as files, pages go through the file system before they are written to/read from the remote memory. For remote memory paging and distributed OS, page faults cause the VMM to write pages to and read them from the remote memory. Remote memory paging is more suitable for traditional applications because it does not require software or hardware modifications.

Non-Transparent Interface. Another approach is to directly expose remote memory through custom API (KV-store, remote memory-aware library or system calls) and modify the applications incorporating these specific APIs [17, 24, 34, 41, 42, 49]. All the memory (de)allocation, transactions, synchronizations, etc. operations are handled by the underlying implementations of these APIs. Performance op-

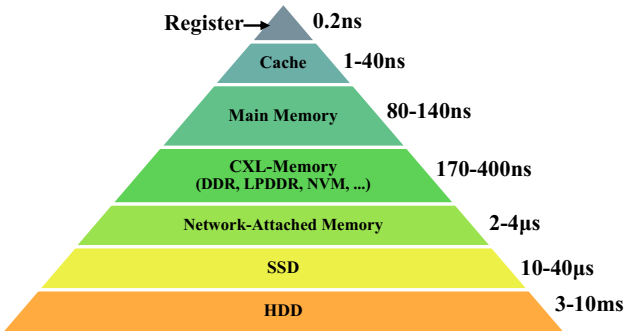

Figure 2: Latency characteristics of different memory technologies.

timizations like caching, local-vs-remote data placement, prefetching, etc. are often the responsibility of the application.

2.3 Challenges in Practical Memory Disaggregation

Simply relying on fast networks or interconnects is not sufficient to make memory disaggregation practical. A comprehensive solution must address challenges in multiple dimensions:

- **High Performance.** A disaggregated memory system involves the network in its remote memory path, which is at least an order-of-magnitude slower than memory channels attached to CPU and DRAM (80–140 nanoseconds vs. microseconds; see Figure 2). Hardware-induced remote memory latency is significant and impacts application performance [22, 32, 33]. Depending on the abstraction, software stacks can also introduce significant overheads. For example, remote memory paging over existing OS VMM can add tens of microseconds latency for a 4KB page [32].
- **Performance Isolation.** When multiple applications with different performance requirements (e.g., latency- vs. bandwidth-sensitive workloads) compete for disaggregated memory, depending on where the applications are running and where the remote memory is located, they may be contending for resources inside the server, on the NIC, and in the network on the hardware side and variety of resources in the application runtimes and OSes. This is further exacerbated by the presence of multiple tiers of memory with different latency-bandwidth characteristics.
- **Memory Heterogeneity.** Memory hierarchy within a server is already heterogeneous (Figure 2). Disaggregated memory – both network-attached and emerging CXL memory [21, 28, 33] – further increases heterogeneity in terms of latency-bandwidth characteristics. In such a heterogeneous setup, simply allocating memory to applications is not enough. Instead, decisions like how much memory to allocate in which tier at what time is critical as well.
- **Resilience to Expanded Failure Domains.** Applications relying on remote memory become susceptible to new failure scenarios such as independent and correlated failures of remote machines, evictions from and corruptions of remote memory, and network partitions. They also suffer from stragglers or late-arriving remote responses due to network congestion and background traffic [16]. These uncertainties can lead to catastrophic failures and service-level objective (SLO) violations.
- **Efficiency and Scalability.** Disaggregated memory systems are inherently distributed. As the number of memory servers, the total amount of disaggregated memory, and the number of applications increase, the complexity of finding unallocated remote memory in a large cluster, allocating them to applications without violating application-specific SLOs, and corresponding meta-data overhead of memory management increase as well. Finding efficient matching at scale is necessary to achieve high overall utilization.
- **Security.** Although security of disaggregated memory is often sidestepped within the confines of a private datacenter, it is a major challenge for memory disaggregation in public clouds. Since data residing in remote memory may be read by entities without proper access, or corrupted from accidents or malicious behavior, the confidentiality and integrity of remote memory must be protected. Additional concerns include side channel and remote rowhammer attacks over the network [45, 46], distributed coordinated attacks, lack of data confidentiality and integrity and client accountability during CPU bypass operations (e.g., when using RDMA for memory disaggregation).

