This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

Design Tradeoffs in CXL-Based Memory Pools for Public Cloud Platforms

Daniel S. Berger*, Dan Ernst*, Huaicheng Li†, Pantea Zardoshti*, Monish Shah*, Samir Rajadnya*, Scott Lee*, Lisa Hsu*, Ishwar Agarwal‡, Mark D. Hill*, Ricardo Bianchini*
(*Microsoft Azure, †Virginia Tech, ‡Intel)

***Abstract*—DRAM is a key driver of performance and cost in public cloud servers. At the same time, a significant amount of DRAM is underutilized due to fragmented use across servers. Emerging interconnects such as CXL offer a path towards improving utilization through memory pooling. However, the design space of CXL-based memory systems is large, with key questions around the size, reach, and topology of the memory pool. At the same time, using pools requires navigating complex design constraints around performance, virtualization, and management. This paper discusses why cloud providers should deploy CXL memory pools, key design constraints, and observations in designing towards practical deployment. We identify configuration examples with significant positive return of investment.**

1. Introduction

Motivation. Many public cloud customers deploy their workloads via virtual machines (VMs). VMs enable performance comparable to on-premises datacenters without the need to manage datacenters. Cloud providers face the challenge of achieving excellent performance at a competitive hardware cost.

A key driver of both performance and cost is main memory. The gold standard for memory performance is to preallocate a VM with cores and memory on the same socket. This leads to memory latency below 100ns and facilitates virtualization acceleration. At the same time, DRAM has become a major portion of hardware cost due to its poor scaling properties with only nascent alternatives [1]. For example, DRAM can be over 50% of server cost [2].

Through analysis of Azure VM traces, we identify *memory stranding* as a dominant source of memory waste and a potential source of cost savings. Stranding happens when all server cores are rented (*i.e.*, allocated to customer VMs) but unallocated memory capacity remains and cannot be rented. We find that up to 30% of DRAM becomes stranded as more cores become allocated to VMs.

Limitations of the state-of-the-art. Reducing DRAM usage in the public cloud is challenging due to its stringent performance requirements. Pooling memory via memory disaggregation is a promising approach because stranded memory can be returned to the disaggregated pool and used by other servers. Unfortunately, existing pooling systems have microsecond access latencies and require page faults or changes to the VM guest [3, 4].

The emerging CXL interconnect. The emerging Compute Express Link (CXL) interconnect [5] enables cacheable load/store (`ld/st`) accesses to pooled memory on many current processors. Pool-memory accesses via loads/stores is a game

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

changer for cloud computing as it allows memory to remain statically preallocated while physically being located in a shared pool. However, CXL access latency depends on the overall system design, especially the pool size (the number of CPU sockets able to use a given pool) and topology. Larger pools require traversing switching levels, which adds significant latency. Additionally, each CXL component adds to the system cost, which must be balanced against stranding savings.

This work. This work builds on an ASPLOS 2023 paper [2] that identifies the memory stranding problem — we paraphrase this analysis as motivation in Section 3— and then focuses on system software policies and mechanisms for allocating/managing normal and pooled memory.

This work focuses on design tradeoffs in the pool's hardware configuration. First, we characterize pool components, possible topologies, and associate memory access latencies. We derive a set of design recommendations from this analysis. Second, we compare savings from memory pooling to the cost of its components for different pool sizes and CXL device types. We find that CXL-based memory pooling can yield significant positive returns on investment. Contrary to the focus of existing literature, smaller pools may be attractive. Third, we discuss future directions for the industry as well as academic research.

2. Background

Cloud resource allocation. Public cloud workloads run inside virtual machines (VMs). To offer performance close to dedicated (non-virtualized) resources, VM resources are statically allocated by reserving each resource (CPU, DRAM, network bandwidth, etc.) for a VM's lifetime. Additionally, providers optimize I/O performance with virtualization accelerators that bypass the hypervisor [6]. For example, accelerated networking is enabled by default on AWS and Azure. Virtualization acceleration requires statically preallocating (or "pinning") a VM's entire address space [7].

