Understanding and Optimizing Serverless Workloads in CXL-Enabled Tiered Memory

Yuze Li, Shunyu Yao

Virginia Tech

Abstract

Recent Serverless workloads tend to be large-scaled/CPU-memory intensive, such as DL, graph applications, that require dynamic memory-to-compute resources provisioning.

Meanwhile, recent solutions seek to design page management strategies for multi-tiered memory systems, to efficiently run heavy workloads. Compute Express Link (CXL) is an ideal platform for serverless workloads runtime that offers a holistic memory namespace thanks to its cache coherent feature and large memory capacity. However, naively offloading Serverless applications to CXL brings substantial latencies.

In this work, we first quantify CXL impacts on various Serverless applications. Second, we argue the opportunity of provisioning DRAM and CXL in a fine-grained, application-specific manner to Serverless workloads, by creating a shim layer to identify, and naively place hot regions to DRAM, while leaving cold/warm regions to CXL. Based on the observation, we finally propose the prototype of Porter, a middleware in-between modern Serverless architecture and CXL-enabled tiered memory system, to efficiently utilize memory resources, while saving costs.

1 Introduction

Serverless computing, or Function-as-a-Service (FaaS) is a cloud abstraction, that offers highly scalable, portable, and intuitive microservice deployment for cloud applications [1, 2, 3]. It eases off developers' burden of setting up system and runtime environment, allowing them to focus on application development. Developers write their code as a set of stateless event-triggered tasks as functions, which are invoked via triggers, e.g., HTTP, gRPC. The providers spawn and tear down instances for each function on demand.

Recent trends show Serverless workloads tend to occupy large memory resources, such as high-performance computing (HPC) [4], machine learning (ML) [5], and graph computation [6], which led memory become a
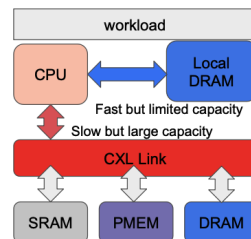

Figure 1: CXL decouples memory from compute.

significant infrastructure expense in hyperscale datacenters [7]. For example, running a DNN training workload may produce more than 40GB of memory footprint that current Serverless providers fail to offer. Meanwhile, different large-scale applications exhibit different ratio of compute-to-memory ratio that requires efficient resource provisioning. DL workloads often comprise numerous dynamic stages each with its own unique resource requirements [8], i.e., feature extraction, model training, and inference, which are served better by provisioning proportional resources to them. However, in extant Serverless datacenters, a monolithic server is statically equipped with a fixed set of compute (e.g., CPUs, GPUs), and memory (e.g., DRAM, NVM) resources. Even worse, current Serverless providers tend to provision a fixed ratio of compute-to-memory resources. There is a fundamental mismatch between the static capabilities of the servers and the dynamic needs of the emerging applications.

Compute Express Link (CXL) is a new memory disaggregation protocol that allows a new memory bus interface to attach memory to the CPU (Figure 1). Such CXL-memory acts as a CPU-less NUMA node that offers a holistic namespace of large memory size for direct *load* and *store* from host CPU [9]. Together with local DRAM, such memory system brings opportunities for designing a tiered-memory solution for fined-grained memory resource provisioning.

In fact, several state-of-art solutions [7, 10, 11, 12] address efficient provisioning of multi-tiered memory (including CXL) resources to workloads in production. However, they all focus on general workloads in a

coarse-grained manner that contains no prior knowledge of workload behavior. However, unlike black-box serverful VMs, the resource monitoring of Serverless functions can be characterized by cloud providers since the function's performance is visible. Major serverless providers, including AWS Lambda [13], already offer tools to monitor and characterize serverless functions. Unfortunately, the area of effectively offloading Serverless workloads to CXL-enabled multi-tiered memory system is left unexplored, and Serverless platforms remain largely unaware of the fundamental differences between tiered memory and DRAM-only datacenters.

In this work, we first quantify the impact of CXL on several Serverless workloads by naively making all *load*, *store* operations via emulated CXL environment. To showcase the opportunity of provisioning DRAM-CXL dual-tiered memory system to Serverless workloads, we then built a shim layer that uses *syscall_intercept* library and a page access monitor to generate hints for naive object placement. Current experiment shows such naive allocation of (potential) hot objects to DRAM, while leaving cold objects to CXL can bring down the execution time from 30% (pure CXL) to under 5%. Finally, we propose the proto design of *Porter*, a user-space middleware between modern multi-tenant Serverless architecture and underlying CXL-enabled memory systems to efficiently exploit large capacity from CXL, without harming Serverless function SLO. Porter works by 1) generating and updating memory objects placement hints by gathering workload characteristics, and 2) intelligent promotion/demotion during function runtime based on SLO requirements and system resource loads.

2 Background

2.1 Memory Provisioning in Serverless

Memory provisioning has been a major pain point for the Serverless world. There are two key insights to support this.

Rigid CPU-DRAM Ratio Provisioning The memory growing speed has been stagnating [14, 15], the CPU-DRAM ratio in VM instances is relatively dropping [16, 17]. This is very likely the reason why Serverless providers come up with memory-oriented resource provisioning strategy/pricing. Despite minor discrepancies amount different Serverless providers, they generally require users to provide three major components to provision resources: 1) function code; 2) memory cap size; 3) timeout. Then the CPU cycles resources are proportionally provisioned with the memory cap size configured by users, and the pricing of a Lambda invocation is proportional to the memory allocated [18]. However, such static CPU-memory ratio assumes all applications need

a similar ratio of different kinds of resources, which is often an incorrect assumption. Memory-intensive workloads usually require dynamic compute-to-memory provisioning. For example, DL workloads often comprise numerous dynamic stages each with its own unique resource requirements.

Insufficient Memory Resource Serverless applications have become increasingly memory-consuming, particularly graph applications [6], ML training [5], and referencing [19, 20]. The granularity and scalability of Serverless offer opportunities for data-intensive ML/DL workloads [5], but compared to the total working set of such applications, there is usually a relatively small memory capacity for each Serverless function, for instance, DL memory footprint is too large to be held in AWS Lambda (10GB) [21]. For providers, memory tiering can be a solution.

2.2 Tiered Memory

The idea of tiered memory is becoming increasingly attractive when DRAM capacity is surpassed by memory demand [10, 12]. As Non-DRAM memory technologies provide a cheaper \$/GB point [22, 23, 24, 7], tiered memory suggests bundling multiple tiers of memory spaces (e.g., DRAM, Non-Volatile Memory, Persistent Memory), with faster memory spaces to be filled out first. Such solutions often place slower-but-larger memory for colder (less frequently accessed) memory page placement, and smaller-but-faster DRAM as hotter (more frequently accessed) memory content cache.

Disaggregated memory has long been a widely applied solution to memory-consuming applications [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36] that has a large pool of cheap memory on remote nodes. Such nature makes disaggregated memory an ideal fit for a last-tier memory rank in tiered memory systems. Most recent memory disaggregation efforts [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36] are specifically designed for RDMA over Infini-Band or Ethernet networks where latency is bottlenecked by network. Recently, the advent of CXL [37] introduces opportunities for a faster-than-conventional last-tier disaggregated memory. CXL allows a new memory bus interface to attach memory to the CPU, which has a latency of around 70ns introduced by the CXL port and controller, compared to local DRAM access [9]. However, this is still order-of-magnitude faster than going through the network.

In recent years, there have been different software-level designs proposed to fully utilize the potential of tiered memory systems. Software-Defined Far Memory [11] proposes to make use of fragmented available memory on the host, regarding it as a software-defined dynamic memory region called *far memory*. It utilizes

zswap [38], a Linux kernel feature that compresses RAM for swapped pages, and compresses cold pages of user applications to be put into the far memory. TPP [7] proposes page promotion/demotion of hot/cold page placement for CXL-based tiered memory. It leverages the cache-line *load*/*store* semantics instead of passively relying on swapping. However, there lacks a viable approach to bridge the gap between Serverless and tiered memory datacenters, as all these works passively migrate pages based on page-level runtime characteristics. None of the existing solutions combines with Serverless workloads behavior to be tier-memory-aware, let alone CXL-enabled memory.

2.3 Overview of CXL Impact on Serverless Workloads

To understand how much CXL environment impacts real-life Serverless workloads, in this subsection, we showcase the experiment setup and our emulated results.

Experiment Setup We use OpenFaaS [39] as our Serverless platform, and our system speculation is shown in table 1. CXL is still not publicly available for broad deployment. Meanwhile, as the CXL access latency is similar to the remote latency on a dual-socket system [7, 9], we emulate the CXL environment by cross-accessing a CPU-less NUMA node, compared to the base (ideal) environment where all memory traffics go to local DRAM. For Serverless workloads, we derived real-world benchmarks from SeBS [40], FunctionBench [41], vSwarm [42], and GAPBS [43] and ported them to OpenFaaS. We use VTune [44], DAMON [45], and Intel Performance Counter Monitor [46] to gather metrics.

Table 1: System Hardware Specifications

Hardware	Specification
CPU	Intel(R) Xeon Gold 6126 CPU @ 2.60GHz
Cores	2 * 24 cores
L3 cache	19.25 MB
Memory	192 GB DDR4 @ 2133 MHz
Storage	240 GB SATA SSD

As seen in figure 2, the x-axis shows the sorted percent of execution time slowdown of the tested workloads, compared to all local DRAM. We observe different Serverless applications have different percentages of slowdown in CXL compared to local DRAM, ranging from 1% to 44%. Naively offloading Serverless tasks to CXL largely harms the performance. In fact, it is presumable that those with heavier *load*/*store* operations are more severely impacted (e.g., graph workloads, linear equation solving, DL training). This trend roughly matches the memory backend boundness (blue
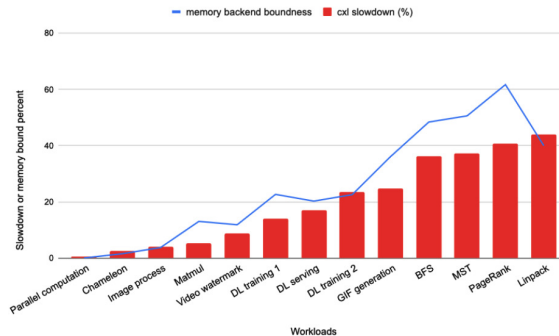

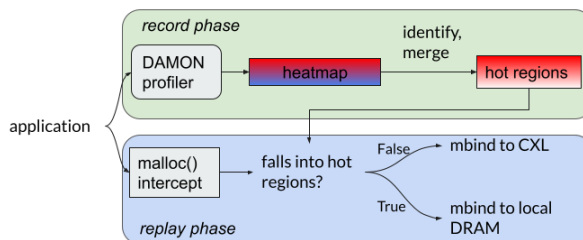Figure 2: CXL has various latency impact to Serverless workloads.

Figure 3: Profiling Memory Objects and Statically Placing.

line), which indicates how much percent of the total time is stalled due to memory traffic such as *load*, *store*, memory bandwidth boundness, and latency boundness.

3 Offline Profiling and Static Placement

Following up on previous work [47], we deployed a proof-of-concept tool to monitor workloads memory accesses and statically place different objects atop DRAM or CXL (figure 3). Specifically, having the observation of hot regions throughout workloads running and memory allocation statistics, by statically placing hot memory objects to DRAM and leaving cold/warm objects to CXL, we aim to observe how much performance gain compared to pure CXL.

3.1 Recording Memory Accesses

In the record phase, we use DAMON [45] to profile the data access patterns throughout workloads running. DAMON is a profiling tool for data access tracing with controllable overhead. It consists of a region-based sampling and an adaptive region adjustment, allowing users to limit the tracing overhead in a bounded range regardless of the size and complexity of the target workloads.

Note that for consistent memory object location, we disabled the *randomize_va_space*. We then use DAMO, a userspace tool of DAMON to generate heatmaps. As shown in figure 4, workloads show varied data access patterns. For example, strong locality (only a specific range of data gets accessed across running) can be observed for DL (ImageNet) training, linear equation solving (Linpack), and graph computation (BFS and PageRank), whereas HTML generation (Chameleon), and image processing show sparse, unpredictable access pattern. After that, we perform an offline processing to filter, merge, and generate huge chunk of hot blocks.

3.2 Tracking Memory Allocations

In the reply phase, we implement a shim layer to intercept dynamic allocations performed by applications. In situations where an application requires memory blocks that exceed a certain threshold (such as the *MMAP_THRESHOLD* value of 128 kilobytes in Linux), the *malloc* function will utilize *mmap* to allocate space within the Memory Mapping Segment region, rather than expanding the Heap through *brk*. Therefore, given a workload executable, we intercept *mmap* and *brk* syscalls by using a shared library *syscall_intercept* [48]. When intercepting each allocation, we gather information such as the timestamp, allocation size, starting memory address, and call stack.

Since for each *mmap* intercept there is a memory address range and each sample has a memory address associated with it (memory objects), we can combine with the profiled hot regions observed over time (3.1) to get placement hints 3. We statically place these objects on either DRAM for hot objects, or to CXL for cold/warm objects based on the hints. The shim layer is hooked dynamically during runtime thus no changes are required for workload source code. For proof-of-concept purposes, we note that the object is naively placed instantly after *mmap*, thus no migrations are needed in this stage. For fine-tuned promotion/demotion from/to DRAM/CXL during execution, we list in our future work in 4.2.

3.3 Static Placement Result

To show how statically placing hot memory objects to DRAM improves performance, we tested on two graph workloads, namely BFS and PageRank on Twitter dataset, as they show a large percentage of slowdown in pure CXL environment. Figure 5 shows a noticeable performance improvement compared to CXL. For PageRank workload, we observed up to 26% execution time reduction compared to pure CXL. The result can be interpreted as more memory accesses taking place on lo-

cal DRAM. This observation brings us the opportunity to identify and place memory objects that are likely to be hot/cold to DRAM/CXL right before get accessed, also workload-specific fine-grained optimizations can be offloaded.
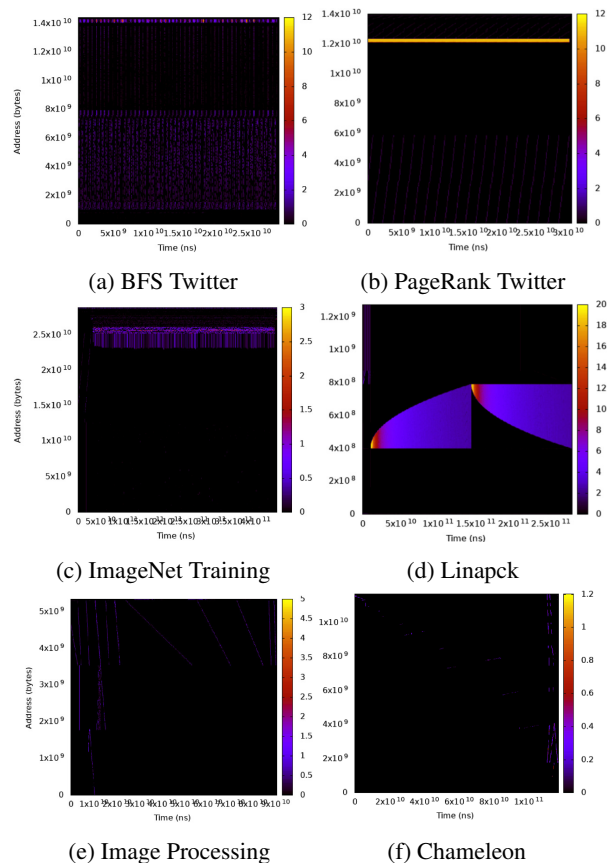

Figure 4: Heatmap of some workloads, where colored areas are denoted as hot regions.

4 Porter and Future Work

4.1 Porter Design

Having the understanding of workloads memory access behavior, we aim to deploy Serverless atop such CXL-enabled tiered memory system, to achieve efficient memory usage, cost alleviation, without harming applications' performance, and SLO-guaranteed application performance.

Our proto-design is shown in figure 6: When a user invokes a function via gateway ①, the load balancer (e.g., Kubernetes) route the request to a server. The invocation payloads with function ID are pushed into a local queue ②, which are fetched by an engine asynchronously. If it is the first time invoking (newly de-
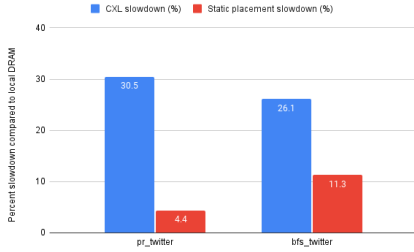
Figure 5: Performance improvement of static placement over pure CXL for PageRank and BFS on Twitter dataset.

ployed/updated function), and because the platform has no clue of the workload behavior, Porter engine is likely to provision local DRAM ③ for the best SLO guarantee, but it also depends on current system loads (memory footprint, bandwidth, etc) ⑥. While serving a request, Porter attaches a hook to intercept syscalls such as *mmap* and *munmap*, and generates heatmaps for memory access patterns. Porter also monitors workloads' back-end boundness by VTune profiler [44] to understand general behaviors. All metrics are sent to an offline tuner ④. Combined with user-defined function speculation (e.g., memory requirement, SLO), the offline tuner generates a placement hint for each function ⑤. The placement hint consists only metadata that can be cached on each server. For subsequent invocations, Porter engine gathers the function-specific hint, together with current system loads ⑥, to decide page placement for memory objects. During each function execution, the engine will spawn a separate thread to dynamically migrate pages among DRAM and CXL ⑦.

Similar to state-of-art Serverless runtime resource regulation solution [49], whenever a function memory access behavior changes, e.g., input size changes, that causes previous hot regions shifting, Porter engine would dynamically predict which memory objects would still be acting as hot/cold. If unpredictable, then it considers using DRAM to ensure the best performance, then captures and updates function metrics to the offline tuner.

4.2 Future Work

Fined-grained data access awareness For our naive placement strategy, two main problems need to be tackled in Porter. First, consider a memory object, *not all* pages (addresses) are hot. Simply regarding the object as a whole for placement causes waste of memory resources. Thus, Porter will consider a fine-grained awareness of hot objects' size. Second, not all hot pages are accessed from time to time throughout the function running. An intelligent page migration (i.e., promotion and demotion) strategy must be considered. For example,
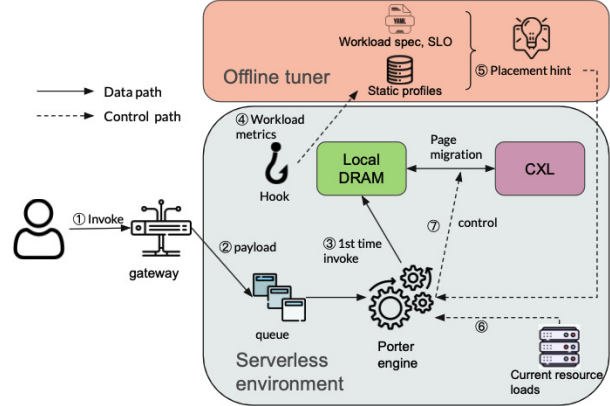

Figure 6: Porter prototype for Serverless Runtime in CXL-Enabled tiered memory system. Placement hints will be generated/updated for each invocation. Subsequent invocations benefit from the hint to efficiently utilize DRAM-CXL memory stacks.

consider ImageNet training (figure 4c), the hot regions around relative address 2.5×10^{10} are sparsely accessed chronologically. Therefore, deciding the time and data range to migrate with acceptable overhead is essential for Porter.

Resistance to payload changing One thing we noticed is even with *randomize_va_space* disabled, when the payload changes (i.e., batch size, input dataset), the underlying compiler will allocate different memory areas to the same object, invalidating the hint for data placement for subsequent invocations. Thus, Porter must be aware of how payload changes affect memory address allocation by the compiler, and still be able to successfully classify and place objects accordingly.

Function runtime impact We note that Serverless function runtime has impact to memory access behavior and overall performance. For example, performing BFS using GAPBS [43] in C++ and igraph [50] in Python on the same graph show different memory access patterns. This is because Python uses its interpreter to compile and run the program without storing the machine code being created. After comparing the call stacks, we note Python always contains extra layers than C++. Another example is when performing matrix multiplication using Python Numpy [51] versus Golang Gonum [52] package in CXL environment, we observed Python always outperforms Golang. We found Numpy uses OpenBLAS [53] as an optimized BLAS (Basic Linear Algebra Subprograms) library that allocate anonymous pages into local DRAM regardless, thus helps accelerating computation. As for Serverless clients are not aware of the underlying library behavior, same functionality in different runtime may result in various behavior, which Porter needs to consider.
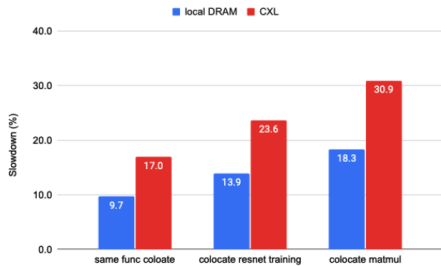
Multi-tenancy resource contention To prove the im-

5

Figure 7: Percent of slowdown in local DRAM and CXL for different colocated functions. CXL always shows more severe impact compared to local DRAM.

pact of Serverless function colocation in CXL, we run a DL serving workload and colocate with the same workload, DL training, and matrix multiplication, respectively. We capture the percent of slowdown compared to running standalone. As seen in figure 7, colocating in CXL always shows more impact on slowdown compared to local DRAM. Previous resource regulations [49, 54] in Serverless only focus on DRAM. We believe different methodologies should be considered in Porter for CXL-enabled tiered memory systems.

5 Conclusion

In this report, we first state the trend of provisioning a CXL-enabled tiered memory system to efficiently run Serverless workloads in a application-behavior-awared manner. We then demonstrate that CXL latency introduces different levels of slowdown to Serverless workloads. By naively identifying and placing hot objects to fast DRAM and leaving cold/warm ones to CXL, we observe significant performance gain. Based on that, we propose the prototype and future work of Porter, a middleware that sits between Serverless architecture and CXL-enabled memory system to effectively provision proper memory resource/type to Serverless functions.

References

[1] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In *USENIX Annual Technical Conference*, pages 475–488, 2019.

[2] Anupama Mampage, Shanika Karunasekera, and Rajkumar Buyya. A holistic view on resource management in serverless computing environments: Taxonomy and future directions. *ACM Computing Surveys (CSUR)*, 54(11s):1–36, 2022.

[3] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

[4] Bartłomiej Przybylski, Maciej Pawlik, Paweł Żuk, Bartłomiej Lagosz, Maciej Malawski, and Krzysztof Rzadca. Using unused: non-invasive dynamic faas infrastructure with hpc-whisk. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.

[5] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. Towards demystifying serverless machine learning training. In *Proceedings of the 2021 International Conference on Management of Data*, pages 857–871, 2021.

[6] AWS. What is amazon neptune? - amazon neptune, 2017.

[7] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. Tpp: Transparent page placement for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 742–755, 2023.

[8] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629. IEEE, 2018.

[9] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Vancouver, BC Canada, March 2023.

[10] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. Tmo: Transparent memory offloading in datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 609–621, New York, NY, USA, 2022. Association for Computing Machinery.

[11] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid 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*, pages 317–330, 2019.

[12] Jonghyeon Kim, Wonkyo Choe, and Jeongseob Ahn. Exploring the design space of page management for Multi-Tiered memory systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 715–728. USENIX Association, July 2021.

[13] Amazon lambda. https://aws.amazon.com/lambda, 2022. Accessed: April 9, 2022.

[14] Seok-Hee Lee. Technology scaling challenges and opportunities of memory devices. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 1.1.1–1.1.8, 2016.

[15] Chris A. Mack. Fifty years of moore's law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2):202–207, 2011.

[16] SeongJae Park, Madhuparna Bhowmik, and Alexandru Uta. Daos: Data access-aware operating system. In *Proceedings of*

the 31st International Symposium on High-Performance Parallel and Distributed Computing, HPDC '22, page 4–15, New York, NY, USA, 2022. Association for Computing Machinery.

[17] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[18] Aws lambda pricing. https://aws.amazon.com/lambda/pricing, 2023. Accessed: May 6, 2023.

[19] Facebook and amazon are causing a memory shortage. https://www.networkworld.com/article/3247775/facebook-and-amazon-are-causing-a-memory-shortage.html, 2018. Accessed: May 8, 2023.

[20] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. The case for ramclouds: Scalable high-performance storage entirely in dram. 43(4):92–105, jan 2010.

[21] Memory and computing power - aws lambda. https://docs.aws.amazon.com/lambda/latest/operatorguide/computing-power.html, 2023. Accessed: May 8, 2023.

[22] Hiwot Tadese Kassa, Jason Akers, Mrinmoy Ghosh, Zhichao Cao, Vaibhav Gogte, and Ronald Dreslinski. Improving performance of flash based Key-Value stores using storage class memory as a volatile memory extension. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 821–837. USENIX Association, July 2021.

[23] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, and Sachin Katti. Reducing dram footprint with nvm in facebook. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[24] Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. Data tiering in heterogeneous memory systems. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[25] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novaković, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote regions: a simple abstraction for remote memory. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 775–787, Boston, MA, July 2018. USENIX Association.

[26] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can far memory improve job throughput? In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[27] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 79–92, New York, NY, USA, 2021. Association for Computing Machinery.

[28] Bülent Abali, Richard J. Eickemeyer, Hubertus Franke, Chung-Sheng Li, and Marc Taubenblatt. Disaggregated and optically interconnected memory: when will it be cost effective? *CoRR*, abs/1503.01416, 2015.

[29] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 519–533. USENIX Association, April 2021.

[30] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, Boston, MA, March 2017. USENIX Association.

[31] Youngmoon Lee, Hassan Al Maruf, Mosharaf Chowdhury, and Kang G. Shin. Mitigating the performance-efficiency tradeoff in resilient memory disaggregation. *CoRR*, abs/1910.09727, 2019.

[32] Hasan Al Maruf and Mosharaf Chowdhury. Effectively prefetching remote memory with leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 843–857. USENIX Association, July 2020.

[33] Hasan Al Maruf, Yuhong Zhong, Hongyi Wang, Mosharaf Chowdhury, Asaf Cidon, and Carl A. Waldspurger. Memtrade: A disaggregated-memory marketplace for public clouds. *CoRR*, abs/2108.06893, 2021.

[34] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. AIFM: High-Performance, Application-Integrated far memory. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 315–332. USENIX Association, November 2020.

[35] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Semeru: A Memory-Disaggregated managed runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 261–280. USENIX Association, November 2020.

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

[37] Debendra Das Sharma. Compute express link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 5–12. IEEE, 2022.

[38] Linux. zswap — the linux kernel documentation, 2019.

[39] OpenFaaS Ltd. Openfaas, 2021.

[40] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. Sebs: A serverless benchmark suite for function-as-a-service computing. In *Proceedings of the 22nd International Middleware Conference*, Middleware '21, page 64–78, New York, NY, USA, 2021. Association for Computing Machinery.

[41] Jeongchul Kim and Kyungyong Lee. Functionbench: A suite of workloads for serverless cloud function service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 502–504. IEEE, 2019.

[42] Dmitrii Ustiugov, Theodor Amariucai, and Boris Grot. Analyzing tail latency in serverless clouds with stellar. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pages 51–62. IEEE, 2021.

[43] Scott Beamer, Krste Asanović, and David Patterson. The gap benchmark suite. *arXiv preprint arXiv:1508.03619*, 2015.

[44] Hassan Shojania. Hardware-based performance monitoring with vtune performance analyzer under linux. *Tech. Rep.*, 2008.

[45] SeongJae Park, Yunjae Lee, and Heon Y Yeom. Profiling dynamic data access patterns with controlled overhead and quality. In *Proceedings of the 20th International Middleware Conference Industrial Track*, pages 1–7, 2019.

[46] Thomas Willhalm and Roman Dementiev. Intel® performance counter monitor - a better way to measure cpu..., Nov 2022.

[47] Diego Moura, Daniel Mossé, and Vinicius Petrucci. Performance characterization of autonuma memory tiering on graph analytics. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*, pages 171–184. IEEE, 2022.

[48] A. Rudoff. syscall_intercept, 2017.

[49] Amoghavarsha Suresh and Anshul Gandhi. Servermore: Opportunistic execution of serverless functions in the cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 570–584, 2021.

[50] The igraph community. python-igraph. https://github.com/igraph/python-igraph/, 2021. Accessed: May 8, 2023.

[51] Charles R. Harris, Travis E. Oliphant, and Jarrod Millman. NumPy: A Website for Scientific Computing. https://github.com/numpy/numpy, 2021. GitHub repository.

[52] The Gonum Authors. Gonum: A set of numeric libraries for the Go language. https://github.com/gonum/gonum, 2021. Accessed: 2023-05-08.

[53] Xianyi Xie and contributors. Openblas. GitHub repository, 2021. Version 0.3.18.

[54] Amoghavarsha Suresh, Gagan Somashekar, Anandh Varadarajan, Veerendra Ramesh Kakarla, Hima Upadhyay, and Anshul Gandhi. Ensure: Efficient scheduling and autonomous resource management in serverless environments. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 1–10. IEEE, 2020.