3 Infiniswap: A Retrospective

To the best of our knowledge, Infiniswap is the first memory disaggregation system with a comprehensive and cohesive set of solutions for all the aforementioned challenges. It addresses host-level, network-level, and end-to-end aspects of practical memory disaggregation over RDMA. At a high level, Infiniswap provides a paging-based remote memory abstraction that can accommodate any application without changes, while providing a high-performance yet resilient, isolated, and secure data path to remote disaggregated memory.

Bootstrapping. Our journey started in 2016, when we simply focused on building an application-transparent interface to remote memory that are distributed across many servers. Infiniswap [22] transparently exposed remote disaggregated memory through paging without any modifications to applications, hardware, or OSes of individual servers. It employed a block device with traditional I/O interface to VMM. The block device divided its whole address space into smaller slabs and transparently mapped them across many servers' remote memory. Infiniswap captured 4KB page faults in runtime and redirected them to remote memory using RDMA.

From the very beginning, we wanted to design a system that would scale without losing efficiency down the line. To this end, we designed decentralized algorithms to identify free memory, to distribute memory slabs, and to evict slabs for memory reclamation. This removed the overhead of centralized meta-data management without losing efficiency.

Improving Performance. Infiniswap's block layer-based paging caused high latency overhead during remote memory accesses. This happens because Linux VMM is not optimized for microsecond-scale operations. We gave up one degree of freedom and designed Leap [32] in 2018 – we modified the OS to optimize the remote memory data path by identifying and removing non-critical functionalities while paging.

Even with the leanest data path, a reactive page fetching system must suffer microsecond-scale network latency on the critical path. Leap introduced a remote memory prefetcher to proactively bring in the correct pages into a local cache to provide sub-microsecond latency (comparable to that of a local page access) on cache hits.

Providing Resilience. Infiniswap originally relied on local disks to tolerate remote failures, which resulted in slow failure recovery. Maintaining multiple in-memory replicas was not an option either as it effectively halved the total capacity. We started exploring erasure coding as a memory-efficient alternative. Specifically, we divided each page into k splits to generate r encoded parity splits and spread the $(k+r)$ splits to $(k+r)$ failure domains – any k out of $(k+r)$ splits would then suffice to decode the original data. However, erasure coding was traditionally applied to large objects [38]. By 2019/20, we built Hydra [27] whose carefully designed data path could perform online erasure coding within a single-digit microsecond tail latency. Hydra also introduced CodingSets, a new data placement scheme that balanced availability and load balancing, while reducing the probability of data loss by an order of magnitude even under large correlated failures.

Multi-Tenancy Issues. We observed early on (circa 2017) that accessing remote memory over a shared network suffers from contention in the NIC and inside the network [54]. While our optimized data paths in Leap and Hydra could address some of the challenges inside the host, they did not extend to resource contentions in the RDMA NIC (RNIC). We finished designing Justitia [56] in 2020 to improve the network bottleneck in RNICs by transparently monitoring the latency profiles of each application and providing performance isolation. More recently, we have looked into improving Quality-of-Service (QoS) inside the network as well [55].

Expanding to Public Clouds. While Infiniswap and related projects were designed for cooperative private datacenters, memory disaggregation in public clouds faces additional concerns. In 2021, we finished designing Memtrade [34] to harvest all the idle memory within virtual machines (VMs) – be it unallocated, or allocated to an application but infrequently utilized, and exposed them to a disaggregated memory marketplace. Memtrade allows producer VMs to lease their idle application memory to remote consumer VMs for a limited period of time while ensuring confidentiality and integrity. It employs a broker to match producers with consumers while satisfying performance constraints.

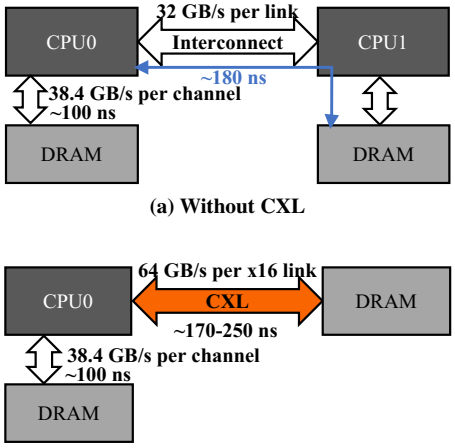

Figure 3: A CXL system compared to a dual-socket server.