Cloud resource scheduling. Scheduling VMs with heterogeneous multi-dimensional resource demands onto servers leads to a challenging bin-packing problem [8, 9]. Scheduling is further complicated by constraints such as spreading VMs across multiple failure domains.

A simplified view of Azure's VM scheduling is that VMs are first assigned to a compute cluster and then placed on a specific server within this cluster. A cluster roughly corresponds to a row of racks with homogenous server configurations. We use the unit of a cluster to characterize our workloads.

Memory stranding. It is often difficult to provision servers that closely match the resource demands of the incoming VM mix. A common reason is that the DRAM-to-core ratio of a server that will last years must be determined at platform design time and is statically fixed over its lifetime. Additionally, fixed-size DIMMs over limited freedom in determining the DRAM-to-core ratio.

When the DRAM-to-core ratio of VM arrivals and a cluster's server resources do not match, tight packing becomes especially difficult. We define a resource as *stranded* when it is technically available to be rented to a customer, but is practically unavailable as some other resource has been exhausted. The typical scenario for *memory stranding* is that all cores have been rented, but there is still memory available in the server.

Reducing stranding via pooling. This work proposes to break the fixed hardware configuration of servers by disaggregating memory into a pool that is accessible by multiple hosts [10]. By dynamically reassigning memory to different hosts at different times, we can shift memory resources to where they are needed. Thus, we can provision servers close to the average DRAM-to-core ratios and tackle deviations via the memory pool.

Pooling via CXL. The CXL.mem protocol for `ld/st` memory semantics maps device memory to the system address space. Last-level cache (LLC) misses to CXL memory addresses translate into requests on a CXL port whose responses bring the missing cachelines. Similarly, LLC write-backs translate into CXL data writes. CXL memory is virtualized using hypervisor page tables and the memory-management unit and is thus compatible with virtualization acceleration.

CXL.mem uses PCIe's electrical interface with custom link and transaction layers for low latency. Intel measures CXL port latencies at 25ns round-trip [11]. With PCIe 5.0, the bandwidth

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

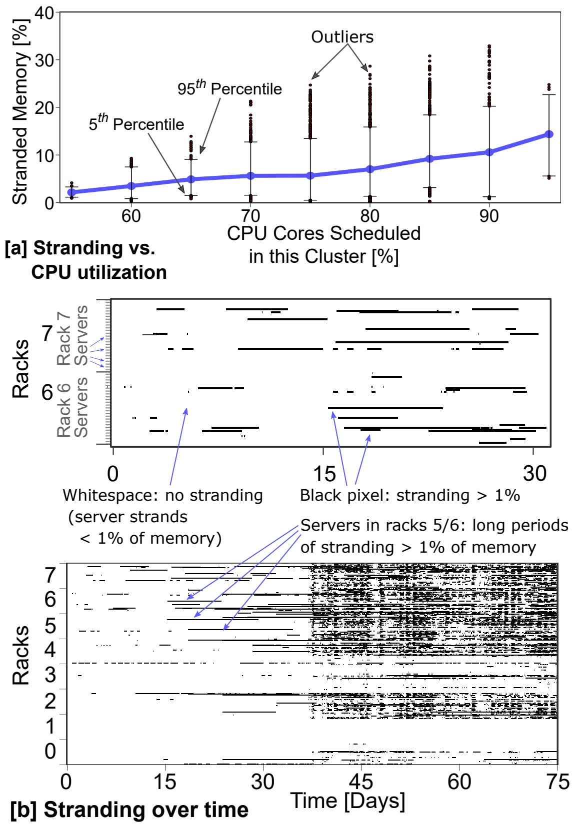

Figure 1: **Memory stranding. (a) Stranding increases significantly as more CPU cores are scheduled; (b) Stranding changes dynamically over time.**

of a bidirectional ×8-CXL port at a typical 2:1 read:write-ratio roughly matches an 80-bit DDR5-4800 channel.

3. Cloud Workload Characterization

3.1. Stranding at Azure

We summarize previous analysis on stranding [2].

Dataset. We measure stranding in 100 general-purpose clusters over a 75-day period. A general-purpose cluster hosts a mix of first-party and third-party VM workloads that do not require special hardware (such as GPUs). We select clusters with similar deployment years, spanning major regions on the planet. Each cluster trace contains millions of per-VM arrival/departure events.

