Make It Real: An End-to-End Implementation of A Physically Disaggregated Data Center

Yiying Zhang
University of California San Diego

Abstract

Resource disaggregation is an approach to separate different hardware resources into independent pools in a data center, so that these pools can be easily managed and their resources can be allocated in a tight but unbounded way. The past decade has seen research and practices in realizing the resource-disaggregation idea on regular servers.

We advocate for a physically disaggregated data center, where disaggregated resource pools consist of hardware devices, not servers. Physical disaggregation could unlock another level of benefits in resource disaggregation, including further improved cost saving, easier maintenance and scaling, and more customization. This paper presents our efforts in building an end-to-end physically disaggregated data center, including the design and implementation of disaggregated hardware devices, networking systems for connecting these devices, operating systems for orchestrating them, and porting of traditional and cloud-computing applications to this physically disaggregated platform.

1 Introduction

For decades, the unit of deployment, operation, and failure in datacenters has been a monolithic server, one that contains all the hardware resources that are needed to run a user program (typically a processor, some main memory, and a disk or an SSD). This long-standing server-centric architecture has a key limitation: all types of hardware resources that a job requires need to be allocated from the same (monolithic) server, which could result in *resource stranding*, a situation where a server has some free resource (*e.g.*, a CPU core) but cannot use it for any jobs. This is because the server does not have enough resource of another type (*e.g.*, 1 GB of memory) for the jobs. Stranded resources can amount to 25% when the load is high in today's cloud VM instances [28]. Datacenters like Google [18] and Alibaba [5] also exhibit significant resource waste: about half CPU and memory remain unused.

Motivated by this limitation, the past decade has seen the rise of a new data-center architecture: resource disaggregation, where different computing resources are organized as different resource pools that can be utilized and managed independently. Disaggregation research so far has mainly taken a server-based approach by using regular servers as disaggregated resources, either directly or via emulation [6, 15, 20, 35, 40, 42, 49]. Many of these server-based disaggregation systems use a server just for one type of resource and could largely waste other resources (as in Figure 1(a)). Other server-based disaggregation systems construct virtual resource pools by allowing a server to be part of multiple pools for its different types of resources (Figure 1(b)). The main problem with this approach is the difficulty to isolate performance and management of different resources in a single server. Similarly, it is hard to grow/shrink one type of resources without also growing/shrinking other resources.

We advocate for a truly disaggregated architecture where hardware resources are decomposed from monolithic servers into pools of individual hardware devices (Figure 1(c)). This *physically disaggregated* architecture offers a set of unique advantages compare to non-physical disaggregation. First, it improves resource efficiency. Each device is dedicated for one type of resource, avoiding wastage of other types of resources as in server-based solutions. The amount of resources of a type can also be independent from other resource types, as each type forms a decoupled pool, which further improves the resource efficiency of a cluster. Second, physically disaggregated resource pools makes maintenance easier. To add new hardware resources, one only needs to attach the new hardware device to the network in the corresponding pool, while with server-based solutions, one would need to either find available slots in existing servers or buy new servers just to host the hardware resource in need. Similarly, it is easier to remove, reconfigure, or replace existing hardware resources. Finally, it is easy to customize and modify the management of hardware resources in a specific pool, as doing so would not affect the management of other pools.

Admittedly, physical disaggregation is a more disruptive approach. However, we believe that now is the time to deploy it in data centers. First, datacenters are incorporating more

heterogeneous and specialized hardware, making physical disaggregation a judicious choice for its maintenance benefits. Second, the execution unit of datacenter applications is becoming smaller, more dynamic, and more heterogeneous with trends like microservices, serverless computing, and composable AI. The fine granularity of applications fits well with the fine granularity of underlying disaggregated hardware architecture. Finally, network speed in data centers keeps increasing, with 400 Gbps Ethernet on the horizon [4]. Interconnects like CXL promises hundreds of nanosecond latency [28] and 128 GB per second bandwidth [14].

With these datacenter hardware and application trends, physical disaggregation is not only feasible but also crucial to the success of future data centers. Datacenters have successfully evolved from physical (DC-1.0) to virtual (DC-2.0). We believe that now is the time for datacenters to evolve further into a physically disaggregated one (DC-3.0).

This paper presents our journey towards building an *end-to-end physically disaggregated data center*, including an operating system that manages a physically disaggregated data center (§2), real physically disaggregated hardware devices (§3), network systems to connect these devices (§4), and programming models and compilers for porting applications to a physically disaggregated system (§5). What we present in this paper is by no means the only way of building a physically disaggregated data center. Instead, we hope that the challenges, opportunities, and solutions we present in this paper could help future researchers and practitioners in building their own physically disaggregated data centers.

2 Operating System for Disaggregation

The first problem we solve is how to manage a physically disaggregated cluster, *i.e.*, building an operating system for it. A straightforward approach is to run an OS on traditional servers and manage disaggregated devices from these servers. Doing so would require multiple network round trips between such a server and a hardware device for each OS task. Moreover, it requires additional full servers just for running OS's. Executing an OS on one disaggregated device (*e.g.*, a CPU device) could avoid this cost, but a single device does not have all the resources for running a full OS, and it still involves network round trips to other devices.

The disaggregation of hardware calls for a corresponding split of the operating system. Our proposal is a "*splitkernel*" approach. It divides traditional operating system functions into separate, loosely-coupled *monitors* that run and manage individual hardware devices, as shown in Figure 1(c). The monitors in a splitkernel can be highly customized to the type of hardware devices they manage. Each monitor operates independently to manage its own functions, only communicating with other monitors to access other resources. This communication could be done by the splitkernel OS via message passing or by hardware via a transparent coherence protocol, as we will discuss in §4. Regardless of the communication approach, a splitkernel monitor can be dynamically added, removed, or restarted without impacting the rest of the system.

With each monitor managing its own device, we only need a coarse-grained pool manager to coordinate devices within a pool. Specifically, each pool manager allocates coarse-grained resources across devices, and each device's monitor performs fine-grained allocation of its local resource. For example, a memory pool manager assigns 1 GB virtual memory chunks to difference memory devices. When an allocation is needed within a 1 GB region, the device's memory monitor performs the actual allocation of the size needed.

We prototyped the splitkernel idea with LegoOS [42], an OS designed for physically disaggregated systems. LegoOS contains a process monitor, a memory monitor, and a storage monitor for each disaggregated device, and global monitors for resource pools. LegoOS improves performance per dollar by up to 1.4× over single-machine Linux, thus demonstrating not only the feasibility but also the benefits of physical disaggregation via the splitkernel approach.

3 Disaggregated Hardware

With physical disaggregation, we need to build hardware devices that can directly attach to the network (or an interconnect) and largely operate on their own. In many cases, this involves designing and implementing OS monitors for hardware, which presents a few new challenges. First, an OS monitor needs to be implemented on a device, with limited resources that the device offers. For example, a memory device would not include general-purpose CPU cores, as it defeats the purpose of physical disaggregation. Yet, with the splitkernel architecture, one would need to implement a full virtual memory system on a memory device in hardware. Second, OS monitors should introduce minimal performance overhead. In a data-center environment, performance usually means not only throughput and average latency, but also tail latency. Furthermore, it is important that one device can serve concurrent requests from a high number of other devices in a disaggregated environment. The difficulty is that monitors only have limited resources to work with but still need to meet these performance and scalability goals. Third, even though a device does not need to be aware of how other devices are implemented, it should consider the fact that non-local resources are fetched from a slower network/interconnect. This often implies new hardware design for a device. We now present how we confront these challenges with a hardware-based disaggregated memory design and a disaggregated CPU design.

3.1 Hardware-Based Disaggregated Memory

As discussed, a disaggregated memory device should not have a full-scale general-purpose CPU but needs to execute a full

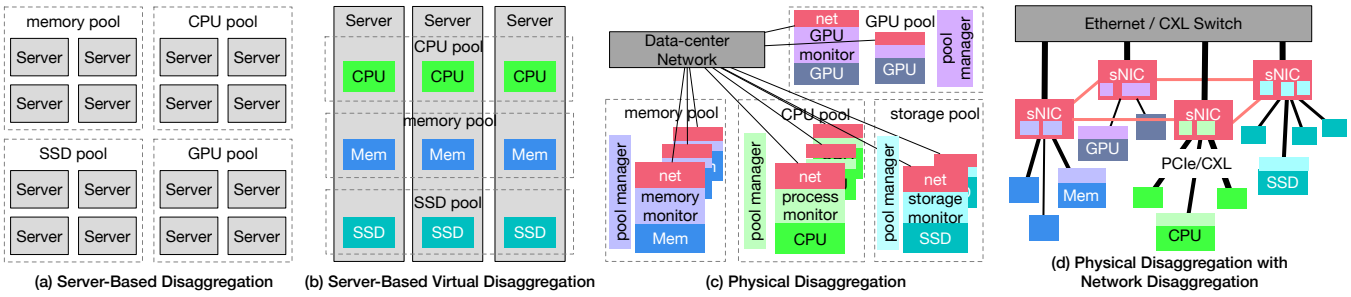

Figure 1: **Disaggregation Architectures.**

virtual memory system. We need to find the appropriate hardware to execute the virtual memory system. To answer this question, we first understand the functionalities of a virtual memory system and the complexity of each functionality. Our insight is that a virtual memory system's responsibilities include metadata tasks like virtual and physical memory allocation and data tasks like virtual-to-physical address translation and data fetching, and we should use different hardware to meet their performance and complexity needs. Our design is to use a small number of low-power ARM cores for metadata tasks and an ASIC for data tasks.

To reduce the CapEx and OpEx cost, we want to minimize the hardware resources used for these metadata and data tasks. This is especially difficult for data tasks, as we need to simultaneously achieve our performance and scalability goals. Our main idea here is to "*eliminate state*" from the ASIC hardware for the data tasks. What this means is that the hardware treats each incoming requests in isolation and does not store any metadata. To achieve these goals, we re-design the memory and networking data plane so that we can shift most state management to the ARM cores or to compute nodes. An interesting example is our implementation of a hardware-based page fault handler. Today's page fault handler lives in software (*e.g.*, Linux) and involves the allocation of physical memory, a stateful operation. We implement our page fault handler in the ASIC data plane but make page fault handling stateless by moving physical memory allocation operations to software running at the ARM cores. We further move these allocation operations off the performance-critical path by ARM cores pre-generating free physical page addresses to a shared buffer that the ASIC hardware pipeline pulls when handling page faults.

We prototyped a hardware memory device using FPGA in Clio [24]. The Clio memory device connects to a processor node that installs a Clio library. Both sides use a customized network stack on top of Ethernet. The Clio FPGA-based virtual memory system's data tasks operate at 100 Gbps throughput and 270 ns latency, both of which could be improved with real ASIC implementation. Meanwhile, each Clio node scales to concurrent accesses of thousands of process from processor nodes. These performance and scalability metrics are orders of magnitude better than our software implementation of a disaggregated memory node in LegoOS [42], while being more energy-efficient than software-based implementations.

3.2 Disaggregated CPU Processor

A disaggregated CPU processor executes application binaries on its CPU. As such, the CPU cores themselves do not need any changes. However, we need a new CPU cache architecture for two reasons. First, as with the splitkernel design, all memory management functionalities including virtual-to-physical address translation are moved to memory nodes, processors can only see virtual addresses and have to use virtual memory addresses to access its caches. Because of this, we need to organize all levels of CPU caches as virtual caches, *i.e.*, virtually-indexed and virtually-tagged caches. Second, as the network/interconnect delay from processor to (remote) memory nodes is larger than the delay to access local memory, we would need to increase the processor cache space to allow good access locality. To do so, we could either increase the size of the last-level CPU cache or add another level (L4) cache underneath it. It is also beneficial to have better cache architecture and replacement policy that can take into consideration the larger size of cache and the longer delay to remote memory. One possible way is to manage the L4 cache in software, which can then easily change the cache architecture and various cache policies. We prototyped the ideas of virtual caches and L4 cache in LegoOS' compute nodes [42] and the idea of software-managed and customized L4 cache architecture in Cocas [23]. Our results show that a well designed software-based L4 cache could largely improve the overall application performance over swap system that swaps compute-node memory pages to remote memory nodes [6, 20].

4 Network and Disaggregation

Physical disaggregation poses new challenges on the network system, as disaggregated devices are separated apart by the network. The first challenge is network speed, which is worse than local systems bus like memory bus in both bandwidth

and latency. The second challenge is the added scale of disaggregated devices: when a sever is decomposed into several disaggregated device, the total number of end points in a physically disaggregated data center could grow by several times or even more [13]. A related challenge is the cost of each end point's network system, including the network interface cost and the cost of network-packet-processing hardware.

Despite the challenges, there are at least two promising trends in data-center networking to make physical disaggregation feasible. The first is the ever-increasing speed of data-center networks. Data-center network has increased from 10 Gbps to 40 Gbps and 100 Gbps in a few years. Now, 200 Gbps network is available and 400 Gbps network is on the horizon [4]. At the same time, new fabrics like CXL [14, 28] offer extremely low-latency (10s to 100s of ns [28]) interconnect solutions. The second is the availability of computation power within network devices. SmartNICs [1, 3, 16, 34] and programmable switches [2, 8] make in-network computation feasible and attractive. Such computation power could not only be used for traditional network processing but potentially also for disaggregation monitor functionalities. Below, we discuss how to build a network for physically disaggregated devices in terms of basic connectivity and more advanced network/OS/application processing.

4.1 Network for Disaggregation

Communication media and abstraction. Today's data-center networks mainly consist of Ethernet- and RDMA-based networks. Traditional Ethernet-based data-center networks often use connection-oriented message or stream interface. Besides messaging, RDMA offers a *one-sided* remote memory access interface. Our experience is that both types of networks need modification to fit physical disaggregation.

For Ethernet-based network, a reliable, connection-oriented transport layer like TCP adds performance, scalability, and cost burden that is not always needed. This is especially true in an *asymmetric* setting like disaggregated compute and memory nodes. Disaggregated memory nodes are *passive* devices that only handles requests and not initiating any. With compute nodes as the request-initiating ones, the communication between them and memory nodes can be treated as an asymmetric one. We can leverage this property to largely simplify the network stack on a memory node, thereby reducing its building and energy cost. For example, in Clio [24], we build a connection-less RPC-like protocol between a compute node and a memory node, removing connection state and transmission buffer from the memory node. We further remove the retransmission buffer and allow out-of-order delivery of packets in a flow, as we treat one remote memory request as a flow and retry the entire request if packet drop/corruption happens. Essentially, we raise the reliability Finally, we perform congestion control and incast control purely at compute nodes, as they know both the outgoing and incoming traffic to memory nodes. As a result, a Clio memory device needs no transport, which largely reduces its cost and increases its scalability.

RDMA's main limitation lies in its primitive, limited one-sided remote memory interface that reads/write to virtual memory addresses allocated at a remote memory node. One-sided RDMA which only involves the requesting node's CPU is especially useful for physically disaggregation systems, as their memory nodes can then be built without processing power. However, applications building on top of one-sided RDMA requires multiple network round trips to accomplish a task because today's one-sided RDMA only offers primitive APIs like reading/writing a remote memory address. We extended RDMA with a richer set of primitives and an object-based addressing mechanism in LITE [47]. Other recent works also proposed richer one-sided RDMA programming abstracts [9, 38]. Disaggregation systems can leverage these extended one-sided RDMA models to coordinate across disaggregated nodes more easily and with less network communication overhead.

Topology. As discussed earlier, when a server is disaggregated into individual devices, we inevitably increase the number of end points in a data center. To handle the increased scale, there are two general approaches. The first is to solve the scale problem at the data-center level, either by increasing the scale of the topology a data center is currently using or use a new type of topology [17]. The data-center-level approach requires changes to existing data-center network infrastructure and is less likely to be adopted.

The second approach and the approach we take is to increase the scale within a rack, while maintaining the same network topology above racks and reusing all existing Top-of-Rack (ToR) switches and higher-level switches. For example, in our SuperNIC [43] work, we add an intermediate layer of multi-host NICs in between end points and the ToR switch (Figure 1(d)). Each NIC connects a small number (*e.g.*, 4 or 8) end points (disaggregated devices) and to the ToR switch, thereby increasing the rack's scale by a factor of this number (*e.g.*, 4× or 8×).

4.2 Disaggregating Network Resources

Disaggregation not only presents challenges to the network system, it also offers opportunities. Here, we take a different perspective of networking and view it as a type of computing resources. We propose to *disaggregate network functionalities* as a separate resource pool, to colocate with other resource pools. Our definition of network functionalities include three categories. The first type is traditional network stacks running at end hosts such as a transport layer. The second is network functions commonly used by data centers for network monitoring and management, such as firewall, IPSec,

and network telemetry. The third is application-specific network processing such as key-value store operations [27, 29], real-time analytics [27], and serverless/microservice functions [12, 31]. Currently, these network functionalities are often scattered across several locations, such as Network Function Virtualization at regular servers [26, 32, 36, 45, 51], SmartNICs [30, 34], host-attached FPGA [11, 37], IPU [25], middleboxes [41, 44, 48], and programmable switches [2].

Our proposal is to pool network devices at the rack scale to avoid having a NIC or other network devices at each endpoint. Disaggregating network functionalities makes them easier and more flexible to manage in a separate, dedicated pool, as adding or changing network functionalities would not involve end hosts or other parts of the data center.

We further propose to *consolidate network functionalities* by connecting multiple (*e.g.*, 4 to 8) endpoints to one programmable network device, as shown in Figure 1(d). Our motivation for network-functionality consolidation is the currently underutilized endpoint network devices. Without disaggregation and consolidation, each endpoint needs to provision all types of network functionalities and do so for its peak network traffic load. However, usually, only a small set of network functionalities are needed at a time [50], and data-center workloads have high load variations [39]. As a result, networking hardware today is usually over-provisioned, resulting in cost wastage. With consolidation, a rack-level disaggregated network pool only needs to provision the peak of *aggregated load* from all endpoints in a rack. Because different endpoints often peak at different times (from our analysis of data-center network traces [21, 39]), *the peak of sum* in a rack is much lower than *the sum of peak*, contributing to a significant cost reduction with network disaggregation.

We realize network-functionality disaggregation and consolidation by prototyping a new FPGA-based multi-host SmartNIC called SuperNIC and by building a distributed management system for a rack-level SuperNIC pool [43]. Our results show a 52% CapEx saving, 56% OpEx saving, and an overall 1.86× performance-per-dollar improvement.

5 Application Porting

Having solved the underlying system and hardware challenges, the next important step is to support applications on a disaggregated platform. Below, we demonstrate the feasibility and benefits of running traditional types of applications and cloud applications on disaggregated systems.

5.1 Porting Traditional Programs

By traditional programs, we mean regular software that run in data centers, *i.e.*, not written specifically for cloud computing. Below, we present our three approaches of porting traditional programs to physically disaggregated systems.

Operating-system approach. Our first approach of porting traditional programs is at the operating systems level, *e.g.*, by supporting Linux ABIs. LegoOS [42] uses a shim indirection layer to implement Linux system call interfaces with LegoOS' internal operations, many of which could involve multiple disaggregated devices. Although being transparent, LegoOS and other transparent systems approach (*e.g.*, those based on remote memory swapping [6, 20]) could have performance and network communication overheads due to their coarse-grained (4 KB-based) accesses to remote memory. 4 KB is often larger than what is actually read/written by a given application. A recent work found that the eight applications they study exhibit 2.3× to 31× read/write amplification [10].

New APIs. Our second approach is to add new APIs specific to disaggregation. Clio [24] uses a set of memory APIs such as remote allocation/deallocation, remote read/write, and remote memory fence, a set of distributed synchronization APIs such as remote lock and unlock, and a set of high-level interfaces such as key-value get/set, object store/retrieve. Although such explicit APIs could be used to precisely control the disaggregated memory accessing behavior, systems with new APIs add burden to programmers and rely on them to make the right decision (*e.g.*, what data to place in disaggregated memory).

Compiler-level approach. To solve these problems, we propose a third approach: *co-designing compilers and execution systems* to transparently and efficiently execute unmodified programs in a disaggregated environment. Our approach is motivated by three observations. First, disaggregated-memory systems' performance is highly dependent on how well they cache data in the local memory. Program analysis can reveal memory access patterns that a run-time system alone cannot predict. Second, dynamic run-time information is also crucial when optimizing disaggregated-memory application performance, as the actual disaggregated-memory access pattern and application behavior can be dependent on an application's input. Third, unlike traditional hardware caches that are fixed and generic, we can dynamically customize and adapt the architecture and configuration of a disaggregated-memory local cache, as it lives in the main memory and can be defined by software (*e.g.*, ExCache in LegoOS [42]).

With these observations, we build *Cocas*, a *Co-Optimized Compiler And System* platform [23]. Cocas includes a set of program analysis designed for far memory, a run-time system running at compute servers that hosts and configures local cache and other system stacks based on program analysis results, a run-time system running at far-memory nodes to manage far-memory data and offloaded computation, a compiler that optimizes user program for far-memory accesses through our run-time systems, and a monitoring system that profiles run-time application behavior. After initialization, Cocas works as a continuous loop of execution profiling and

code+system optimization. Cocas utilizes static program analysis (*e.g.*, loop and pointer analysis), profiled execution information (*e.g.*, amount of remote accesses to a data object), and system environments (*e.g.*, available network bandwidth, local memory and far memory capacity) together to optimize both the compiled code and the system configurations. Our evaluation shows that Cocas improves the performance of real applications like data analytics and machine learning by at least 2× compared to the state of the art [6, 33, 40].

5.2 Porting Serverless-Computing Services

Serverless computing is a type of cloud service that allows users to deploy and execute their applications without managing servers. Serverless computing has gained significant popularity in recent years because of its benefits to cloud users: minimal IT burdens and paying only when their applications run. We choose serverless computing to demonstrate the benefits of disaggregation to cloud computing, both because of its popularity and its natural synergy with disaggregation.

Today's serverless computing is commonly offered in the form of Function-as-a-Service (FaaS), where users write programs as functions and choose from cloud-provider-offered function sizes when deploying them. A function is a rigid, fixed-size box that couples all types of computing resources as an inseparable unit throughout a function's execution and across the function's invocations. Because of this, FaaS has a set of resource-related limitations, including per-function resource limits, fixed CPU-to-memory ratio, and constant resource allocation throughout a function execution and across different invocations of it.

We propose a "*resource-centric*" model for serverless computing that captures fine-grained resource needs in an application's execution using components of distinct resource type, amount, and time span. As these components have arbitrary resource features (*e.g.*, different amount of CPU/memory over different periods of time), scheduling them in a data center would be hard and leading to resource stranding (*i.e.*, available resources in a server that cannot be used by any applications). Resource disaggregation provides a natural solution to this challenge. By allocating only one type at a time in that type's pool, we can avoid the need to find a server that has available amount of every resource type [7, 19]. We port serverless computing by running resource-based components in a disaggregated system. To further reduce resource stranding, we propose to *"materializing" an arbitrary sized (virtual) component* of a type with multiple *fixed-size physical components* of that type. This materialization approach has the same spirit of mapping arbitrary sized memory segments to fixed size memory pages.

Based on these ideas, we built Scad [22], a system that first maps serverless computing workloads to resource-based virtual components and then materialize them at run time to physical components on a disaggregated system. Our results show that Scad reduces resource consumption by 16% to 84% and improves performance by up to 28% for a machine-learning workload and a data analytics workload.

6 Conclusion and Future Work

This paper presents the past few years of our journey towards building an end-to-end physical disaggregation systems, including new disaggregated hardware devices, a new operating system for managing disaggregated devices, a new network system for connecting these devices and for consolidating network functionalities, and new approaches for porting traditional applications and cloud applications to a disaggregated system. There are still many open research questions that we have not solved yet, as we briefly discuss below.

Reliability and security. As with traditional non-disaggregated systems, disaggregated systems too need to meet various reliability and security requirements. Disaggregation also raise new challenges for these requirements. For example, one disaggregated memory device can host data for multiple compute devices. Thus, its failure could potentially affect more applications and tenants. Disaggregated memory devices also demands proper memory protection, confidentiality, and integrity, and computation offloaded to a memory device demands proper isolation as well. Moreover, disaggregated systems could introduce new side channels (as shown in our Pythia work [46]).

CXL. A promising media for connecting disaggregated nodes is CXL [14], an emerging PCIe-based, cache-coherent interconnect. Different from traditional network connections, CXL ensures the coherence of data that is cached at different endpoints. In addition, it promises to achieve high bandwidth (*e.g.*, 128 GB/s with CXL 3.0 and x16 lanes) and low latency (*e.g.*, 155ns to 270ns [28]). Because of its coherence support and its superior performance, CXL has new implications on the abstraction and patterns of communication among disaggregated devices. In addition, since it is faster to access remote memory node with CXL, compute nodes could potentially cache less data. Appealing as it is, CXL is not fully available (*e.g.*, CXL 2.0 and 3.0 are not available yet). When it is, disaggregation systems would need to be adapted to fully utilize it.

Heterogeneity support. So far, most disaggregation works focus on CPU, memory, and storage devices, and our work also disaggregates network resources. In a data center, there can be more types of hardware resources that can benefit from disaggregation. For example, accelerators such as GPUs are precious computing resources and could benefit from tighter resource packing. Their access patterns are different from

CPU or other devices that have been explored for disaggregation. It is also important to have a solution to support new types of hardware, ideally, without designing a new set of disaggregated system every time a new hardware is introduced.

Beyond a data center. Finally, we believe that the general idea of disaggregation can go beyond a data center. It would be interesting to see how multiple data centers can incorporate the disaggregation idea at the data-center level, for example, by having each data center host a *specialized* hardware resources (*e.g.*, a GPU-specialized data-center, a CPU-centric one, etc.). Each could then build its own hardware in a better and more economic way, and together offering a better/cheaper service for users. Disaggregation could also be potentially applied to the edge environments, for example, by having different home devices work together for a computing task at a home.

References

[1] https://netfpga.org/.

[2] Intel Tofino Series Programmable Ethernet Switch. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

[3] SmartNIC Overview - Netronome. https://www.netronome.com/products/smartnic/overview/.

[4] Acton. The New World of 400 Gbps Ethernet. https://www.accton.com/Technology-Brief/the-new-world-of-400-gbps-ethernet/.

[5] Alibaba. Alibaba Production Cluster Trace Data. https://github.com/alibaba/clusterdata.

[6] 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)*.

[7] Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and Bruce Shepherd. Tight Bounds for Online Vector Bin Packing. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing (STOC '13)*, Palo Alto, California, USA, 2013.

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, jul 2014.

[9] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan R. K. Ports. Prism: Rethinking the rdma interface for distributed systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, Virtual Event, 2021.

[10] 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)*, Virtual, USA, 2021.

[11] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '16)*, Taipei, Taiwan, December 2016.

[12] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. λ-NIC: Interactive Serverless Compute on Programmable SmartNICs. http://arxiv.org/abs/1909.11958, 2019.

[13] Paolo Costa, Hitesh Ballani, Kaveh Razavi, and Ian Kash. R2c2: A network stack for rack-scale computers. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, London, UK, August 2015.

[14] CXL Consortium. https://www.computeexpresslink.org/.

[15] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI '14)*, Seattle, WA, USA, April 2014.

[16] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: Smartnics in the public

cloud. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI '18)*, Renton, WA, USA, April 2018.

[17] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan M. G. Wassel, Zhehua Wu, Sunghwan Yoo, Raghuraman Balasubramanian, Prashant Chandra, Michael Cutforth, Peter Cuy, David Decotigny, Rakesh Gautam, Alex Iriza, Milo M. K. Martin, Rick Roy, Zuowei Shen, Ming Tan, Ye Tang, Monica Wong-Chan, Joe Zbiciak, and Amin Vahdat. Aquila: A Unified, Low-Latency Fabric for Datacenter Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI '22)*, Renton, WA, April 2022.

[18] Google. Google Production Cluster Trace Data. https://github.com/google/cluster-data.

[19] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-Resource Packing for Cluster Schedulers. *SIGCOMM Comput. Commun. Rev.*, 44(4):455–466, aug 2014.

[20] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang Shin. Efficient Memory Disaggregation with Infiniswap. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, Boston, MA, USA, April 2017.

[21] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, 2019.

[22] Zhiyuan Guo, Zachary Blanco, Mohammad Shahrad, Zerui Wei, Bili Dong, Jinmou Li, Ishaan Pota, Harry Xu, and Yiying Zhang. Decomposing and Executing Serverless Applications as Resource Graphs, 2022. arXiv:2206.13444.

[23] Zhiyuan Guo, Zijian He, and Yiying Zhang. Co-Optimizing Compilers and Execution Systems for Far Memory. under submission.

[24] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying 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 '22)*, Lausanne, Switzerland, March 2022.

[25] Intel. Intel Infrastructure Processing Unit (Intel® IPU). https://www.intel.com/content/www/us/en/products/details/network-io/ipu.htm.

[26] Georgios P. Katsikas, Tom Barbette, Dejan Kostić, Rebecca Steinert, and Gerald Q. Maguire Jr. Metron: NFV service chains at the true speed of the underlying hardware. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[27] Antoine Kaufmann, SImon Peter, Naveen Kr. Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with flexnic. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*.

[28] 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 '23)*, Vancouver, BC Canada, March 2023.

[29] Jialin Li, Jacob Nelson, Ellis Michael, Xin Jin, and Dan R. K. Ports. Pegasus: Tolerating skewed workloads in distributed storage with in-network coherence directories. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020.

[30] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. PANIC: A high-performance programmable NIC for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[31] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Mangpo Phothilimthana. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *Proceedings of Usenix ATC*, 2019.

[32] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014.

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

[34] Mellanox. Bluefield smartnic. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf, 2018.

[35] 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, 2018.

[36] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: A framework for nfv applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015.

[37] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA '14)*, Minneapolis, MN, USA, June 2014.

[38] Waleed Reda, Marco Canini, Dejan Kostić, and Simon Peter. RDMA is turing complete, we just did not know it yet! In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, Renton, WA, April 2022.

[39] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, 2015.

[40] 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)*.

[41] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, 2012.

[42] 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)*, Carlsbad, CA, October 2018.

[43] Yizhou Shan, Will Lin, Ryan Kosta, Arvind Krishnamurthy, and Yiying Zhang. Supernic: A hardware-based, programmable, and multi-tenant smartnic, 2022. arXiv:2109.07744.

[44] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. *SIGCOMM Comput. Commun. Rev.*, 2012.

[45] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. Nfp: Enabling network function parallelism in nfv. SIGCOMM '17, 2017.

[46] Shin-Yeh Tsai, Mathias Payer, and Yiying Zhang. Pythia: Remote oracles for the masses. In *28th USENIX Security Symposium (USENIX Security 19)*.

[47] Shin-Yeh Tsai and Yiying Zhang. LITE Kernel RDMA Support for Datacenter Applications. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*, Shanghai, China, October 2017.

[48] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation - Volume 6*, OSDI'04, 2004.

[49] 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)*.

[50] Tao Wang, Hang Zhu, Fabian Ruffy, Xin Jin, Anirudh Sivaraman, Dan R. K. Ports, and Aurojit Panda. Multitenancy for fast and programmable networks in the cloud. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '20)*.

[51] Yang Zhang, Bilal Anwer, Vijay Gopalakrishnan, Bo Han, Joshua Reich, Aman Shaikh, and Zhi-Li Zhang. Parabox: Exploiting parallelism for virtual network functions in service chaining. In *Proceedings of the Symposium on SDN Research*, SOSR '17, 2017.