Detours Along the Way. Throughout this journey, we collaborated on side quests like designing a decentralized resource management algorithm using RDMA primitives [51], meta-data management inside the network using programmable switches [53], fine-grained compute disaggregation [52] etc. Some of our forays into designing hardware support were nipped in the bud, often because we could not find the right partners. In hindsight, perhaps we were fortunate given how quickly the industry converged on CXL.

Summing it Up. Infiniswap together with all its extensions can provide near-memory performance for most memory-intensive applications even when 75% and sometimes more of their working sets reside in remote disaggregated memory in an application- and hardware-transparent manner, in the presence of failures, load imbalance, and multiple tenants. After seven years, we declared victory on this chapter in 2022.

4 Hardware Trend: Cache-Coherent Interconnects

Although networking technologies like InfiniBand and Ethernet continue to improve, their latency remain considerably high for providing a cache-coherent memory address space across disaggregated memory devices. CXL (Compute Express Link) [3] is a new processor-to-peripheral/accelerator cache-coherent interconnect protocol that builds on and extends the existing PCIe protocol by allowing coherent communication between the connected devices.[2] It provides byte-addressable memory in the same physical address space and allows transparent memory allocation using standard memory allocation APIs. It also allows cache-line granularity access to the connected devices and underlying hardware maintains cache-coherency and consistency. With PCIe 5.0, CPU-to-

[2]Prior industry standards in this space such as CCIX [2], OpenCAPI [8], Gen-Z [5] etc. have all come together under the banner of CXL consortium. While there are some related research proposals (e.g., [26]), CXL is the de facto industry standard at the time of writing this paper.

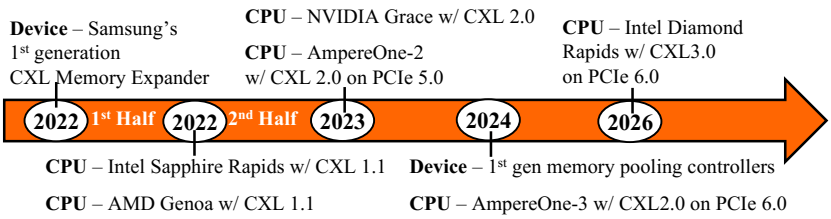

Figure 4: CXL roadmap paves the way for memory pooling and disaggregation in next-generation datacenter design.

CXL interconnect bandwidth is similar to the cross-socket interconnects (Figure 3) on a dual-socket machine [57]. CXL-Memory access latency is also similar to the NUMA access latency. CXL adds around 50-100 nanoseconds of extra latency over normal DRAM access.

CXL Roadmap. Today, CXL-enabled CPUs and memory devices support CXL 1.0/1.1 (Figure 4) that enables a point-to-point link between CPUs and accelerator memory or between CPUs and memory extenders. CXL 2.0 spec enables one-hop switching that allows multiple accelerators without (*Type-1 device*) or with memory (*Type-2 device*) to be configured to a single host and have their caches be coherent to the CPUs. It also allows memory pooling across multiple hosts using memory expanding devices (*Type-3 device*). A CXL switch has a fabric manager (it can be on-board or external) that is in charge of the device address-space management. Devices can be hot-plugged to the switch. A virtual CXL switch partitions the CXL-Memory and isolate the resources between multiple hosts. It provides telemetry for load on each connected devices for load balancing and QoS management.