Memory stranding. Figure 1a shows the hourly average amount of stranded DRAM across our cluster sample, bucketed by the percentage of scheduled CPU cores. In clusters where 75% of CPU cores are scheduled for VMs, 6% of memory is stranded. This grows to over 10% when ∼85% of CPU cores are allocated to VMs. This makes sense since stranding is an artifact of highly utilized nodes, which correlates with highly utilized clusters. Outliers are shown by the error bars, representing 5^{th} and 95^{th} percentiles. At 95^{th}, stranding reaches 25% during high utilization periods. Individual outliers reach more than 30% stranding.

Figure 1b shows stranding over time across eight adjacent racks. Every row shows a server within each rack. A workload change (around day 36) suddenly increased stranding significantly. Furthermore, stranding can affect many racks concurrently (*e.g.*, racks 2, 4–7) and it is generally hard to predict which clusters/racks will have stranded memory.

3.2. VM Memory Utilization in Azure

Dataset. We perform measurements on the same 100 general-purpose production clusters. For untouched memory, we rely on guest-reported memory usage counters cross-referenced with hypervisor page table access bit scans. We sample memory bandwidth counters using Intel RDT [12] for a subset of clusters with compatible hardware. Finally, we use hypervisor counters to measure *NUMA spanning* in dual-socket servers, where a VM has cores on one socket and some memory from another socket.

Memory bandwidth. Memory bandwidth usage of general-purpose workloads is generally low with average bandwidth utilization below 10 GB/s. VMs on a small number of hosts do, however, use 100% of memory bandwidth.

NUMA spanning. Most VMs are small and can fit on a single socket. Azure's hypervisor aims to schedule VMs on dual-socket servers such that they fit entirely (cores and memory) on a single NUMA node. We find that spanning occurs for only 2-3% of VMs.

Overall, untouched memory and low memory bandwidth requirements make VM workloads a good fit for memory pooling. However, with 97-98% of VMs using NUMA-local memory, performance parity for pooled memory will be challenging.

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

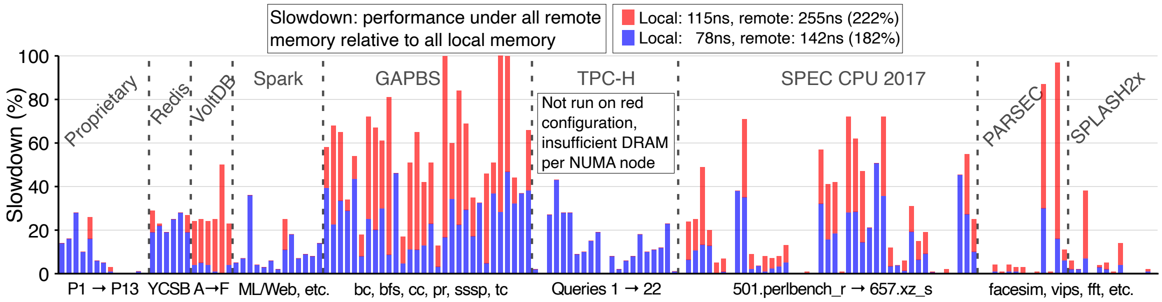

Figure 2: **Performance slowdowns when memory latency increases by 182-222% (§3.3). Workloads have different sensitivity to increased memory latency as they would see with CXL. X-axis shows 158 representative workloads; Y represents the normalized performance slowdown, *i.e.*, performance under higher (remote) latency relative to all local memory. "Proprietary" denotes production workloads at Azure.**

3.3. Workload Sensitivity to Memory Latency

We summarize previous experiments on latency sensitivity [2].

Dataset. We evaluate 158 workloads across proprietary workloads, in-memory stores, data processing, and benchmark suites. They run on dual-socket Intel Skylake 8157M, with a 182% latency increase for socket-remote memory, or AMD EPYC 7452, with 222% latency increase. We normalize performance as slowdown relative to NUMA-local performance.

