Performance Evaluation on CXL-enabled Hybrid Memory Pool

Qirui Yang
Samsung
San Diego, California
qirui.y@samsung.com

Runyu Jin
Arizona State University
Tempe, Arizona
runyu.jin@asu.edu

Bridget Davis
Samsung
San Diego, California
b.davis@samsung.com

Devasena Inupakutika
Samsung
San Diego, California
devasena.i@samsung.com

Ming Zhao
Arizona State University
Tempe, Arizona
mingzhao@asu.edu

Abstract—**The emerging cache coherent Compute Express Link (CXL) interconnect provides a practical way to disaggregate cloud memory resources from monolithic servers into memory pools with DRAM-level access latency. While DRAM-only memory pool improves the resource utilization and reduces the Total Cost of Ownership (TCO) for cloud providers, we investigate the possibility of applying cheaper SSDs to a memory pooling system to further reduce the cost of cloud servers without sacrificing the application's performance. In this study, we build a simulated CXL-enabled DRAM-SSD hybrid memory pool based on Linux and commodity hardware, and conduct performance evaluation by running representative cloud workloads which cover deep learning training, database, data analytics and video processing on the testbed. The evaluation results show that a hybrid memory pool can potentially reduce memory cost while maintaining the same level of application performance for computation-intensive applications. For example, with memory overcommit ratio of 2, the performance degradation of training ResNet50 on ImageNet dataset is only 2.68%.**

Index Terms—**Memory Disaggregation, Hybrid Memory Pool, DRAM, NVMe, SSD, CXL, NUMA**

I. Introduction

Modern applications such as Deep Learning (DL) training, big data processing, along with the usage of in-memory database and key-value stores are using more and more memory to enable fast data processing that is usually beyond the memory capacity of a local machine [1]. Besides, recent studies [2], [3] show that public cloud vendors including Google and Alibaba only utilize half of the memory hardware resources. Most of the time, up to 25% of DRAM remains unrented when the processing resources are fully rented [4]. To remedy the above mentioned problems, various memory disaggregation solutions have been proposed to pool the memory resources so that memory is managed and scaled as a whole. The benefits are three-fold. First, memory disaggregation provides better scalability of the memory resources where memory can be added, removed, or upgraded easily. This can better satisfy the large memory requirements of modern applications and help to save the cost of data center management. Second, it helps improve memory utilization and reduce the hardware cost since different types of resources are disaggregated and not allocated as a whole. Finally, memory disaggregation provides failure isolation since a single memory chip failure will not affect other resources and vice versa. This also helps save the cost of maintenance [3].

However, the largest cost of public cloud still comes from the memory itself. Public providers, such as Azure, spend half the server cost on DRAM and the cost will continue to grow due to memory scaling [4]. Also, DRAM is using two to three times more power and cooling than SSDs do [5]. At the same time, NVMe SSDs are getting higher throughput while continuing be more power-efficient and cost much less, nearly one fifth the cost of DRAM [6]. To further reduce the cost from DRAM, SSDs are good candidates for disaggregated memory pools. They can be incorporated with DRAM to save even more cost. The addition of fast SSDs to the memory pool forms another tiered memory architecture in the existing two-tier memory pooling systems (local memory and remote memory) which presents new challenges to memory disaggregation in both performance and cost.

Before the emergence of the Compute Express Link (CXL) [7], Remote Direct Memory Access (RDMA) based solutions have been proposed to provide an easy-to-use, manageable and scalable memory disaggregation [3], [8]–[10]. RDMA provides high throughput and low memory access latency through kernel bypassing and zero copy. It caches virtual memory address to physical memory address to avoid frequent access to host memory for address mapping. However, RDMA has much higher memory access latency compared to local DRAM memory when the cache gets full [11]. Further, RDMA is not cache coherent which still suffers from the overhead of the traditional DMA style data transfer. Therefore, RDMA-based memory disaggregation has not been deployed in performance critical cloud environments. Besides RDMA, some other cache-coherent solutions are also developed [7], [12]–[14]. Among them, CXL provides easy-to-use native load/store instructions for memory access. Compared to RDMA, CXL avoids the DMA overhead for its cache-coherency and reduces the memory access latency to the scale of nanoseconds. CXL has become a promising solution for memory disaggregation compared to other competitors.

In this paper, we are trying to answer the following questions. First, how to simulate the behavior of the CXL-

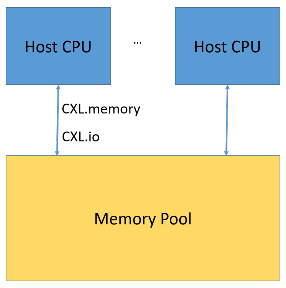
Partly supported by National Science Foundation 1955593, 2126291

based hybrid memory pool using software based solution in the absence of real hardware? Second, how does the hybrid memory pool of both DRAM and SSDs affect the applications performance compared to DRAM only memory pool in the cloud environment? Third, how does different compositions of DRAM and SSDs, or overcommit ratio, change the performance degradation? By answering the three questions, we would like to shed some light in solving the previously mentioned challenges of tiered memory disaggregation.

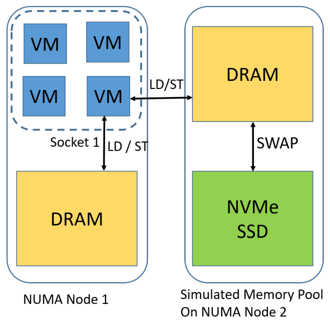
The contributions of this paper are as follows. 1) We designed a valid software simulator of CXL-based hybrid memory pool by leveraging the cache coherent NUMA architecture [15]. 2) We evaluated four representative workloads to show the performance degradation of employing hybrid memory pool compared to DRAM-only memory pool. 3) We altered the composition of DRAM and SSDs in the hybrid memory pool and present the tradeoff between performance and cost.

In the following sections, Section II describes the methodology to build the simulated CXL-enabled hybrid memory pool. Section III conducts the performance evaluation by running representation cloud workloads on the simulated memory pool. Section IV introduces related works and Section V concludes the paper.

II. Methodology



(a) CXL-enabled Memory Pool



(b) Simulated testbed

Fig. 1: Architecture Overview

A. Testbed Architecture

Figure 1a shows a common way to implement CXL-enabled memory pool hardware [4], [16]. CXL supports a variety of use cases via three protocols: CXL.io, CXL.cache, and CXL.memory. Among them, CXL.memory allows the host to access attached memory using load/store commands. The memory pool has multiple CXL.memory endpoints that can be directly connected to the host CPUs, through which the host CPUs can access the memory pool using load/store commands. However, the hardware and software management of the memory pool is done by the memory pool itself and is transparent to the host CPUs.

As there is no real CXL-enabled memory pool at this time, we simulate the memory pool by leveraging the cache coherent NUMA architecture that is widely available on today's multi-socket systems. There are two reasons to choose NUMA architecture. First, NUMA architecture is cache coherent and uses load/store commands as CXL. Second, from the host CPUs' point of view, the memory provided by the attached memory pool has similar memory access latency compared to memory from the remote NUMA nodes [7]. Even with SSDs added to the memory pool, an intelligent memory migration algorithm can help amortize the memory access latency. Figure 1b shows an example of the architecture of the simulated testbed. We are using only 2 NUMA nodes here for simplicity but in reality, the architecture can be expanded to a group of NUMA nodes.

NUMA node 1 serves as the local memory for the VMs running on socket 1's CPUs. NUMA node 2 and NVMe SSD form the tiered hybrid memory pool. In case of page allocation, pages are preferably allocated on NUMA node 1. When NUMA node 1 is under pressure, pages are then allocated in the hybrid memory pool. Inside the memory pool, most frequently used pages are cached in NUMA node 2. A page will be evicted from the NUMA node 2 to the NMVe SSD when NUMA node 2 is under pressure. Pages can migrate between NUMA node 1 and memory pool with the support of the hypervisor [17]. In this way, a tiered memory system is formed. NUMA node 1 is the first tier with the lowest memory access latency. The hybrid memory pool is the second tier with higher memory access latency. Within the memory pool, there is a sub-tiered memory system with NUMA node 2 as the first tier and NVMe SSD as the second tier.