CXL 3.0 adds multi-hop hierarchical switching – one can have any complex types of network through cascading and fan-out. This expands the number of connected devices and the complexity of the fabric to include non-tree topologies, like Spine/Leaf, mesh- and ring-based architectures. CXL 3.0 supports PCIe 6.0 (64 GT/s i.e., up to 256 GB/s of throughput for a x16 duplex link) and expand the horizon of very complex and composable rack-scale server design with varied memory technologies (Figure 5). A new Port-Based Routing (PBR) feature provides a scalable addressing mechanism that supports up to 4,096 nodes. Each node can be any of the existing three types of devices or the new Global Fabric Attached Memory (GFAM) device that supports different types of memory (i.e., Persistent Memory, Flash, DRAM, other future memory types, etc.) together in a single device. Besides memory pooling, CXL 3.0 enables memory sharing across multiple hosts on multiple end devices. Connected devices (i.e., accelerators, memory expanders, NICs, etc.) can do peer-to-peer communicate bypassing the host CPUs.

In essence, CXL 3.0 enables large networks of memory devices. This will proliferate software-hardware co-designed memory disaggregation solutions that not only simplify and better implement previous-generation memory disaggregation solutions (e.g., Infiniswap) but also open up new possibilities.

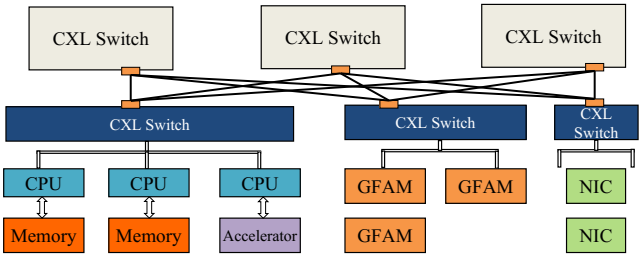

Figure 5: CXL 3.0 enables a rack-scale server design with complex networking and composable memory hierarchy.

5 Disaggregation Over Intra-Server CXL

With the emergence of new hardware technologies comes the opportunity to rethink and revisit past design decisions, and CXL is no different. Earlier software solutions for memory disaggregation over RDMA are not optimized enough in CXL-based because of its much lower latency bound, especially for intra-server CXL (CXL 1.0/1.1) with 100s of nanoseconds latency. Recent works in leveraging CXL 1.0/1.1 within a server have focused on (tiered) memory pooling [28, 33] because a significant portion of datacenter application working sets can be offloaded to a slower-tier memory without hampering performance [25, 33, 34]. We have recently worked on two fundamental challenges in this context.

Memory Usage Characterization. Datacenter applications have diverse memory access latency and bandwidth requirements. Sensitivity toward different memory page types can also vary across applications. Understanding and characterizing such behaviors is critical to designing CXL-enabled heterogeneous tiered-memory systems. Chameleon [33] is a lightweight user-space memory access behavior characterization tool that can readily be deployed in production without disrupting running application(s) or modifying the OS kernel. It utilizes the Precise Event-Based Sampling (PEBS) mechanism of modern CPU's Performance Monitoring Unit (PMU) to collect hardware-level performance events related to memory accesses. It then generates a heat-map of memory usage for different page types and provides insights into an application's expected performance with multiple temperature tiers.

Memory Management. Given applications' page characterizations, TPP [33] provides an OS-level transparent page placement mechanism, to efficiently place pages in a tiered-

memory system. TPP has three components: **(a)** a lightweight reclamation mechanism to demote colder pages to the slow tier; **(b)** decoupling the allocation and reclamation logic for multi-NUMA systems to maintain a headroom of free pages on the fast tier; and **(c)** a reactive page promotion mechanism that efficiently identifies hot pages trapped in the slow memory tier and promote them to the fast memory tier to improve performance. It also introduces support for page type-aware allocation across the memory tiers.

6 CXL-Disaggregated Memory at Rack-Scale and Beyond: Open Challenges

Although higher than intra-server CXL latency, rack-scale CXL systems with a CXL switch (CXL 2.0) will experience much lower latency than RDMA-based memory disaggregation. With a handful of hops in CXL 3.0 setups, latency will eventually reach a couple microseconds similar to that found in today's RDMA-based disaggregated memory systems. For next-generation memory disaggregation systems that operate between these two extremes, i.e., rack-scale and a little beyond, many open challenges exist. We may even have to revisit some of our past design decisions (§2). Here we present a non-exhaustive list of challenges informed by our experience.

6.1 Abstractions

Memory Access Granularity. CXL enables cache-line granular memory access over the connected devices, whereas existing OS VMM modules are designed for page-granular (usually, 4KB or higher) memory access. Throughout their lifetimes, applications often write a small part of each page; typically only 1-8 cache-lines out of 64 [14]. Page-granular access causes large dirty data amplification and bandwidth overuse. In contrast, fine-grained memory access over a large memory pool causes high meta-data management overhead. Based on an application's memory access patterns, remote memory abstractions should support transparent and dynamic adjustments to memory access granularity.

Memory-QoS Interface. Traditional solutions for memory page management focus on tracking (a subset of) pages and counting accesses to determine the heat of the page and then moving pages around. While this is enough to provide a two-level, hot-vs-cold QoS, it cannot capture the entire spectrum of page temperature. Potential solutions include assigning a QoS level to (1) an entire application; (2) individual data structures; (3) individual `mmap()` calls; or even (4) individual memory accesses. Each of these approaches have their pros and cons. At one extreme, assigning a QoS level to an entire application maybe simple, but it cannot capture time-varying page temperature of large, long-running applications. At the other end, assigning QoS levels to individual memory accesses requires recompilation of all existing applications as well as cumbersome manual assignments, which can lead to erroneous QoS assignments. A combination of aforementioned approaches may reduce developer's overhead while providing sufficient flexibility to perform spatiotemporal memory QoS management.

6.2 Management and Runtime

Memory Address Space Management. From CXL 2.0 onward, devices can be hot-plugged to the CXL switches. Device-attached memory is mapped to the system's coherent address space and accessible to host using standard write-back semantics. Memory located on a CXL device can either be mapped as Host-managed Device Memory (HDM) or Private Device Memory (PDM). To update the memory address space for connected devices to different host devices, a system reset is needed; traffic towards the device needs to stop to alter device address mapping during this reset period. An alternate solution to avoid this system reset is to map the whole physical address space to each host when a CXL-device is added to the system. The VMM or fabric manager in the CXL switch will be responsible to maintain isolation during address-space management. How to split the whole address-space in to sizable memory blocks for the efficient physical-to-virtual address translation of a large memory network is an interesting challenge [26, 53].

Unified Runtime for Compute Disaggregation. CXL Type-2 devices (accelerator with memory) maintains cache coherency with the CPU. CPU and Type-2 devices can interchangeably use each other's memory and both get benefited. For example, applications that run on CPUs can benefit as they can now access very high bandwidth GPU memory. Similarly, for GPU users, it is beneficial for capacity expansion even though the memory bandwidth to and from CPU memory will be lower. In such a setup, remote memory abstractions should track the availability of compute cores and efficiently perform near-memory computation to improve the overall system throughput.

Future datacenters will likely be equipped with numerous domain-specific compute resources/accelerators. In such a heterogeneous system, one can borrow the idle cores of one compute resource and perform extra computation to increase the overall system throughput. A unified runtime to support malleable processes that can be immediately decomposed into smaller pieces and offloaded to any available compute nodes can improve both application and cluster throughput [41, 52].

6.3 Allocation Policies

Memory Allocation in Heterogenous NUMA Cluster. For better performance, hottest pages need to be on the fastest memory tier. However, due to memory capacity constraints across different tiers, it may not always be possible to utilize the fastest or performant memory tier. Determining what fraction of memory is needed at a particular memory tier to maintain the desired performance of an application at different points of its life cycle is challenging. This is even more difficult when multiple applications coexist. Efficient promo-

tion or demotion of pages of different temperatures across memory tiers at rack scale is necessary. One can consider augmenting TPP by incorporating a lightweight but effective algorithm to select the migration target considering node distances from the CPU, load on CPU-memory bus, current load on different memory tiers, network state, and the QoS requirements of the migration-candidate pages.

Allocation Policy for Memory Bandwidth Expansion. For memory bandwidth-bound applications, CPU-to-DRAM bandwidth often becomes the bottleneck and increases the average memory access latency. CXL's additional memory bandwidth can help by spreading memory across the top-tier and remote nodes. Instead of only placing cold pages into CXL-Memory, which has low bandwidth consumption, an ideal solution should place the right amount of bandwidth-heavy, latency-insensitive pages to CXL-Memory. The methodology to identify the ideal fraction of such working sets may even require hardware support.

Memory Sharing and Consistency. CXL 3.0 allows memory sharing across multiple devices. Through an enhanced coherency semantics, multiple hosts can have a coherent copy of a shared segment, with back invalidation for synchronization. Memory sharing improves application-level performance by reducing unnecessary data movement and improves memory utilization. Sharing a large memory address space, however, results in significant overhead and complexity in the system that plagued classic distributed shared memory (DSM) proposals [36]. Furthermore, sharing memory across multiple devices increases the security threat in the presence of any malicious application run on the same hardware space. We believe that disaggregated memory systems should cautiously approach memory sharing and avoid it unless it is absolutely necessary for specific scenarios.

6.4 Rack-Level Objectives

Rack-Scale Memory Temperature. To obtain insights into an application's expected performance with multiple temperature tiers, it is necessary to understand the heat map of memory usage for that application. Existing hot page identification mechanisms (including Chameleon) are limited to a single host OS or user-space mechanism. They either use access bit-based mechanism [4, 6, 47], special CPU feature-based (e.g., Intel PEBS) tools [39, 43, 50], or OS features [7, 33] to determine the page temperature within a single server. So far, there is no distributed mechanism to determine the cluster-wide relative page temperature. Combining the data of all the OS or user-space tools and coordinating between them to find rack-level hot pages is an important problem. CXL fabric manager is perhaps the place where one can get a cluster-wide view of hardware counters for each CXL device's load, hit, and access-related information. One can envision extending Chameleon for rack-scale environments to provide observability into each application's per-device memory temperature.

Hardware-Software Co-Design for a Better Ecosystem. Hardware features can further enhance performance of memory disaggregation systems in rack-scale setups. A memory-side cache and its associated prefetcher on the CXL ASIC or switch might help reduce the effective latency of CXL-Memory. Hardware support for data movement between memory tiers can help reduce page migration overheads in an aggressively provisioned system with very small amount of local memory and high amount of CXL-Memory. Additionally, the fabric manager of a CXL switch should implement policies like fair queuing, congestion control, load balancing etc. for better network management. Incorporating Leap's prefetcher and Hydra's erasure-coded resilience ideas into CXL switch designs can enhance system-wide performance.

Energy- and Carbon-Aware Memory Disaggregation. Datacenters represent a large and growing source of energy consumption and carbon emissions [13]. Some estimates place datacenters to be responsible for 1-2% of aggregate worldwide electricity consumption [23, 37]. To reduce the TCO and carbon footprint, and enhance hardware life expectancy, datacenter rack maintain a physical energy budget or power cap. Rack-scale memory allocation, demotion, and promotion policies can be augmented by incorporating energy-awareness in their decision-making process. In general, we can introduce energy-awareness in the software stack that manage compute, memory, and network resources in a disaggregated cluster.

7 Conclusion

We started the Infiniswap project in 2016 with the conviction that memory disaggregation is inevitable, armed only with a few data points that hinted it might be within reach. As we conclude this paper in 2023, we have successfully built a comprehensive software-based disaggregated memory solution over ultra-fast RDMA networks that can provide a seamless experience for most memory-intensive applications. With diverse cache-coherent interconnects finally converging under the CXL banner, the entire industry (and ourselves) are at the cusp of taking a leap toward next-generation software-hardware co-designed disaggregated systems. Join us. Memory disaggregation is here to stay.

Acknowledgements

Juncheng Gu, Youngmoon Lee, and Yiwen Zhang co-led different aspects of the Infiniswap project alongside the authors. We thank Yiwen Zhang for his feedback on this paper. Special thanks to our many collaborators, contributors, users, and cloud resource providers (namely, CloudLab, Chameleon Cloud, and UM ConFlux) for making Infiniswap successful. Our expeditions into next-generation memory disaggregation solutions have greatly benefited from our collaborations with Meta. Our research was supported in part by National Science Foundation grants (CCF-1629397, CNS-1845853, and CNS-2104243) and generous gifts from VMware and Meta.

References

[1] Alibaba Cluster Trace 2018. https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace_2018.md.

[2] CCIX. https://www.ccixconsortium.com/.

[3] Compute Express Link (CXL). https://www.computeexpresslink.org/.

[4] DAMON: Data Access MONitoring Framework for Fun and Memory Management Optimizations. https://www.linuxplumbersconf.org/event/7/contributions/659/attachments/503/1195/damon_ksummit_2020.pdf.

[5] Gen-Z. https://genzconsortium.org/.

[6] Idle page tracking-based working set estimation. https://lwn.net/Articles/460762/.

[7] NUMA Balancing (AutoNUMA). https://mirrors.edge.kernel.org/pub/linux/kernel/people/andrea/autonuma/autonuma_bench-20120530.pdf.

[8] OpenCAPI. https://opencapi.org/.

[9] Rack-scale computing at Yahoo! http://www.intel.com/content/dam/www/public/us/en/documents/presentation/idf15-yahoo-rack-scale-computing-presentation.pdf.

[10] Tencent explores datacenter resource-pooling using Intel rack scale architecture (Intel RSA). http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/rsa-tencent-paper.pdf.

[11] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, S. Novaković, A. Ramanathan, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei. Remote regions: a simple abstraction for remote memory. In *USENIX ATC*, 2018.

[12] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker. Can far memory improve job throughput? In *EuroSys*, 2020.

[13] T. Anderson, A. Belay, M. Chowdhury, A. Cidon, and I. Zhang. Treehouse: A case for carbon-aware datacenter software. In *HotCarbon*, 2022.

[14] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli. Rethinking software runtimes for disaggregated memory. In *ASPLOS*, 2021.

[15] H. Chen, Y. Luo, X. Wang, B. Zhang, Y. Sun, and Z. Wang. A transparent remote paging model for virtual machines. In *International Workshop on Virtualization Technology*, 2008.

[16] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[17] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast remote memory. In *NSDI*, 2014.

[18] S. Dwarkadas, N. Hardavellas, L. Kontothanassis, R. Nikhil, and R. Stets. Cashmere-VLM: Remote memory paging for software distributed shared memory. In *IPPS/SPDP*, 1999.

[19] M. J. Feeley, W. E. Morgan, E. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing global memory management in a workstation cluster. In *ACM SIGOPS Operating Systems Review*, volume 29, pages 201–212. ACM, 1995.

[20] E. W. Felten and J. Zahorjan. Issues in the implementation of a remote memory paging system. Technical Report 91-03-09, University of Washington, Mar 1991.

[21] D. Gouk, S. Lee, M. Kwon, and M. Jung. Direct access, High-Performance memory disaggregation with DirectCXL. In *USENIX ATC*, 2022.

[22] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin. Efficient memory disaggregation with Infiniswap. In *NSDI*, 2017.

[23] N. Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 561:163–166, 2018.

[24] A. Kalia, M. Kaminsky, and D. G. Andersen. FaSST: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In *OSDI*, 2016.

[25] A. Lagar-Cavilla, J. Ahn, S. Souhlal, N. Agarwal, R. Burny, S. Butt, J. Chang, A. Chaugule, N. Deng, J. Shahid, G. Thelen, K. A. Yurtsever, Y. Zhao, and P. Ranganathan. Software-defined far memory in warehouse-scale computers. In *ASPLOS*, 2019.

[26] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong, and A. Bhattacharjee. MIND: In-network memory management for disaggregated data centers. In *SOSP*, 2021.

[27] Y. Lee, H. A. Maruf, M. Chowdhury, A. Cidon, and K. G. Shin. Hydra : Resilient and highly available remote memory. In *FAST*, 2022.

[28] H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, 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 *ASPLOS*, 2023.

[29] S. Liang, R. Noronha, and D. K. Panda. Swapping to remote memory over InfiniBand: An approach using a high performance network block device. In *IEEE International Conference on Cluster Computing*, 2005.

[30] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. *SIGARCH*, 2009.

[31] E. P. Markatos and G. Dramitinos. Implementation of a reliable remote memory pager. In *USENIX ATC*, 1996.

[32] H. A. Maruf and M. Chowdhury. Effectively prefetching remote memory with Leap. In *USENIX ATC*, 2020.

[33] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan. TPP: Transparent page placement for CXL-enabled tiered-memory. In *ASPLOS*, 2023.

[34] H. A. Maruf, Y. Zhong, H. Wong, M. Chowdhury, A. Cidon, and C. Waldspurger. Memtrade: A disaggregated-memory marketplace for public clouds. In *SIGMETRICS*, 2023.

[35] T. Newhall, S. Finney, K. Ganchev, and M. Spiegel. Nswap: A network swapping module for Linux clusters. In *Euro-Par*, 2003.

[36] B. Nitzberg and V. Lo. Distributed shared memory: A survey of issues and algorithms. *Computer*, 24(8):52–60, 1991.

[37] F. Pearce. Energy hogs: Can world's huge data centers be made more efficient? *Yale Environment*, 2018.

[38] K. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran. EC-Cache: Load-balanced, low-latency cluster caching with online erasure coding. In *OSDI*, 2016.

[39] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter. HeMem: Scalable tiered memory management for big data applications and real NVM. In *SOSP*, 2021.

[40] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *SoCC*, 2012.

[41] Z. Ruan, S. J. Park, M. K. Aguilera, A. Belay, and M. Schwarzkopf. Nu: Achieving microsecond-scale resource fungibility with logical processes. In *NSDI*, 2023.

[42] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay. AIFM: High-performance, application-integrated far memory. In *OSDI*, 2020.

[43] H. Servat, A. J. Peña, G. Llort, E. Mercadal, H.-C. Hoppe, and J. Labarta. Automating the application data placement in hybrid memory systems. In *IEEE International Conference on Cluster Computing*, 2017.

[44] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *OSDI*, 2018.

[45] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *ATC*, 2018.

[46] S.-Y. Tsai, M. Payer, and Y. Zhang. Pythia: Remote oracles for the masses. In *USENIX Security*, 2019.

[47] Vladimir Davydov. Idle Memory Tracking. https://lwn.net/Articles/639341/.

[48] M. Vuppalapati, J. Miron, R. Agarwal, D. Truong, A. Motivala, and T. Cruanes. Building an elastic query engine on disaggregated storage. In *NSDI*, 2020.

[49] C. Wang, H. Ma, S. Liu, Y. Li, Z. Ruan, K. Nguyen, M. D. Bond, R. Netravali, M. Kim, and G. H. Xu. Semeru: A Memory-Disaggregated managed runtime. In *OSDI*, 2020.

[50] K. Wu, Y. Huang, and D. Li. Unimem: Runtime data managementon non-volatile memory-based heterogeneous main memory. In *SC*, 2017.

[51] D. Y. Yoon, M. Chowdhury, and B. Mozafari. Distributed lock management with RDMA: Decentralization without starvation. In *SIGMOD*, 2018.

[52] J. You, J. Wu, X. Jin, and M. Chowdhury. Ship compute or ship data? why not both? In *NSDI*, 2021.

[53] Z. Yu, Y. Zhang, V. Braverman, M. Chowdhury, and X. Jin. NetLock: Fast, centralized lock management using programmable switches. In *SIGCOMM*, 2020.

[54] Y. Zhang, J. Gu, Y. Lee, M. Chowdhury, and K. G. Shin. Performance isolation anomalies in RDMA. In *ACM SIGCOMM KBNets*, 2017.

[55] Y. Zhang, G. Kumar, N. Dukkipati, X. Wu, P. Jha, M. Chowdhury, and A. Vahdat. Aequitas: Admission control for performance-critical RPCs in datacenters. In *ACM SIGCOMM*, 2022.

[56] Y. Zhang, Y. Tan, B. Stephens, and M. Chowdhury. Justitia: Software Multi-Tenancy in hardware Kernel-Bypass networks. In *NSDI*, 2022.

[57] W. Zhao and J. Ning. Project Tioga Pass Rev 0.30 : Facebook Server Intel Motherboard V4.0 Spec. https://www.opencompute.org/documents/facebook-server-intel-motherboard-v40-spec.