Latency sensitivity. Figure 2 surveys workload slowdowns. Under a 182% increase in memory latency, we find that 26% of the 158 workloads experience less than 1% slowdown under CXL. At the same time, some workloads are severely affected with 21% of the workloads facing >25% slowdowns. Overall, every workload class has at least one workload with less than 5% slowdown and one workload with more than 25% slowdown (except SPLASH2x). Our proprietary workloads are less impacted than the overall workload set with almost half seeing <1% slowdown. These production workloads are NUMA-aware and often include data placement optimizations.

Under a 222% increase in memory latency, we find that 23% of the 158 workloads experience less than 1% slowdown under CXL. More than 37% of workloads face >25% slowdowns — a significantly higher fraction than on the 182% emulated latency increase. We find that the processing pipeline for some workloads, like VoltDB, seems to have just enough slack to accomodate the smaller 182% latency increase with significant pipeline stalls for 222% latency increase. Other workload classes like graph processing (GAPBS) are sensitive to both latency and bandwidth, and both effects are worsened on the 222% system.

4. The Memory Pool Design Space

Designing a memory pool involves multiple hardware components and design choices that expand with every new CXL release. To limit complexity, we focus on two design aspects: 1) whether to provide connectivity via CXL switches or through CXL multi-headed devices (MHDs) [5, §2.5] and 2) how large the constructed pool should be to maximize return-on-investment (ROI). We discuss a particular set of choices suitable for general-purpose cloud computing. Other use cases may see different sets of choices and tradeoffs.

4.1. Components

CXL memory controller (MC) devices act as a bridge between the CXL protocol and memory devices such as DDR5 DRAMs. Today's MCs typically bridge between 1-2 CXL ×8 ports and 1-2 80b channels of DDR5 (e.g., [13]).

CXL switches behave similar to other network switches in that they forward requests and data, without serving as an endpoint. Physically, CXL switches will likely share many characteristics (e.g., port count) with PCIe switches, due to using the same physical interface. For the purposes of this analysis, we assume that switches with 128-lanes (16-ports) of CXL are used to build a fabric layer.

A CXL MHD essentially combines a switch and a memory controller in a single device. Specifically, the MHD offers multiple CXL ports

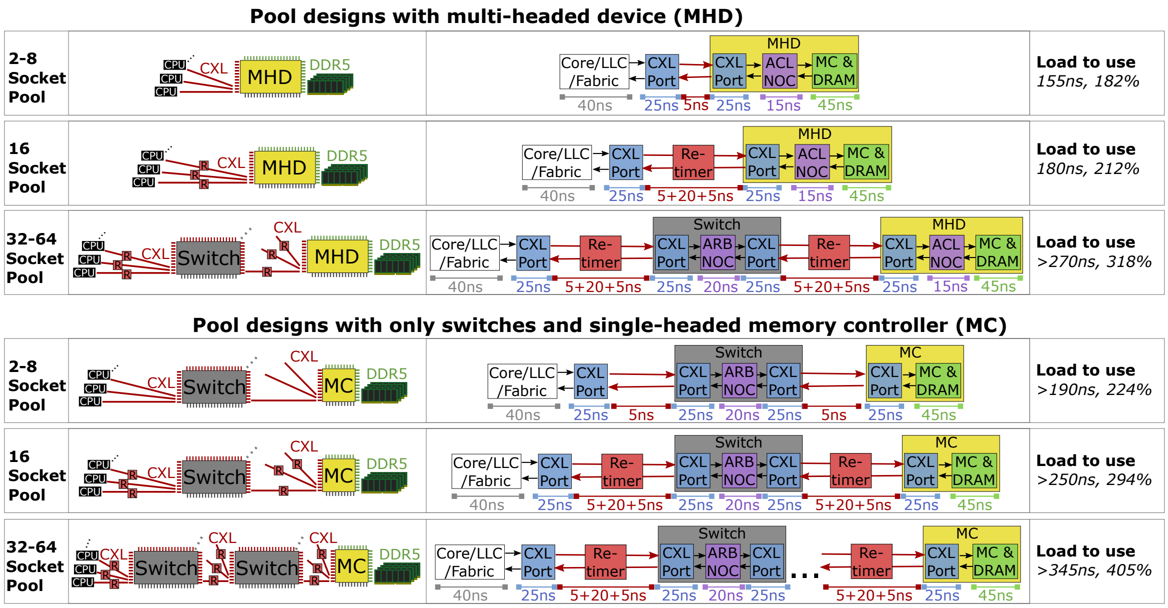

Figure 3: **Pool size and latency tradeoffs. Small pools of 8-16 sockets add only 75-90ns relative to NUMA-local DRAM. Latency increases for larger pools that require retimers and a switch.**

and appears to each connected host as a single logical memory device [5]. The most significant tradeoffs for MHD designs are the number of incoming CXL ports and DDR channels. A useful design comparison is a modern server CPU IO-die (IOD), such as the one in AMD Genoa [14]. The Genoa IOD offers 128 PCIe5 lanes as well as 12 DDR5 channels. With the $\times$8-CXL requirement, this would be analogous to a 16-headed device with at least 8 channels of DDR5. In our analysis we consider both this 16-headed device as well as a smaller 8-headed device.

4.2. Pool size vs latency

At a high level, the first design decision is whether cloud compute servers can pool all of their memory. With 21-37% of workloads facing significant slowdowns on pool-only configurations (§3), we do not recommend fully disaggregating compute and memory. Servers need to retain significant amounts of local DRAM to maintain performance expectations, which will likely go beyond the scope of on-die memory. Further, achieving maximum memory bandwidth requires CPUs to populate all available local DDR channels, creating a practical minimum for local memory capacity.

Observation 1:

A significant percentage (more than 25%) of datacenter memory needs to remain local to compute servers.

To understand pool latencies, we first characterize the impact on latency of achievable topologies given viable components.

Observation 2:

When using at least a $\times$8-CXL port for each host, pool sizes beyond 16-32 hosts will require at least one level of switches if MHDs are used or two levels of switches if using only individual MCs.

Access latencies derive from multiple parameters. Port latency plays a dominant role with initial measurements indicating 25ns [11]. Retimers are devices used to maintain CXL/PCIe signal integrity over distances above roughly half a meter, depending on the implementation of the signal path. They add about 10ns of latency in each direction (e.g., [15]). Each switch will add at least 70ns of latency due to ports, arbitration, and network-on-chip (NOC).

Figure 3 shows a range of CXL path types based on pool sizes and the use of MHDs versus

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

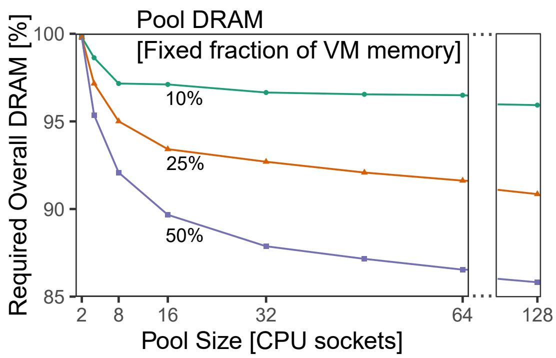

Figure 4: **Impact of pool size. Small pools of 32 sockets are sufficient to significantly reduce overall memory needs.**

switches with single-headed devices. We find that small 8-and 16-socket pools using MHDs increase latencies to 182-212% (155-180ns) relative to NUMA-local DRAM. Latency when using only switches and single-headed memory controllers would further increase by 23-38%.

Rack-scale pooling with 64 sockets would increase latencies by 318-405% (270-345ns) and pooling across multiple racks would require yet another level of switching and potentially multiple retimers, increasing latencies by more than 465% (395ns). Comparing these latencies to the slowdowns observable at 182-222% (§3), we observe that large-scale pooling will likely be prohibitive from a performance perspective.

Observation 3:
The size of CXL-based memory pools will likely be a subset of a rack to minimize the performance impact of access latencies.

Modern CPUs can connect to multiple MHDs or switches, which allows scaling to meet bandwidth and capacity goals for different clusters.

4.3. Pool size vs DRAM savings

We analyze VM-to-server traces from Azure (§3) to estimate the amount of DRAM that could be saved via pools of different sizes. The reduction in DRAM comes from averaging host's peak memory needs across the pool. Our simulation plays back VM traces while assigning a fixed percentage of pool memory. We repeatedly run cluster simulations while decreasing overall memory in 0.5% steps until the first VM is rejected. The minimum amount of cluster memory corresponds to the "required overall DRAM" reported below.

Figure 4 presents cluster DRAM requirements when VMs are assigned either 10%, 30%, or 50% of pool DRAM. As the pool size increases, the figure shows that required overall DRAM decreases. However, this effect diminishes for larger pools. For example, with a fixed 50% pool DRAM, a pool with 32 sockets saves 12% of DRAM while a pool with 64 sockets saves 13% of DRAM. Note that allocating 50% of VM memory to pool DRAM require latency mitigation techniques (§6).

Besides low latency, feasible configurations also must be ROI positive, as discussed next.

4.4. Pool size vs system cost

System cost depends on many factors. We consider a simplified model that focuses on key hardware components: DRAM, memory controllers, cables, and the memory blade enclosure/PCB. Our model ignores factors of time, scale, and market competition. Specifically, our model calculates cost relative to a non-pooled server's bill of materials (BOM) based on the following set of parameters.

MC cost of a typical 2x8 CXL memory controller (e.g., 0.4%)

MHD8 cost of an 8-headed memory controller (e.g., 0.8%)

MHD16 cost of a 16-headed memory controller (e.g., 2.0%)

Switch cost of a 16-port CXL switch (e.g., 1.6%)

Ret cost of a CXL retimer (e.g., 0.02%)

Infra cost of the supporting memory enclosure, PCBs, and cables expressed as a multiplier applied to MHD or switch cost (e.g., 0.5-2×)

The exemplary values for the parameters are roughly based on estimates of silicon area as well as connectivity and infrastructure necessary to support the memory pools. Note that there is significant room for these parameters to change between companies, server configurations, use cases, and over time.

Figure 5 presents cost overheads for pool sizes from 2-64 sockets and for pools encompassing two different capacity points relative to total system memory. The baseline for comparison is the full cost of a non-pooled server, including CPU,

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

DRAM, and other standard infrastructure (e.g. NICs, power delivery, management controllers, boards, etc.). Within this baseline, DRAM memory is assumed to account for approximately half of the total cost, with the CPU and other infrastructure splitting the other half. All other modeled configurations hold the total cost of the base system constant, but add the costs of the extra components required for pooling part of the memory. Our results are reported as a percentage of cost uplift versus the baseline configuration. We vary the infrastructure overhead cost to show that the overall costs are very sensitive to the ability for a design to cost-effectively provide connectivity to the pool. The analysis also shows that overhead for switch-based designs versus MHD designs is significant. As an example, an 8-socket pool implemented with switches adds over two-and-a-half times the cost of an 8-socket pool based on MHDs.

This overhead is important, as the system-level goal is reaching a beneficial pooling configuration, which is one where the cost uplift of moving memory into the pool is less than the efficiency benefit of having flexible memory as outlined in the savings analysis above. In Figure 5, the black line plots the savings estimate from the earlier analysis (Figure 4). Configurations below this line are ROI positive, while those above the line are likely ROI negative unless further optimizations can be made to improve savings. Note in particular that most switch-based configurations are ROI negative, while many MHD-based configurations are ROI positive, especially for smaller pool sizes.

Observation 4:
Positive ROI requires pool designers to navigate a complex tradeoff between pool size, topology, and savings, which is workload dependent. Infrastructure overheads may become a major hurdle to adopting CXL-based pooling as expensively-designed configurations will not achieve beneficial ROI.

5. Related Work

Low memory resource utilization and stranding has been observed at Google [16] and Microsoft [17]. This motivated at least three lines of prior research on memory pooling prior to CXL.

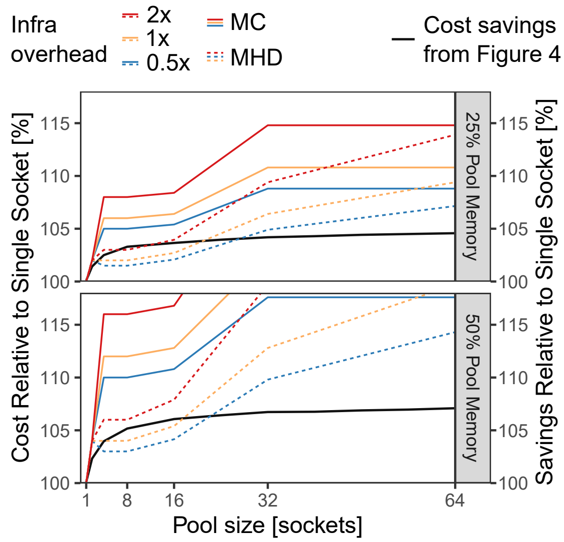

Figure 5: **Pool system cost tradeoffs. Both cost and savings increase with pool size. Infrastructure overheads also play a key factor in cost. Cost savings (black line) from Figure 4 are workload dependent and may look significantly different for other use cases. We advise practitioners to evaluate savings for their workloads.**

Hypervisor/OS level approaches such as [3] rely on page faults and access monitoring to maintain the working set in local DRAM. In the context of general-purpose cloud computing, these OS-based approaches bring too much overhead and jitter. They are also incompatible with virtualization acceleration (*e.g.*, DDA).

Runtime-based disaggregation designs [4, 18] propose customized APIs for remote memory access. While effective, this approach requires developers to explicitly use these mechanisms at the application level.

Hardware-based memory disaggregation have served as an inspiration for CXL but prior approaches were not available on commodity hardware [10, 19].

Prior analysis of requirements for disaggregation are related to our goals. However, network-based disaggregation [20] lead to a different design space, e.g., with latency considered in the range of 1us to 40us, whereas we consider latencies lower by an order of magnitude.

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

6. Discussion and Conclusion

CXL-based memory pooling promises to reduce DRAM needs for general-purpose cloud platforms. This paper outlines the design space for memory pooling and offers a framework to evaluate different proposals.

As cloud datacenters are quickly evolving, some key parameters will differ significantly even among cloud providers and over time. The fraction of VM memory that can be allocated on CXL pools depends largely on the type of latency mitigation employed. For example, the recent Pond [2] system can allocate an average of 35-44% of DRAM on CXL pools while satisfying stringent cloud performance goals. Future techniques for performance management may lead to significantly higher CXL pool usage. Another difference comes from server and infrastructure cost breakdowns, which lead to entirely different cost curves (Figure 5).

Regardless of the variability in system and cost parameters, we believe that Observations 1-4 broadly apply to general-purpose clouds. We highlight that small pools, spanning up to 16 sockets, can lead to significant DRAM savings. This requires keeping infrastructure cost overheads low, which reinforces the need for standardization of pooling infrastructure. Latency and cost increase quickly for larger pool sizes, while the efficiency benefits fall off, which may make large pools counterproductive in many scenarios.

Our savings model focuses on pooling itself, e.g., averaging peak DRAM demand across the pool, and for Azure specific workloads. CXL also enables other savings including using cheaper media behind a CXL controller, such as reusing DDR4 from decommissioned servers. We thus advise practitioners to create a savings model for their specific use cases, which might differ from ours.

CXL re-opens memory controller architecture as a research frontier. With memory controllers decoupled from CPU sockets, new controller features can be more quickly explored and deployed. Cloud providers need improved reliability, availability, and serviceability (RAS) capabilities including memory error correction, management, and isolation. Tighter integration between memory chips, modules, and controllers can enable improvements along the Pareto frontier of RAS, memory bandwidth, and latency.

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/MM.2023.3241586

REFERENCES

1. Shigeru Shiratake. Scaling and Performance Challenges of Future DRAM. In *IMW '20*.
2. Huaicheng Li, Daniel S. Berger, Stanko Novakovic, Lisa Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *ASPLOS '23*.
3. Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient Memory Disaggregation with Infiniswap. In *NSDI '17*.
4. Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. AIFM: High-Performance, Application-Integrated Far Memory. In *OSDI '20*.
5. CXL Specification. Available at https://www.computeexpresslink.org/download-the-specification, accessed December 2020, 2020.
6. Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R. K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and Anirudh Badam. LeapIO: Efficient and Portable Virtual NVMe Storage on ARM SoCs. In *ASPLOS '20*.
7. Ilya Lesokhin, Haggai Eran, Shachar Raindel, Guy Shapiro, Sagi Grimberg, Liran Liss, Muli Ben-Yehuda, Nadav Amit, and Dan Tsafrir. Page Fault Support for Network Controllers. In *ASPLOS '17*.
8. 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 *OSDI '20*.
9. Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *SOSP '17*.
10. Christian Pinto, Dimitris Syrivelis, Michele Gazzetti, Panos Koutsovasilis, Andrea Reale, Kostas Katrinis, and Peter Hofstee. ThymesisFlow: A Software-Defined, HW/SW Co-Designed Interconnect Stack for Rack-Scale Memory Disaggregation. In *MICRO-53*.
11. Debendra Das Sharma. Compute Express Link: An Open Industry-standard Interconnect Enabling Heterogenous Data-centric Computing. In *HotI29*.
12. Intel Resource Director Technology (Intel RDT). Available at https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html, accessed September 2022, 2015.
13. AsteraLabs. Leo Memory Connectivity Platform for CXL 1.1 and 2.0. Available at https://www.asteralabs.com/wp-content/uploads/2022/08/Astera_Labs_Leo_Aurora_Product_FINAL.pdf, accessed August 2022, 2022.
14. Lisa Su. AMD Unveils Workload-Tailored Innovations and Products at The Accelerated Data Center Premiere. https://www.amd.com/en/press-releases/2021-11-08-amd-unveils-workload-tailored-innovations-and-products-the-accelerated, November 2021.
15. CXL Use-cases Driving the Need For Low Latency Performance Retimers. https://www.microchip.com/en-us/about/blog/learning-center/cxl--use-cases-driving-the-need-for-low-latency-performance-reti, 2021.
16. Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the Next Generation. In *EuroSys '20*.
17. Qizhen Zhang, Philip A. Bernstein, Daniel S. Berger, and Badrish Chandramouli. Redy: Remote Dynamic Memory Cache. In *VLDB '22*.
18. Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking Software Runtimes for Disaggregated Memory. In *ASPLOS '21*.
19. Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. Clio: A Hardware-Software Co-Designed Disaggregated Memory System. In *ASPLOS '22*.
20. Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network Requirements for Resource Disaggregation. In *OSDI '16*.

Daniel S. Berger is a Senior Researcher in the Azure Systems Research group (AzSR) at Microsoft Azure. Daniel received his Ph.D. degree in computer science from TU Kaiserslautern.

Dan Ernst is a Principal Architect in the Leading Edge Architecture Pathfinding (LEAP) at Microsoft Azure. Dan received his Ph.D. degree in computer science and engineering from the University of Michigan.

Huaicheng Li is an Assistant Professor in the Computer Science department at Virginia Tech. Huaicheng received his Ph.D. degree in computer science from the University of Chicago.

Pantea Zardoshti is a Research Software Development Engineer in the AzSR group at Microsoft Azure. Pantea received her Ph.D. degree in computer science from the Lehigh University.

Monish Shah is a Senior Principal Hardware Engineer in the LEAP group at Microsoft Azure. Monish received his M.Sc. degree in electrical engineering from Stanford University.

Samir Rajadnya is a Principal Memory System Engineer in the LEAP group at Microsoft Azure. Samir received his M.Tech. degree in electrical engineering from IIT Bombay.

Scott Lee is a Principal Software Engineer Lead at Microsoft. Scott received his B.Sc in computer engineering from the University of Washington.

Lisa Hsu is a Principal Architect at Microsoft Azure. Lisa received her Ph.D. degree in computer science from the University of Michigan.

Ishwar Agarwal is a Senior Principal Engineer at Intel Corporation. Ishwar received his M.Sc. in eletrical and computer engineering from Georgia Tech.

Mark D. Hill is a Partner Architect and leads the LEAP group at Microsoft Azure. Mark received his Ph.D. degree in computer science from UC Berkeley

and served 32 years at University of Wisconsin Computer Science.

Ricardo Bianchini is a Distinguished Engineer at Microsoft Azure. Ricardo received his Ph.D. degree in computer science from University of Rochester.