B. Software Simulation

To control how much memory should be allocated from the memory pool to guest VMs which run the applications, we modified the Linux cgroups [18] to allow separate memory control policy for each memory tier. In the simulation, the application has different cgroups memory limits for NUMA node 1 (first tier) and hybrid memory pool (second tier). Within the hybrid memory pool, another set of cgroups memory limits is set for NUMA node 2 (first tier) and NVMe SSD (second tier) respectively. In this way, we are able to simulate the tiered memory system.

Considering the large latency gap between DRAM and SSD, we use Linux swap to simulate page migration between NUMA node 2 and the NVMe SSD as the data access latency overwhelms the page fault handling overhead. Swap happens when NUMA node 2 runs out of memory. If a guest VM accesses the pages that have been swapped out by the hypervisor, page faults caused by Extended Page Table (EPT) violation are generated. Note that using real CXL hardware can potentially achieve better application performance since in the simulation, page fault handling is removed from the guest VM but still happens on the host side.

NUMA node 1 only migrates memory pages with NUMA node 2. To ensure that no pages on NUMA node 1 are swapped out to NVMe SSD, we pin the memory pages of a VM that are allocated to NUMA node 1 in DRAM. This action prevents the pages allocated to NUMA node 1 from getting evicted. Our simulator also extends swap to provide an interface so that different prefetching and cache replacement algorithms can be easily supported in the hybrid memory pool as a separate kernel module.

III. Evaluation

We choose four workloads that cover a large portion of today's cloud application categories including video processing, database, data analytic and deep learning training. Each workload runs inside a VM provisioned by QEMU with KVM acceleration enabled on a dual socket Linux server (Table I). We run the workloads on physical machine to get the memory usage, and then set the VM memory size accordingly.

Overcommit ratio is set to help us understand the performance and cost impact by adopting NVMe SSDs and altering the composition of DRAM and SSDs in the hybrid memory pool. Equation 1 shows how the DRAM memory usage is computed with overcommit ratio. In the equation, we use *R* to represent the overcommit ratio, *actual_m* to represent the actual memory usage from DRAM, *req_m* to represent the required memory usage of the application.

$$actual\_m = \frac{req\_m}{R} \qquad (1)$$

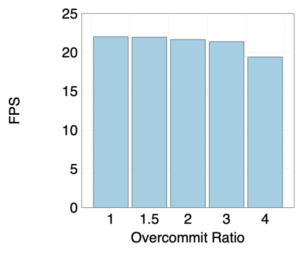
With overcommit ratio set as 1, no NVMe SSD is used and all the memory of a VM will be allocated on DRAM. When overcommit ratio is increased, the percentage of NVMe SSD used is increased. We allocate 50% of the DRAM memory space on the local NUMA node (first tier), and the rest on the remote NUMA nodes (second tier). For example, with a 32GB VM memory size and the overcommit ratio set to 2, at most 16GB of memory space will be allocated on DRAM and the rest 16GB of memory space is allocated using NVMe SSD. Of the 16GB of DRAM memory space, 8GB is allocated on local NUMA node and the rest is on remote NUMA nodes. In our evaluation, we use one local NUMA node and one remote NUMA node.

For each workload, we set the overcommit ratio to be 1, 1.5, 2, 3 and 4. In the evaluation, the video encoding VM is configured to use 4GB of memory and 2 virtual cores, while the other three workloads VMs are configured to use 64GB of memory and 32 virtual cores. To understand the baseline performance of the hybrid memory pool, we use the Linux default prefetcher for the evaluation.

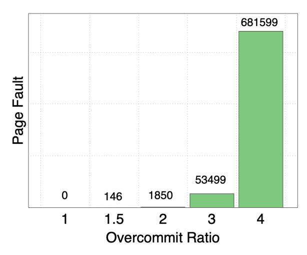
In the following evaluation, we will evaluate the performance and cost tradeoff of the four workloads using hybrid memory pool (overcommit ratio bigger than 1) compared to using DRAM-only memory pool (overcommit ratio equal to 1). Each workload runs three times and the mean values are presented.

TABLE I: Specifications of the testbed.

CPU	DRAM	SSD	GPU	QEMU	OS	Kernel
2x Intel Platinum 8268	2x 384GB DDR4	Samsung PM983 3.5TB	8x NVIDIA V100	4.2.1	Ubuntu 20.04 focal	5.15.37



(a) Frame Per Second (FPS)

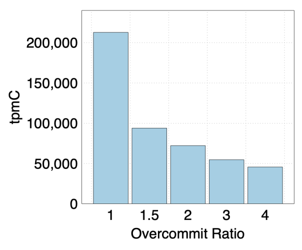


(b) Page Fault

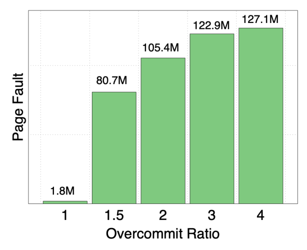
Fig. 2: FFmpeg Evaluation Results

A. H.264 Video Encoding

H.264 is a video coding format for full High Definition (FHD) video and audio. FFmpeg is a suite of programs and libraries for video encoding, decoding and transcoding. We used FFmpeg 4.2.1 to encode a 1080P 10-minute long video to H.264 format using libx264. We use Frames Per Second (FPS) to measure FFmpeg's performance. Figure 2a and Figure 2b show the FFmpeg evaluation results. From the figures, when overcommit ratio increases from 1 to 1.5, the performance degrades slightly by 0.3%. When we continue to increase the overcommit ratio, the performance degradation does not get much worse, drops by 11.9% for overcommit ratio of 4. The performance degradation is unified with the increase of the page fault. Note that although the number of page faults increases a lot for overcommit ratio of 4, the performance degradation is not much. This is because FFmpeg is computation-bounded workloads and the increase in I/Os does not become the bottleneck of the overall performance.



(a) Transaction Per Min (tpmC)



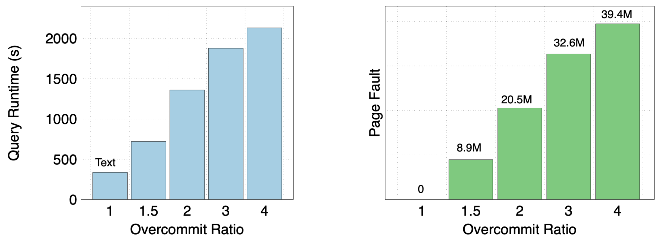
(b) Page Fault

Fig. 3: TPC-C Evaluation Results

B. TPC-C

TPC Benchmark C (TPC-C) is an on-line transaction processing (OLTP) benchmark. It is measured in transactions per minute (tpmC). We used MySQL 8.0.29 as the database. Figure 3a and Figure 3b show the TPC-C evaluation results of 1000 warehouses. When overcommit ratio increases from

1 to 1.5, the performance degrades abruptly by 55.8%. This degradation is in accordance with the abrupt increase in the number of page faults. For overcommit ratio 4, the performance degradation is the worst, drops by 78.5%.

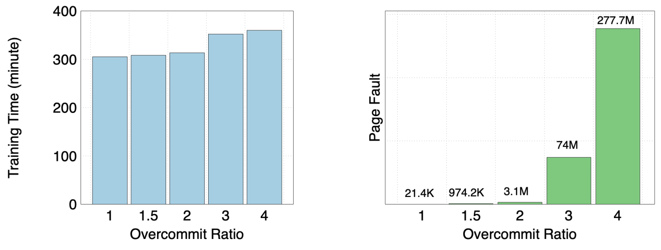


(a) Query Runtime (b) Page Fault

Fig. 4: TPC-H Evaluation Results

C. TPC-H

TPC Benchmark H (TPC-H) is a decision support benchmark. It includes a suite of 22 business ad-hoc queries and concurrent data modifications. We used Greenplum 6.20.3 [19], a Postgresql compatible Database Management System (DBMS) for data analytics, as the database deployed on a single VM. Figure 4a and Figure 4b show the TPC-H evaluation results of scale factor 30. We use the accumulated runtime of the 22 queries to measure its performance. When overcommit ratio increases from 1 to 1.5, the performance degrades a lot by 1.13X. With the increase of the overcommit ratio, the performance continues to degrade by 5.32X at last. The number of page faults increases following similar pattern.



(a) Training Time (b) Page Fault

Fig. 5: ResNet50 Evaluation Results

D. ResNet50

ImageNet is an dataset organized according to the WordNet hierarchy for image classification. We used ImageNet [20] to train ResNet50 [21] using Pytorch 1.11.0 with 8 GPUs. We used training time of 15 epochs to measure the performance. All runs use the same training seed to provide consistent results. Figure 5a and Figure 5b show the ResNet50 evaluation results. When overcommit ratio increases from 1 to 1.5, the performance degrades only by 1%. With the increase of the overcommit ratio, the performance continues to degrade by 17.9% with overcommit ratio of 4. This is related to the large increase in the number of page fault.

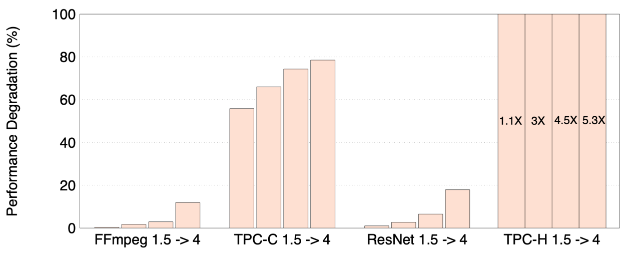


Fig. 6: Normalized Performance Degradation of Four Workloads with Varying Overcommit Ratio

E. Analysis

Figure 6 shows the performance degradation for all four workloads. From the above evaluation, we find that the hybrid memory pool involving SSDs does affect the applications performance. However, the performance degradation is largely dependent on the application. For computation-intensive workloads, like video processing and deep learning training (FFmpeg has almost 100% CPU utilization and ResNet50 has 74% GPU utilization on average), the degradation is marginal. Since the performance bottleneck depends mostly on the computation instead of the addition of memory access latency introduced by slower media. Applying SSDs to the memory pool does reduce cost while maintaining the same level of performance for these type of workloads. On the other hand, for database and analytic workloads, such as TPC-C and TPC-H, which have high requirement to memory latency and bandwidth, are severely affected by the hybrid memory pool.

F. Total Cost of Memory (TCM)

We compute the TCM using Equation 2. Based on Equation 1, we use *mem_p* to represent the unit price of DRAM and *SSD_p* to represent the unit price of NVMe SSD.

$$TCM = mem\_p * \frac{req\_m}{R} + SSD\_p * (req\_m - \frac{req\_m}{R}) \quad (2)$$

We computed the unit price of the DRAM and NVMe SSD in Table I as $4.875 [22] and $0.16 [23] per GB, respectively. Based on our configuration, Figure 7 shows the saved TCM percentage at different overcommit ratio compared to the original TCM with overcommit ratio 1. By using NVMe SSD, even with a mere addition of NVMe SSD at an overcommit ratio 1.5, we can already save 32% of TCM. What's more, we can save up to 72.5% of TCM when overcommit ratio is 4.

IV. Related Works

A. memory disaggregation

Li et al., [4] employs a Machine Learning (ML) based predictor to allocate VM memory on local and CXL-enabled memory pool to achieve tolerable performance degradation. Based on VM traces, they observed that different workloads suffer different levels of performance degradation while still 20% of applications do not experience slow down. TMO [24] dynamically offloads cold memory of applications to cheaper

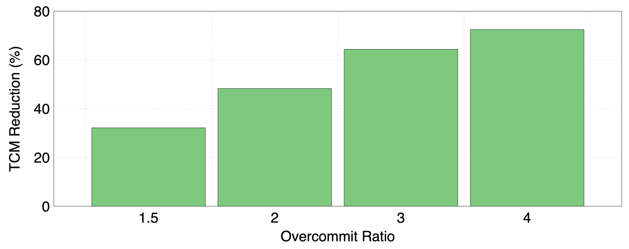


Fig. 7: Total Cost of Memory (TCM) Reduction Percentage with Varying Overcommit Ratio

memory media by monitoring the performance degradation caused by high memory access latency. However, none of the existing works has studied the performance degradation of applications using disaggregated hybrid memory pools which involves multiple tiers and multiple media.

B. CXL development

Some FPGA-based memory disaggregation have been proposed recently. DirectCXL [25] proposes a CXL-based memory disaggregation prototype which can be directly accessed by the host. A software runtime is also developed to manage the underlying CXL devices and provide accessibility to applications. ThymesisFlow [26] uses FPGA along with OpenCAPI to build a rack-scale hardware memory disaggregation prototype. In the evaluation, some applications can achieve negligible performance degradation from memory disaggregation by using the combination of local and remote memory pools. Byte-addressable SSD [27] is also advocated to provide large working memory space through high-density NVMe SSD.

V. Conclusion

In this paper, we simulate CXL-based memory disaggregation by utilizing both DRAMs and SSDs in the memory pool. We evaluate some representative workloads and show the potential of employing the hybrid memory pool to reduce memory cost for computation-intensive workloads. Given that we use the simple Linux prefetcher in the evaluation which only detects sequential access pattern, the large performance penalty from which the TPC-C and TPC-H suffer does not mean those types of workloads are prohibited on the hybrid memory pool as pages may be able to migrate between tiers in a more intelligent way. Memory disaggregation is an effective way to improve the memory utilization for cloud providers. CXL is a promising memory disaggregation solution which provides high throughput and DRAM-level latency. In order to reduce cost, memory pools can also utilize cheaper but slower medias like PMEM or NVMe SSDs.

References

[1] 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 *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[2] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, USENIX Association, Mar. 2017.

[3] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "LegoOS: A disseminated, distributed OS for hardware resource disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, (Carlsbad, CA), pp. 69–87, USENIX Association, Oct. 2018.

[4] H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, M. Shah, I. Agarwal, M. Hill, M. Fontoura, *et al.*, "First-generation memory disaggregation for cloud platforms," *arXiv preprint arXiv:2203.00241*, 2022.

[5] S. Park, Y. Kim, B. Urgaonkar, J. Lee, and E. Seo, "A comprehensive study of energy efficiency and performance of flash-based ssd," *Journal of Systems Architecture*, vol. 57, no. 4, pp. 354–365, 2011.

[6] M. Webb, "Overview of persistent memory," *Flash Memory Summit*, vol. 2018, 2018.

[7] https://www.computeexpresslink.org/, "Compute express link," 2019.

[8] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017.

[9] H. Al Maruf and M. Chowdhury, "Effectively prefetching remote memory with leap," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 843–857, 2020.

[10] A. Lagar-Cavilla, J. Ahn, S. Souhlal, N. Agarwal, R. Burny, S. Butt, J. Chang, A. Chaugule, N. Deng, J. Shahid, *et al.*, "Software-defined far memory in warehouse-scale computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 317–330, 2019.

[11] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2022, p. 417–433, 2022.

[12] http://www.nvidia.com/object/nvlink.html, "Nvlink interconnect," 2019.

[13] http://genzconsortium.org, "Gen-z specification," 2017.

[14] http://opencapi.org, "Opencapi consortium," 2017.

[15] https://github.com/njjry/cxlSim, "Simulator code," 2022.

[16] https://www.businesswire.com/news/home/20220303006046/en/Tanzanite-Silicon-Solutions-Demonstrates-IndustryNext-Generation-Composable-Data-Centers, "Gen-z specification," 2022.

[17] https://lwn.net/Articles/488709/, "Autonuma: the other approach to numa scheduling," 2012.

[18] https://www.kernel.org/doc/html/latest/admin-guide/cgroup v1/cgroups.html, "Control groups."

[19] http://genzconsortium.org, "Greenplum: Massively parallel postgres for analytics."

[20] https://www.image-net.org/, "Imagenet," 2022.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[22] https://memory.net/memory prices/, "Memory prices," 2022.

[23] u. https://diskprices.com/?locale=uscondition=new, "Nvme ssd prices," 2022.

[24] J. Weiner, N. Agarwal, D. Schatzberg, L. Yang, H. Wang, B. Sanouillet, B. Sharma, T. Heo, M. Jain, C. Tang, *et al.*, "Tmo: transparent memory offloading in datacenters," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 609–621, 2022.

[25] D. Gouk, S. Lee, M. Kwon, and M. Jung, "Direct access, High-Performance memory disaggregation with DirectCXL," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, (Carlsbad, CA), pp. 287–294, USENIX Association, July 2022.

[26] C. Pinto, D. Syrivelis, M. Gazzetti, P. Koutsovasilis, A. Reale, K. Katrinis, and H. P. Hofstee, "Thymesisflow: a software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 868–880, IEEE, 2020.

[27] M. Jung, "Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd)," in *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, HotStorage '22, (New York, NY, USA), p. 45–51, Association for Computing Machinery, 2022.