Towards a Fully Disaggregated and Programmable Data Center

Yizhou Shan*[1], Will Lin[2], Zhiyuan Guo[2], and Yiying Zhang[2]

[1]Huawei Cloud
[2]UC San Diego

Abstract

The data center deployment status quo is being challenged by the pressure from the applications it hosts and the hardware it runs on. On one hand, the applications are becoming more fine-grained driven by the recent trend of serverless and microservice. On the other hand, the hardware is increasingly faster, heterogeneous, and specialized as a result of recent network improvements and hardware resource disaggregation trend. Data center designers are at a crossroads facing several challenges when these two trends meet.

In this paper, we envision a futuristic data center consisting of disaggregated devices connected by a highly flexible network. We further propose a set of guidelines for data center designers to navigate design trade-offs. Specifically, we propose a three-layer approach. At the top layer, we use a disaggregation-native methodology to co-design the software with the hardware infrastructure using explicit annotations. At the bottom layer, we describe the hardware infrastructure and key features required to build disaggregated devices and reconfigurable networks. Between these two layers, we propose an MLIR-based intermediate representation layer to decouple the hardware from the software. It transforms software into small executable units to tame heterogeneity and aid runtime migration. Our envisioned model could unlock the possibility of complete hardware customization for large-scale workloads and domain users.

1 Introduction

For decades, data centers are organized as racks of general-purpose servers connected with a static topology of fixed-logic network switches, hosting workloads from various industries and domains. However, this status quo is being challenged by the pressure from the applications it hosts and the hardware infrastructure it runs on.

On one hand, data center applications are becoming more fine-grained, driven by the recent trend of serverless and microservice. Consequently, data center customers can focus on their business logic and avoid deployment chores. Essentially, cloud vendors lift the abstraction using a layer of indirection [22], which hides all the complexities of hardware heterogeneity, auto-scaling, and failure handling.

Besides, the data center hardware infrastructure is increasingly faster, programmable, and specialized. Several factors contribute to this trend. First, vendors are customizing hardware to avoid energy and computing inefficiencies as a result of the ending of Moore's Law and Dennard scaling [2, 3, 29]. Second, the data center network (DCN) is not only faster with single-digit microsecond latency and more than 100 Gbps bandwidth [4, 19], but also more flexible with programmable control [13, 15] and data planes [7, 13, 15, 16]. Finally, the above two trends together enabled hardware resource disaggregation, an architecture that breaks servers into network-attached hardware resource pools [6, 23, 43]. We make a key observation that both computing and non-computing-oriented (*e.g.*, memory) disaggregated devices will incorporate some computation power such as FPGA. Consequently, disaggregated devices are usually programmable and could host application offloads to realize novel paradigms such as near-data processing.

These trends enable great opportunities, the data centers designers, however, are facing several key challenges when they try to run emerging systems on new hardware. First, software and hardware are not evolving at the same pace, with hardware having a much longer lead time. This mismatch can result in lockstep development and delay time to market [16]. Second, there is a semantic gap between the software and the hardware. Unlocking the full potential of new hardware, especially heterogeneous devices such as FPGA or RMT [7], requires a non-trivial software redesign, *e.g.*, integrating accelerators with hypervisors [3, 16]. However, developers are usually inertial [34]. Though automatic offloading frameworks can help [38, 40, 41], they fall short in a distributed setting or when dealing with multiple types of heterogeneous resources. In all, it is not easy for data center designers to make full use of the emerging hardware.

Third, though disaggregation has attracted great attention recently, most prior works build or emulate disaggregated hardware using regular servers [6, 30, 43, 48, 63], which suffers from software-based, sub-optimal performance. Since little has been explored on building standalone disaggregated devices, it is not clear what are the commonly required features and what is the best practice. Further, many works have built systems for disaggregation [6, 17, 36, 43, 63]. However, they are usually limited to a few design choices. For

*Work done while working at UCSD.

instance, some delegate the memory address translation service onto disaggregated memory devices [43, 62]. However, this service could also run on a disaggregated computing device or a programmable switch depending on traffic patterns [33]. This begs the question of whether data center designers have explored all the trade-offs among programmability, manageability, performance, and cost.

Lastly, although programmable switches and NICs are gradually deployed in production [35], they often lack flexible programming models [37] and commercial multi-tenancy or hot-plug support [14, 51, 55, 56]. Moreover, the long-standing static network topology is hitting its limitations [5, 53, 54]. It is not clear how data center designers can best co-design network with running workloads.

To help data center designers tackle these challenges, we propose principled guidelines for them to navigate the design trade-offs. The foundation of our vision is a fully disaggregated and programmable data center infrastructure consisting of disaggregated devices connected by a highly flexible DCN. In particular, we use a three-layer approach (Figure 1).

At the top layer, we propose the disaggregation-native design methodology, a set of non-intrusive but informative ways for developers to consciously make their programs in a way that could benefit from the underlying heterogeneous and disaggregated hardware. Specifically, we believe that it is best for developers to directly control the placement of their data/code, failure handling policies, and network topologies by explicitly annotating their program.

Though the above methodology can mitigate the tension to some extend, it cannot address the fundamental semantic gap between the software and the hardware. We therefore propose an intermediate representation (IR) layer to map diverse software onto the underlying hardware infrastructure. The IR layer has several benefits. First, it decouples the software from the hardware, enabling each to evolve independently and the IR can bridge the discrepancy. Second, the IR layer helps software to run on heterogeneous hardware and facilitates runtime migration without any code changes. Moreover, we use the above IR layer to transform high-level disaggregation-native annotations into low-level hardware primitives to reduce developer hazards. Together with the top layer, they address the first two challenges.

In specific, the IR consists of "disaggregatable" execution units, or *codelets*. We use MLIR [32] to decompose a program into multiple smaller *codelets* and a companion DAG dictating the execution order of the codelets. The codelet is a powerful entity designed to exploit the hardware infrastructure. A codelet encapsulates everything that is needed to run a part of a software on multiple devices, *e.g.*, it has data, IR instructions, network topology configurations, and a set of policies such as security requirements. We would compile the codelet for distinct hardware targets using backends such as LLVM or CIRCT [1]. Furthermore, we use the codelet as the smallest unit of migration among devices.

The last layer concerns the physical hardware. We envision a fully disaggregated and programmable hardware infrastructure. In this model, devices are disaggregated from each other into separate resource pools. Each device is designed in a principled way and can host certain type of computations. Moreover, this infrastructure's DCN is programmable and reconfigurable at the same time. It offers in-network computing to accelerate distributed applications [27, 28], also reconfigures network topologies to best fit workload characteristics [53, 54]. We use a disaggregated control plane to oversee all the disaggregated devices and the DCN, responsible for resource management, load balancing, etc. In all, we envision a versatile infrastructure with great flexibilities, hoping that it could meet future data center workloads' fast-changing requirements.

To address the third challenge, we propose principled guidelines to build its disaggregated devices. We identify three key features that any disaggregated devices should have: network connectivity, multi-tenancy and virtualization support, and security defenses, with the last being offered to environments where security is a must.

For the last challenge, we propose a three-step approach. Similar to prior works [12, 53], we first use reconfigurable topology to morph network topology to best match workload characteristics. Different from prior works, we propose to build the reconfigurable topology using both circuit switches and packet switches with cut-through forwarding. Second, we expose both network's programmability and reconfigurability to the IR layer. Workloads can leverage both at the same time. Third, we adapt the network disaggregation idea [44] to aggregate networking hardware into a logical computing plane, orchestrated by a global SDN controller. In all, the network becomes an "ordinary" computing resource of which the workloads can take full control.

We now briefly discuss how a distributed database system could benefit from our proposals. First, developers can annotate SQL operations with distinct network, security, and fault-tolerance requirements, e.g., use a device with trusted execution environment. Then, our IR layer will decompose the database code into multiple codelets and compile each codelet into multiple binaries for heterogeneous devices (*e.g.*, FPGA, RMT, and CPU). During runtime, we will schedule codelets following their embedded policies. If we detect that a codelet is better served on a different device, we would launch a migration. For instance, we can migrate a codelet with SQL aggregations or caching onto a p4 switch to reduce network round-trips. The colelet can also reconfigure network topologies to, say, create virtual racks across multiple physical racks to reduce network latency [53].

With these initial proposals, this vision paper outlines a possible path for data center designers to deal with the fast-changing workloads and hardware landscape. The solutions we proposed are by no means complete. We call for more contribution in this space.

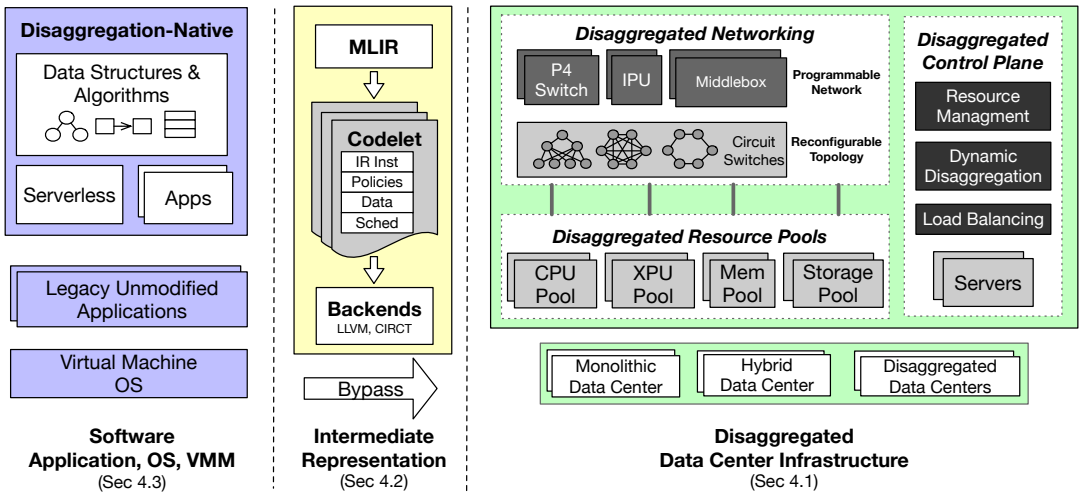

Figure 1: **Overview of a Fully Disaggregated and Programmable Data Center.**

2 Background and Motivation

This section presents recent networking, hardware, and compiler trends motivating our vision.

2.1 Programmable and Reconfigurable Network

As cloud traffic has doubled roughly every year since 2005 [5], the data center networking infrastructure is constantly changing and is never short of innovations. 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 [13, 15, 31]. Recently, the p4 switch [7] and emerging heterogeneous networking devices [26, 44] 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 [15, 28].

Furthermore, emerging switching solutions and fabrics are challenging the status quo on how we interconnect data center hardware. In particular, optical circuit switch reconfigures the network topologies by changing the physical connections among devices to best fit workload's network traffic pattern [53, 54], rendering a drastic departure from the fixed-topology models [21, 47]. On the other hand, emerging fabrics such as Gen-Z [18] and CXL [10, 34] offer feature-rich, energy-efficient, and high-performance data center-scale interconnection solutions for disaggregated devices. In all, the DCN infrastructure is more programmable, modularized, and flexible than ever and is the cornerstone to realizing a fully programmable and disaggregated data center.

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 [43].

Typically, a disaggregated device is a self-contained, standalone board that works without a companion server. It is directly attached to the DCN fabric. Multiple such devices could be tightly packaged into a standard chassis for a tighter deployment, but they still function and fail independently [34]. In general, a disaggregated device has two categories of resources: **a dominant resource** that defines a device's *personality* (*e.g.*, DRAM is a memory device's dominant resource), and **auxiliary resources** such as FPGA or ASIC that make the device functional (*e.g.*, allow network access). Compute-intensive disaggregated devices have dominant resources like CPU, GPU, or TPU. For non-compute disaggregate device, its auxiliary resource is the key to enabling its programmability, which allows user to run the processing near the dominant resource to achieve schemes such as near memory/storage computing.

2.3 Compilers and Programming Frameworks

The confluence of domain-specific abstractions (*e.g.*, graph operators) and accelerators (*e.g.*, GPU and FPGA) calls for the next-generation compilers and programming language (PL) frameworks to reduce software fragmentation and improve compilation for heterogeneous devices.

For instance, MLIR [32] consolidates high-level abstractions onto a shared framework for common code analysis and optimizations. MLIR uses *dialect* to logically group a set of operations, attributes, and types designed for a specific application input (*e.g.*, a tensor-dialect for ML graph operators). Other works such as CIRCT [1], oneAPI [25] provide hardware abstractions spanning diverse accelerators. Nonetheless, our visioned disaggregated data center model has new requirements that no existing PL framework supports: (1) dealing with emerging network architectures such as RMT [7], (2) facilitating runtime migration among heterogeneous devices, (3) primitives for built-in proactive

scheduling and load balancing.

3 Overview

In a fully disaggregated data center, 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 presents the end-to-end deployment and architecture overview of such a data center, which has three main layers: (1) a disaggregated data center infrastructure layer, the enabling factor of our envisioned model, without which the whole end-to-end flow will not exist, (2) an IR layer, and (3) a software design methodology layer. The latter two improve the usability of the underlying infrastructure.

Disaggregated Data Center Infrastructure. At a high-level, the infrastructure has three 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 [6, 43] could use them without changes. **(2)** The networking infrastructure consists of circuit switches and programmable networking devices. Network functionalities are disaggregated from other resource pools and are consolidated into a standalone resource pool. The concept of network disaggregation was initially proposed by a recent work [44]. We enlarge its scope by adding more networking devices into the spectrum and using circuit switches to enable reconfigurable topologies to adjust topology based on software needs dynamically. **(3)** Finally, a control plane runs on a small set of regular servers which is responsible for managing global resources, balancing load, and handling hardware failures.

Software. Analogous to the cloud-native idea, we propose a disaggregation-native, or disaggregation-conscious methodology, whose core idea is to expose the heterogeneous hardware nature and co-design software with it. In specific, we advocate for non-transparency and explicit code annotation to expose heterogeneity, reconfigurable network topologies, and hardware failures to the software. Consequently, users can take full control of their data and computation and customize their failure handling to achieve a desired balance among performance, reliability, and cost. We also expect a backward compatible interface so that legacy application, OS, hypervisor could run without changes.

Intermediate Representation (IR). We propose the IR layer to bridge the gap between domain-specific applications and the hardware infrastructure. We use MLIR [32] to decompose an application into multiple smaller *codelets* and a companion DAG dictating the execution order of codelets. The codelet is a rich entity designed to exploit the underlying hardware infrastructure. A codelet consists of data, purposely-defined IR instructions, scheduling primitives, network topology configurations, generated state-management APIs, and a set of policies such as security requirements. We compile the codelet for distinct hardware targets using backends such as LLVM or CIRCT [1]. A codelet is the smallest unit of data and computation migration. Applications can bypass the IR layer to use the underlying infrastructure directly.

3.1 Key Challenges

We identify the following challenges in achieving our goal.

1. **Build disaggregated hardware devices.** Standalone disaggregated devices are not widely available yet. Only two hardware-based memory and network devices are publicly known [44, 62]. It is not clear what are the required features to build such devices.
2. **Set up disaggregated networking.** Programmable switches are just being deployed recently, other programmable networking devices are still in their infancy [26, 44]. Worse, it is a huge undertaking to replace the existing packet switch-based fixed topology with a circuit switch-based dynamic topology.
3. **Run applications efficiently.** Even though initial works have been proposed to transparently execute traditional applications or to design new programming models for disaggregation, it is unclear how to best execute applications on a disaggregated cluster, as the former incurs significant performance overhead [34, 43] and the latter requires significant programmer efforts [42]. Special care must be taken to avoid scalability bottlenecks and redundant functionalities across stacks (*e.g.*, multiple replications). Moreover, it is unclear how applications could leverage customizable network infrastructure.
4. **Ensure security.** With data and computation spread across heterogeneous devices and over the network, it poses new challenges in delivering security and privacy guarantees. