Towards a Fully Disaggregated and Programmable Data Center

Yizhou Shan
University of California, San Diego
ys@ucsd.edu

Will Lin
University of California, San Diego
w5lin@ucsd.edu

Zhiyuan Guo
University of California, San Diego
z9guo@eng.ucsd.edu

Yiying Zhang
University of California, San Diego
yiying@ucsd.edu

Abstract

Today, we are seeing two trends in the data center. On the one hand, applications are becoming more fine-grained, driven by the recent trend of serverless computing and microservices. On the other hand, data-center hardware is becoming more heterogeneous and customized to different computing needs. Because of these trends and for better manageability, several major data centers are moving towards a *disaggregated* architecture, where different hardware resources like storage and accelerators are organized as independent, network-attached pools. However, data centers today are still server-centric and relies heavily on traditional CPU-based servers.

In this paper, we take a step further and explore the possibility of building a fully disaggregated data center, where every type of resource is disaggregated. Moreover, we explore the requirements and implications of making each of the disaggregated device *programmable*. We present guidelines and initial solutions for data center designers to navigate design trade-offs. Specifically, we decompose the overarching problem into four sub-problems and propose solutions to each of them. At the top layer, we explore two types of abstractions and propose a disaggregation-native design methodology. At the bottom layer, we describe the hardware and key features required to build disaggregated devices as well as the networking infrastructure to connect them. To bridge these two layers, we propose a static-time component that compiles different user programs into heterogeneous disaggregated devices through a disaggregation-native intermediate representation. We also propose a run-time system that manages hardware resources and schedules compiler generated execution units. We hope our proposal can pave the way for future disaggregated and programmable data center deployment.

APSys '22, August 23–24, 2022, Virtual Event, Singapore
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9441-3/22/08.
https://doi.org/10.1145/3546591.3547527

CCS Concepts

• **Computer systems organization → Cloud computing**.

Keywords

Resource Disaggregation, Data-Center Hardware Architecture, Data-Center Network

ACM Reference Format:
Yizhou Shan, Will Lin, Zhiyuan Guo, and Yiying Zhang. 2022. Towards a Fully Disaggregated and Programmable Data Center. In *13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '22), August 23–24, 2022, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3546591.3547527

1 Introduction

We live at an exciting time when both software and hardware in data centers are experiencing revolutional developments. At the one hand, data center applications are becoming more fine-grained, driven by the microservices software model and the cloud serverless computing paradigm. Fine grained computation units are easier to scale and can more efficiently utilize data-center hardware resources. When coupled with a management layer that hides the complexity of deploying fine-grained application units, users can focus on their core business logic, leaving IT burdens to the provider. Because of these benefits, fine-grained computing models such as serverless computing are often expected to keep seeing wider adoption or even become "the default computing paradigm of the cloud era" [54]. Hence, we will see more existing applications or system software transitioning into smaller, DAG-based serverless counterparts.

On the other hand, the data center hardware infrastructure is becoming more heterogeneous and programmable. As CPU is meeting its limitation with the slow down of Dennard scaling and Moore's Law, accelerators like GPU, FPGA, and TPU are deployed in large scale in major data centers [4, 5, 21, 34, 50]. Unlike traditional fixed-logic hardware, many of today's

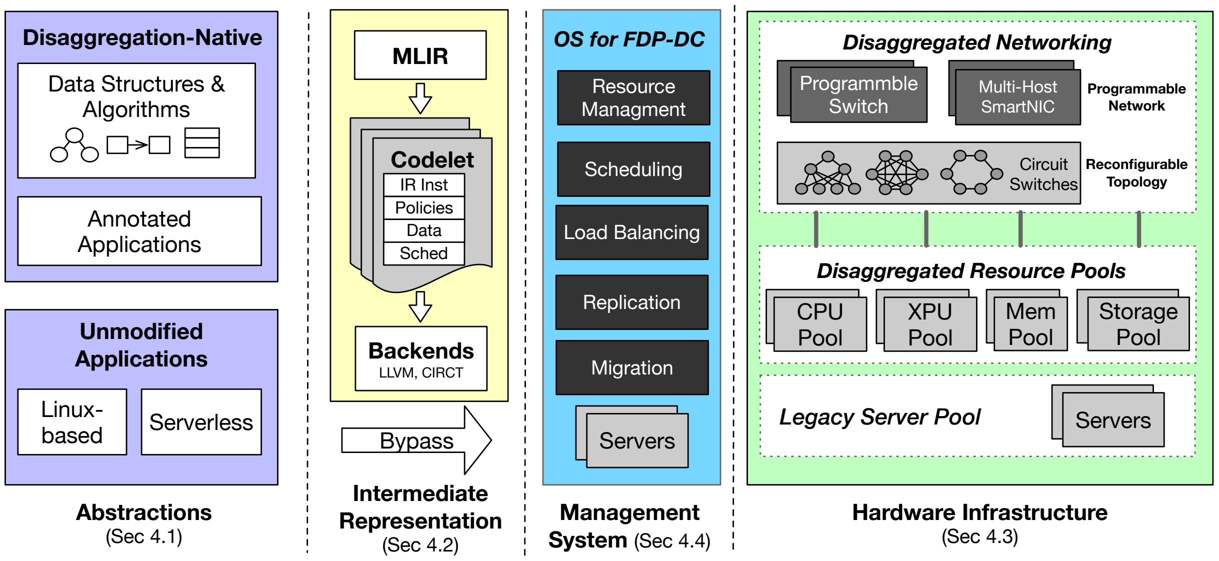

Figure 1: Overview of a Fully Disaggregated and Programmable Data Center.

accelerators are fully programmable or have a programmable part. Apart from computing resources, networking resources are also becoming more heterogeneous and programmable. Instead of traditional middleboxes and NICs, programmable switches and SmartNICs are making their ways into data centers [13, 18, 18, 19, 19, 21].

Although both software and hardware in data centers are evolving quickly, the data center architecture has largely remain the same for decades: regular servers connected with a data-center-wide network. Unfortunately, the fine granularity and heterogeneity of today's software and hardware make deployment and management hard in a server-based data center. Partly because of this, several major data centers started to embrace a new data-center architecture called *resource disaggregation*. The idea of resource disaggregation is to organize each type of hardware resource as a separate resource pool and to allow applications to utilize any resource from a pool. For example, many data centers today disaggregate storage resources from compute servers [6, 23, 45, 65]. With disaggregation, data-center owners can easily add, remove, manage, and change hardware of one type without affecting other hardware types.

Despite the success in today's disaggregation solutions, three open research questions remain. First, although certain types of hardware resources like storage have been successfully disaggregated, it is unclear how to disaggregate other types of resources like accelerators and network devices. Instead of today's point solutions for each resource type, ***what is a generic disaggregation design that could work for heterogeneous types of resources?*** Second, today's data-center network is designed for connecting servers, but ***how to efficiently connect disaggregated devices?*** Can we make use of programmability and reconfigurability in the network for a disaggregated data center? Finally, it is still unclear ***how to best map applications to a disaggregated hardware platform***. Shall we use new programming models? How to support legacy programs? Are there any optimizations we can do when mapping applications (especially serverless/microservice software) to disaggregated clusters?

If and when these challenges are solved, we can evolve data centers into being *fully disaggregated and programmable* (*i.e.*, a *FDP-DC*). This positioning paper provides some guidelines and lays out one potential solution to build such a data center. We decouple the overarching goal into four sub-problems, as shown in Figure 1.

The first problem is a user-facing one, where we need to decide how users can program and run their applications on a FDP-DC (*i.e.*, a FDP-DC's abstraction). We envision two types of abstractions, and a FDP-DC can adopt either one of them or both of them. The first abstraction is a backward-compatible one, where users are not aware of the disaggregation or programmable-hardware nature. They would either assume that their programs run on a virtual machine or are completely server-agnostic (*i.e.*, a serverless model). The second abstraction exposes (to some extent) the underlying disaggregated and programmable nature of FDP-DC to applications. Although this abstraction requires more developer effort, we expect it to yield better performance as users can directly control and leverage low-level systems features such as disaggregation-aware data/compute placement, failure handling, and network communication.

The second problem is how to map applications to the FDP-DC hardware and system infrastructure. Similar to the traditional server setting, we believe that a compiler is needed for FDP-DC. Different from traditional servers, both applications and hardware in a FDP-DC are heterogeneous. To easily manage such heterogeneity, we propose to use an Intermediate Representation (IR) for the middle layer. Our proposed IR centers around the concept of disaggregated execution units, or *codelets*, the unit for scheduling and execution. In addition to code and data, we encapsulate various execution features or hints in each codelet, such as a replication factor and security features. We propose to use MLIR [39] to decompose a program into multiple smaller *codelets* and a companion DAG dictating the execution order of the codelets. Our compiler can further add features to the DAG edges such as communication patterns between codelets.

The third problem is building the hardware infrastructure in a FDP-DC. We provide guidelines for building a disaggregated device and identify three key features for it: network connectivity, hardware virtualization, and multi-tenancy isolation/security. In addition, a device could offer some programmability or configurability. We then propose a network design to connect disaggregated devices. We envision a network topology that is reconfigurable, with the help of circuit switches and/or packet switches with cut-through forwarding. Such a dynamic topology could better fit different application needs and hardware availability [69, 70]. On top of this topology, we propose a programmable network infrastructure that consists of programmable switches and multi-host SmartNICs, which application codelets and provider management tasks can be offloaded to [32, 33, 42]. We advocate the use of multi-host SmartNICs, which consolidates network functionalities of multiple endpoints to one device, instead of one SmartNIC per endpoint. We further pool these multi-host SmartNICs together into a pool, effectively disaggregating network functionalities [56]

With applications, compiler, and hardware infrastructure in place, the last missing piece is a runtime management system (*i.e.*, an operating system) for FDP-DC. The FDP-DC OS would oversee all resource pools and the network system and map (*i.e.*, schedule) codelet DAGs to the underlying infrastructure. It will use the hints from the compiler when scheduling. For example, based on the DAG edges, the OS can choose a network topology and potentially initiate a reconfiguration of the network. It would also choose the right devices to launch codelets, monitor load, and potentially migrate codelets based on load changes.

To illustrate how the above four components work together, we now briefly discuss the end-to-end development and execution flow of a data processing system. First, developers of the system can annotate each operator (*e.g.*, a select, an aggregator) to indicate what resources it needs, whether it is intended to run on an accelerator, etc. They will also capture the processing pipeline (*e.g.*, the output of a SQL query optimizer) as a DAG of operators. Then, our compiler will generate codelets and DAG of codelets based on the DAG of operators. During this process, our compiler would decide the scope of a codelet (*e.g.*, one operator or multiple highly-correlated operators) and compile each codelet into multiple binaries for heterogeneous devices (*e.g.*, one for FPGA and one for CPU). During runtime, our OS will schedule the codelet DAG based on resource availability and dynamically choose the hardware device to run a codelet (*e.g.*, if no FPGA is available, we would run a codelet on CPU). The OS will also use the DAG edge information to decide a network topology and network policy for the workload. Based on the network topology, the OS may decide to execute some codelet on a programmable network device (*e.g.*, performing caching on a programmable switch). Finally, the hardware network devices will each execute a codelet in a virtualized and isolated environment.

With these initial proposals, this vision paper outlines one possible path for data center designers to build a fully disaggregated and programmable data center. While our proposal may appear drastic, we hope some of it can be useful when dealing with challenges in today's fast-changing workloads and hardware landscape. The solutions we proposed are by no means complete. We call for more contribution in this space.

2 Background and Related Works

This section presents background and related works in programmable/reconfigurable networks, resource disaggregation, and programming models.

2.1 Programmable and Reconfigurable Network

As cloud traffic has doubled roughly every year since 2005, the data center networking infrastructure is constantly changing and is never short of innovations [10]. Around the late 2000s, software-defined networking was proposed to improve management efficiency by decoupling the *control plane* (which decides how to handle the traffic) from the *data plane* (which forwards traffic according to decisions that the control plane makes) and then consolidating the control plane onto a set of commodity servers [18, 19, 37]. Recently, the p4 switch [13] and emerging heterogeneous networking devices [31, 56] further empower the data plane with unprecedented programmability and in-network computing at various link locations. Nowadays, both the data and control planes in data center networks are programmable and able to run customized user computation [19, 33].

Furthermore, emerging switching solutions and fabrics are challenging the status quo on how we interconnect data center hardware. For instance, optical circuit switch can reconfigure the network topologies by changing the physical connections among devices to best fit workload's network traffic pattern [69, 70], rendering a drastic departure from the fixed-topology models [26, 59]. On the other hand, emerging fabrics such as CXL [15, 25, 43, 47] offer extremely low-latency interconnect solutions for disaggregated devices. Overall, today's data-center network infrastructure is more programmable, modularized, and flexible, enabling the realization of a FDP-DC.

2.2 Hardware Resource Disaggregation

Hardware resource disaggregation is a proposal that breaks regular servers into segregated, network-attached hardware resource pools. It promises to improve a data center's manageability and resource utilization.

We briefly discuss related works of hardware resource disaggregation in chronological order of four phases. Initially around 2009, disaggregation was proposed to mitigate memory capacity issues [44], which got limited attention and follow-up works. There was a renewed interest around 2016 as the network speed improved to a point where disaggregation can be realistic. As such, the second phase sees a spike of infrastructure [22], low-level systems [8, 27, 28, 35], and operating systems (OS) [55] targeting disaggregation. The third phase moves up the stack with systems co-designing disaggregation with applications or language runtime [46, 51, 61, 66–68, 74, 75, 82]. Now we are at the fourth phase, which aims for real deployment of resource disaggregation. On the one hand, researchers are seeking how to match application programming models and cloud services with disaggregated hardware [62, 79]. On the other hand, researchers and practitioners are building real hardware and deploying them in data centers [24, 25, 43, 47, 56, 80]. This vision paper is not proposing a next phase. Instead, we are envisioning areas where the next phase may prosper.

2.3 Programming Models and Runtime Support

For deploying and running workloads on emerging disaggregated architectures, two approaches have been proposed. First is to design new programming models [51], allowing application or library writers to explicitly annotate and characterize their workload's behavior. The runtime then utilizes this information to inform optimizations on memory management, data structures and concurrency mechanisms. The second approach transparently provides disaggregation and focus on improving existing runtime and operating systems [3, 55, 66, 67]. Existing works focus on adapting and optimizing components such as garbage collector [66, 67] and swap system [3, 55] for disaggregated architectures.

In contrast, compiler support has been lacking for disaggregation. With more heterogeneous hardware being adopted by data centers, compilers targeting individual architectures have seen adoption [12, 48, 53]. Unfortunately, they are not designed for a disaggregated architecture and miss many opportunities for disaggregation-specific optimizations.

3 FDP-DC Design

In a FDP-DC, all devices and networking hardware are programmable and disaggregated from each other. Moreover, the network topology can be dynamically reconfigured to best match workload requirements. Figure 1 illustrates the four components of a FDP-DC solution. The abstraction and compiler components are user facing and aims to enhance the usability of FDP-DC. The OS and hardware infrastructure components are the building blocks of FDP-DC and the environment that executes compiler outputs. Below, we discuss each of them in more details.

3.1 Abstractions and Usage Models

We support two types of abstractions: a backward-compatible, transparent abstraction to support legacy server programs and serverless programs, and a disaggregation-native abstraction that exposes some of the underlying FDP-DC features for programmers to better manage their applications.

Transparent abstraction. Today's applications all run on a server setting, usually on a hypervisor or a container. To continue support these applications, we can virtualize the underlying FDP-DC hardware into a virtual machine abstraction. LegoOS [55] has demonstrated the feasibility of offering a Linux compatible interface; it also supports the concept of vNode, which is a virtual node that can span multiple physical nodes or be a part of one physical node. We envision similar approaches to be taken to provide other traditional virtual interface, such as the container interface.

Besides the above server-based interface, there are also a fair amount of applications that are deployed on serverless computing frameworks today. These applications are server-agnostic and give the underlying system more freedom to choose the execution environment. Thus, a FDP-DC system can potentially map one serverless function onto one device and use a DAG of serverless functions as the input to the FDP-DC compiler.

Disaggregation-native abstraction. Transparent abstractions require no developer effort but cannot fully exploit various performance optimization opportunities. We propose another type of abstraction, one that is *disaggregation-native*. The core idea is to expose the heterogeneous hardware nature and co-design software with it. Developers can either use a new interface or annotate their existing programs to give hints or specify requirements for *how* their programs would run on a FDP-DC. Such an abstraction allows users to more freely and closely manage the way to execute their programs and can potentially improve the application performance and/or

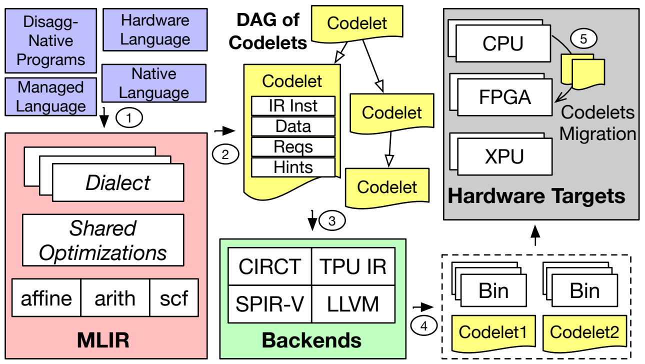

Figure 2: IR and Compilation Flow. *CIRCT is a backend for FPGA. SPIR-V is an IR for GPU. A codelet can be compiled into multiple executable binaries, or bitstreams.*

reduce its execution cost. Below, we list three potential ways of explicitly exposing FDP-DC infrastructure information.

First, programmers could annotate their data and code to control hardware choice, placement, co-location, and fault handling. For example, they could indicate what data structure can be placed in disaggregated memory, similar to the AIFM interface [51], and specify the replication factor of a data structure. They could also give hints on what types of compute hardware (CPU, FPGA, etc.) a function can run on.

Second, we can expose device or network failure domains to developers or application systems administrators. They can then pack functions or data structures that can fail together in the same failure domain and further specify different failure handling mechanisms for each domain, For example, they may want to replicate important user data but are OK with losing intermediate data.

Third, programmers can specify a taskflow as a DAG to represent their application. They can also specify multiple feasible DAGs for the compiler and the run-time system to choose a best one based on the available network topologies and resources in a FDP-DC.

3.2 Intermediate Representation

To support heterogeneous hardware and applications, we propose to use a disaggregation-native intermediate representation (IR) and have a compiler to manipulate the mapping between different representations. Our IR centers around the concept of *codelet*s. A codelet encapsulates code and data that is closely related (and thus should execute together). It is the smallest unit of scheduling and execution. It can also have additional features like target hardware architecture(s) and reliability policy. Codelets in a program are organized into a DAG, whose edges represent the communication between codelets. computation, policies, network topology configurations, and scheduling primitives. and a companion DAG dictating the execution order of codelets.

On top of the codelet concept, we further propose a compilation flow based on MLIR (Multi-Level Intermediate Representation) [40]. The basic idea is to use multiple levels of IRs to capture different levels of code optimization opportunities. For each layer, we can define disaggregation-specific optimizers. Specifically it works as follows (Figure 2). The MLIR framework takes existing languages or our disaggregation-native programming model as inputs (①). Within MLIR, there could be multiple domain-specific dialects for distinct inputs, *e.g.*, a *p4-dialect* for P4 programs. We use a universal layer to carry out common optimizations. The optimized dialects are then lowered onto a codelet-based IR, which packages instructions that are closely related to one codelet (②). We will also associate execution requirements and hints with each codelet. Then, we will compose different codelets together into one or multiple DAGs. Subsequently, the codelets are transformed onto various backends for final compilation (③). We can compile the codelet for distinct hardware targets using existing compilers such as LLVM or CIRCT [1]. Multiple binaries or bitstreams will be produced for each codelet together with generated APIs that can interact with the states within a codelet binary (④). Finally, codelets are executed and can be migrated across devices when load changes (⑤).

3.3 Hardware Infrastructure

The FDP-DC hardware infrastructure has two components. **(1)** The disaggregated resource pools each hosts a type of resource (*e.g.*, compute, memory, and storage) and can be built, scaled, and managed independently. Existing systems built for disaggregation [11, 55] can run atop of them without changes. **(2)** The networking infrastructure consists of circuit switches and programmable networking devices. Network functionalities are first disaggregated from other resource pools and then consolidated into a standalone networking pool, following the network disaggregation idea initially proposed by SuperNIC [56]. We improve SuperNIC by incorporating more types of programmable networking devices into the networking pool, by proposing a data-center-wide network solution, and by using circuit switches to enable reconfigurable topologies.

3.3.1 Disaggregated Devices

To facilitate future development of more types of disaggregated device, we identify three core functionalities for a disaggregated device and discuss how to provide each of them.

The first is **network connectivity**, an essential feature that all disaggregated devices should have. Different from traditional servers which is each equipped with a NIC or SmartNIC, we believe that disaggregated devices only need basic connectivity. Its sole job is to send and receive data packets for the *last hop*, and only need to provide basic physical and link layer functionalities. Ethernet and PCIe can both work for the basic connectivity. Emerging interconnects like

CXL [15, 25, 43, 47] can also work, which provides additional coherence features. As will be discussed in §3.3.2, we propose to disaggregate all remaining network functionalities like transport layers into a separate network resource pool.

Another basic feature is **multi-tenancy and virtualization support**, which enables safe and fair sharing of hardware resources with a flexible virtualized interface. Multi-tenancy and virtualization have been extensively studied for traditional server settings, but applying them to disaggregated devices poses new challenges. First, when virtualizing a disaggregated device, we need to ensure the scalability of the virtual system, since one device is anticipated to serve many clients at the same time. For example, the disaggregated memory device that we built [80] implements a virtual memory system that can support TBs of memory and thousands of concurrent virtual address space. We believe that similar scalability considerations should be incorporated when developing new virtual systems. Second, since a disaggregated device can have more than one computing resource, we need to ensure the overall fairness of all different types of resources for multiple tenants. For instance, the SuperNIC disaggregated network device that we recently built [56] includes ASIC, FPGA, memory, and ingress/egress bandwidth resources. In addition to ensure proper isolation for each of them, we develop a fairness policy that considers multiple types of resources and both time- and space-multiplexing. Finally, device designers should seek software-hardware co-design opportunities and properly split virtualization/multi-tenancy tasks between hardware and software and between client and server side. A common approach is to put the data plane (*e.g.*, virtual memory to physical memory address mapping) in hardware and the control plane (*e.g.*, memory allocation) in software, which would also work for disaggregated devices [80].

Last but not least, **security features** could be added for environments that require in-depth security guarantees. We identify that a disaggregated device can provide three levels of security defenses. The first level is ensuring basic data encryption, authentication, and authorization and secures the device from malicious entities in the network. Unfortunately, security mechanisms required for this level are missing in many recent devices [58, 72, 80]. The next level provides stronger security guarantees by delivering confidential computing, allowing users to offload highly-sensitive computation and data to untrusted cloud providers and their management stack. Though TEEs and confidential computing have been studied individually for resources like CPU [9, 14, 41], FPGA [77, 78], and GPU [29, 63], they share the same challenges when applied to disaggregated devices which commonly use SoCs [80] that could contain a heterogenous set of CPUs, GPUs, and FPGAs.

Finally, the last level provides mechanism and tools to mitigate covert [20, 30] and side-channels [29, 71]. This level should be highly configurable as some security features such

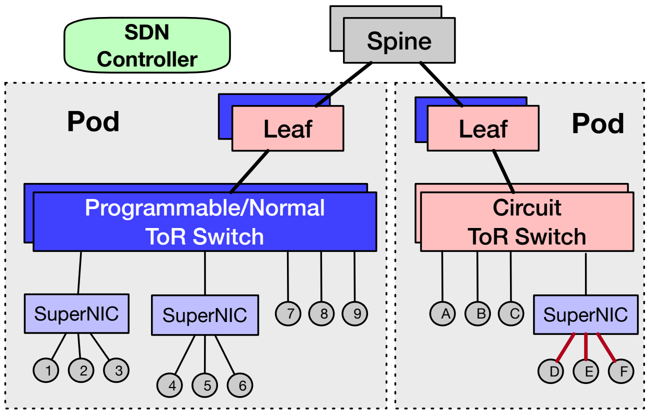

Figure 3: Proposed Network Infrastructure. *Gray circles represent devices or servers. ToR switches can be programmable, circuit, or normal switches. Network programmability is enabled by all the blue-colored boxes. Dynamic topology is enabled by red-colored circuit switches* ***and*** *blue-colored devices with cut-through forwarding. Black links are standard Ethernet links. Red links could be novel links such as GNet [24].*

as oblivious communication [60, 64] may have large impact on performance. In addition, techniques for performing untrusted computation on secrets also has a wide range of performance to security trade-offs. Examples include sandboxing techniques [30] and secure computation algorithms such as homomorphic encryption [38, 52]. Allowing each tenant to make their own trade-offs will be critical in supporting a wide range of applications.

In all, we believe future disaggregated devices *should* invest in all or some of the following features: network connectivity, multi-tenancy and virtualization support, and security defenses. The key technical challenges are dealing with multiple or novel computing resources.

3.3.2 Data Center Networking We now discuss issues related to the data-center networking. Figure 3 shows the envisioned architecture in which the data-center network is organized as pods and managed by a logically centralized SDN controller [7, 19, 24].

Topology and deployment scale. When disaggregating resources from each other, the network communication delay between them would be a key limiting factor of the end-to-end application performance in a FDP-DC. Thus, we should try to have fewer network hops between communicating devices. Meanwhile, recent study [43] shows that a handful of compute nodes are enough to fully utilize one disaggregated memory device. Thus, we believe that the future FDP-DC would go with a hierarchical topology where the lowest level (which we call a pod) contains hundreds of disaggregated devices that are expected to communicate fairly frequently. A pod is connected with at most one switch and possibly a disaggregated network resource pool (to be discussed soon). Cross-pod communication is less frequent but has a larger radius, expanding the entire data center. This topology is similar to the recent Google Aquila work [24] which is targeted for enabling low-latency data-center network communication.

Network programmability. Today's data-center network is becoming more programmable, with the emergence of SmartNICs, programmable switches [13], and multi-host SmartNICs [2, 56]. Together, they make a rich set of hardware resources such as RMT, FPGA, and CPU available to perform in-network computing. We treat the network as a first-class disaggregated resource. Essentially, we can view the collection of programmable switches and NICs as a pool of network resources where endpoints can offload their network tasks to [56]. We can further consolidate network resources by enabling tighter sharing and multi-tenancy on devices in a network resource pool. Together, the disaggregated network pool could lower the total CapEx and OpEx costs and make network programmability easier to manage. Not that these benefits inherit the benefits of resource disaggregation, and we believe many techniques used in building other disaggregated resource pools can be used for networking as well.

Dynamic reconfigurable topology. We propose to enable dynamic reconfigurable topology on top of fixed physical network deployment. Specifically, we want to build conceptual point-to-point connections among selective devices to avoid in-network buffering. At its core, we use circuit-switching to create temporal links and reconfigurable topologies are realized by adjusting those temporal links. As Figure 3 shows, we think both optical circuit switches and packet-based switches with cut-through forwarding can be used. Nonetheless, we believe a key open challenge is to develop an efficient scheduling policy co-designed with the infrastructure along with the running workloads [17, 57, 69, 70].

3.4 Operating Systems for FDP-DC

Finally, there is the operating system or control plane that oversees all the disaggregated and network infrastructure. This data-center-wide OS is responsible for resource allocation, task scheduling, health monitoring, load balancing, failure handling, etc.

Resource Management. For resource management, the challenge is to efficiently map applications to FDP-DC resources. One of the promises that FDP-DC brings is going beyond the physical constraints of the traditional monolithic server. In order to achieve this, two things need to be realized: mapping codelets to pools of resources and making codelets unlimited to individual device's resource constraints. Moreover, the scheduler's fairness policy and allocation mechanism should factor in architectural differences between disaggregated devices. To navigate through these challenges and scale well, we adopt a two-level approach similar to LegoOS [55] in which a global manager only performs coarse-grained, architecture-agnostic allocations while the specific devices perform finer-grained, architecture-aware allocation [36, 73].

Fault Tolerance. To achieve fault tolerance, the OS can expose a menu of abstractions for failure handling such as best-effort handling and transparent handling [81]. The former exposes failures to applications while the latter hides them, allowing users to choose the one they see fit. The OS also needs to consider the topology when meeting fault tolerance and SLA requirements as rack-scale fault domains may change as we introduce reconfigurable topology.

Load Balancing. We observe that the control plane's most challenging task will be load balancing computation, data, and network bandwidth among disaggregated devices at high speed during runtime. Most notably, disaggregation architecture decouples the relationship between how scaling the application affects network bandwidth requirements, also known as the "disaggregation tax" [62]. The inclusion of network bandwidth results in a complicated choice matrix concerning device capability, data location, network topology, and network bandwidth. Traditional methods may fall short in making such decisions fast enough to produce reasonable outcomes. In response, we identify two non-exclusive approaches to tackle this issue. First, we can utilize reinforcement learning to transform it into a learning problem. Similar methods are proven effective for OS [76], database [49], and data structures [16]. Second, we plan to *onload* this task from the operating system into the application layer by introducing explicit data movement [62] and scheduling primitives [51] in the IR layer (§3.2).

4 Conclusion

This paper presents one vision into building a futuristic fully disaggregated and programmable data center. Our proposed solution incorporates four key components: easy-to-use abstractions, flexible compiler optimizations, scalable systems management, and efficient and programmable hardware. We hope this vision or part of it can help researchers and practitioner in solving some of the future application/hardware challenges in data centers.

Acknowledgment

We would like to thank Yizhou Shan' thesis committee members, Geoffrey M. Voelker, Stefan Savage, Alex C. Snoeren, and George C. Papen. Their insightful questions during his defense motivate us to write this paper. We thank the anonymous reviewers, Sanidhya Kashyap, Chenxi Wang, Qizhen Zhang, and Pengfei Zuo for their valuable feedback. This material is based upon work supported by the National Science Foundation under the following grant: NSF 2022675.

References

[1] CIRCT. https://circt.llvm.org/.

[2] NVIDIA BLUEFIELD DATA PROCESSING UNITS. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[3] 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, 2020.

[4] Amazon. Amazon EC2 Elastic GPUs. https://aws.amazon.com/ec2/elastic-gpus/.

[5] Amazon. AWS Nitro System. https://aws.amazon.com/ec2/nitro/.

[6] Amazon. Amazon s3. https://aws.amazon.com/s3/, 2019.

[7] Manikandan Arumugam, Deepak Bansal, Navdeep Bhatia, James Boerner, Simon Capper, Changhoon Kim, Sarah McClure, Neeraj Motwani, Ranga Narasimhan, Urvish Panchal, Tommaso Pimpo, Ariff Premji, Pranjal Shrivastava, and Rishabh Tewari. Bluebird: High-performance SDN for bare-metal cloud services. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[8] Krste Asanović. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers, February 2014. Keynote talk at the 12th USENIX Conference on File and Storage Technologies (FAST '14).

[9] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. CURE: A security architecture with CUstomizable and resilient enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1073–1090. USENIX Association, August 2021.

[10] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Benn Thomsen, Kai Shi, and Hugh Williams. Sirius: A flat datacenter network with nanosecond optical switching. In *SIGCOMM 2020*. ACM, August 2020.

[11] Laurent Bindschaedler, Ashvin Goel, and Willy Zwaenepoel. Hailstorm: Disaggregated compute and storage for distributed lsm-based databases. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, 2020.

[12] 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.*, 2014.

[13] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *SIGCOMM Comput. Commun. Rev.*, 2013.

[14] Costan, Victor and Devadas, Srinivas. Intel SGX Explained. https://eprint.iacr.org/2016/086.pdf.

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

[16] Yifan Dai, Yien Xu, Aishwarya Ganesan, Ramnatthan Alagappan, Brian Kroth, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. From WiscKey to bourbon: A learned index for Log-Structured merge trees. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*.

[17] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, 2010.

[18] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: An intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 2014.

[19] Andrew D. Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, Richard Alimi, Shawn Shuoshuo Chen, Mike Conley, Subhasree Mandal, Karthik Nagaraj, Kondapa Naidu Bollineni, Amr Sabaa, Shidong Zhang, Min Zhu, and Amin Vahdat. Orion: Google's Software-Defined networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021.

[20] Nicole Fern, Ismail San, Çetin Kaya Koç, and Kwang-Ting Tim Cheng. Hiding hardware trojan communication channels in partially specified soc bus functionality. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(9):1435–1444, 2017.

[21] 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 *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*.

[22] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[23] 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)*, 2021.

[24] 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)*, 2022.

[25] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access, High-Performance memory disaggregation with DirectCXL. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022.

[26] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, 2009.

[27] 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)*.

[28] Hewlett-Packard. The Machine: A New Kind of Computer. http://www.hpl.hp.com/research/systems-research/themachine/.

[29] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J. Rossbach, and Emmett Witchel. Telekine: Secure computing with cloud GPUs. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*.

[30] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 533–549, Savannah, GA, November 2016. USENIX Association.

[31] Intel. Intel Unveils Infrastructure Processing Unit. https://www.intel.com/content/www/us/en/newsroom/news/infrastructure-processing-unit-data-center.html.

[32] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. Netchain: Scale-free sub-rtt coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[33] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 121–136, 2017.

[34] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Matt Dau Mike Daley, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Harshit Khaitan Alexander Kaplan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Matt Ross Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Andy Swing Dan Steinberg, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*.

[35] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dReDBox project vision. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE '16)*.

[36] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J. Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018.

[37] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, 2010.

[38] Sam Kumar, David E. Culler, and Raluca Ada Popa. MAGE: Nearly zero-cost virtual memory for secure computation. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 367–385. USENIX Association, July 2021.

[39] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: A compiler infrastructure for the end of moore's law. *arXiv preprint arXiv:2002.11054*, 2020.

[40] Chris Lattner, Jacques A. Pienaar, Mehdi Amini, Uday Bondhugula, River Riddle, Albert Cohen, Tatiana Shpeisman, Andy Davis, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: A compiler infrastructure for the end of moore's law. *CoRR*, abs/2002.11054, 2020.

[41] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, 2020.

[42] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, and Abhishek Bhattacharjee. Mind: In-network memory management for disaggregated data centers. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, 2021.

[43] Huaicheng Li, Daniel S Berger, Stanko Novakovic, Lisa Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Ishwar Agarwal, Mark Hill, Marcus Fontoura, et al. First-generation memory disaggregation for cloud platforms. *arXiv preprint arXiv:2203.00241*, 2022.

[44] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*.

[45] LISA'17. Disaggregating the Network: Switching as a Service. https://www.usenix.org/conference/lisa17/conference-program/presentation/schiff.

[46] Haoran Ma, Shi Liu, Chenxi Wang, Yifan Qiao, Michael D. Bond, Stephen M. Blackburn, Miryung Kim, and Guoqing Harry Xu. Mako: A low-pause, high-throughput evacuating collector for memory-disaggregated datacenters. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2022, 2022.

[47] 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. *arXiv preprint arXiv:2206.02878*, 2022.

[48] Nvidia. Cuda Compiler Driver NVCC. https://docs.nvidia.com/cuda/pdf/CUDA_Compiler_Driver_NVCC.pdf, 2022. [Online; accessed 19-July-2022].

[49] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. Self-driving database management systems. In *CIDR*, volume 4, page 1, 2017.

[50] Parthasarathy Ranganathan, Daniel Stodolsky, Jeff Calow, Jeremy Dorfman, Marisabel Guevara, Clinton Wills Smullen IV, Aki Kuusela, Raghu Balasubramanian, Sandeep Bhatia, Prakash Chauhan, et al. Warehouse-scale video acceleration: co-design and deployment in the wild. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[51] 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)*, 2020.

[52] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 238–252, 2021.

[53] Eric Schkufza, Michael Wei, and Christopher J. Rossbach. Just-in-time compilation for verilog: A new technique for improving the fpga programming experience. ASPLOS '19, 2019.

[54] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J Yadwadkar, Raluca Ada Popa, Joseph E Gonzalez, Ion Stoica, and David A Patterson. What serverless computing is and should become: The next phase of cloud computing. *Communications of the ACM*, 64(5):76–84, 2021.

[55] 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)*.

[56] Yizhou Shan, Will Lin, Ryan Kosta, Arvind Krishnamurthy, and Yiying Zhang. Disaggregating and Consolidating Network Functionalities with SuperNIC. *arXiv preprint arXiv:2109.07744*, 2021.

[57] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A Network Architecture for Disaggregated Racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19)*.

[58] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. Strom: Smart remote memory. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, 2020.

[59] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Sigcomm '15*, 2015.

[60] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. *J. ACM*, 65(4), apr 2018.

[61] Shin-Yeh Tsai, Yizhou Shan, and Yiying Zhang. Disaggregating Persistent Memory and Controlling Them Remotely: An Exploration of Passive Disaggregated Key-Value Stores. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020.

[62] Lluís Vilanova, Lina Maudlej, Shai Bergman, Till Miemietz, Matthias Hille, Nils Asmussen, Michael Roitzsch, Hermann Härtig, and Mark Silberstein. Slashing the disaggregation tax in heterogeneous data centers with fractos. In *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022.

[63] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*.

[64] Midhul Vuppalapati, Kushal Babel, Anurag Khandelwal, and Rachit Agarwal. SHORTSTACK: Distributed, fault-tolerant, oblivious data access. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 719–734, Carlsbad, CA, July 2022. USENIX Association.

[65] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. Building an elastic query engine on disaggregated storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[66] 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)*, 2020.

[67] Chenxi Wang, Haoran Ma, Shi Liu, Yifan Qiao, Jonathan Eyolfson, Christian Navasca, Shan Lu, and Guoqing Harry Xu. MemLiner: Lining up tracing and application for a Far-Memory-Friendly runtime. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022.

[68] Qing Wang, Youyou Lu, and Jiwu Shu. Sherman: A write-optimized distributed b+ tree index on disaggregated memory. *arXiv preprint arXiv:2112.07320*, 2021.

[69] Weitao Wang, Dingming Wu, Sushovan Das, Afsaneh Rahbar, Ang Chen, and T. S. Eugene Ng. RDC: Energy-Efficient data center network congestion relief with topological reconfigurability at the edge. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[70] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Zhijao Jia, Dheevatsa Mudigere, Ying Zhang, Anthony Kewitsch, and Manya Ghobadi. Topoopt: Optimizing the network topology for distributed dnn training. *arXiv preprint arXiv:2202.00433*, 2022.

[71] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2421–2434, 2017.

[72] Zeke Wang, Hongjing Huang, Jie Zhang, Fei Wu, and Gustavo Alonso. FpgaNIC: An FPGA-based versatile 100gb SmartNIC for GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022.

[73] Yue Zha and Jing Li. Virtualizing fpgas in the cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, 2020.

[74] Ming Zhang, Yu Hua, Pengfei Zuo, and Lurong Liu. Ford: Fast one-sided rdma-based distributed transactions for disaggregated persistent memory. In *20th USENIX Conference on File and Storage Technologies (FAST 22). USENIX Association*.

[75] Qizhen Zhang, Yifan Cai, Xinyi Chen, Sebastian Angel, Ang Chen, Vincent Liu, and Boon Thau Loo. Understanding the effect of data center resource disaggregation on production dbmss. *Proc. VLDB Endow.*, may 2020.

[76] Yiying Zhang and Yutong Huang. " learned" operating systems. *ACM SIGOPS Operating Systems Review*, 53(1):40–45, 2019.

[77] Mark Zhao, Mingyu Gao, and Christos Kozyrakis. Shef: Shielded enclaves for cloud fpgas. ASPLOS 2022.

[78] Mark Zhao and G. Edward Suh. Fpga-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*.

[79] Mohammad Shahrad Zerui Wei Bili Dong Jinmou Li Ishaan Pota Harry Xu Yiying Zhang Zhiyuan Guo, Zachary Blanco. Resource-Centric Serverless Computing. *arXiv preprint arXiv:2206.13444*, 2022.

[80] Zhiyuan Guo and Yizhou Shan and Xuhao Luo and Yutong Huang and Yiying Zhang. Clio: A hardware-software co-designed disaggregated memory system. In *the 27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*, Lausanne, Switzerland, March 2022.

[81] Yang Zhou, Hassan M. G. Wassel, Sihang Liu, Jiaqi Gao, James Mickens, Minlan Yu, Chris Kennelly, Paul Turner, David E. Culler, Henry M. Levy, and Amin Vahdat. Carbink: Fault-Tolerant far memory. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022.

[82] Pengfei Zuo, Jiazhao Sun, Liu Yang, Shuangwu Zhang, and Yu Hua. One-sided RDMA-Conscious extendible hashing for disaggregated memory. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021.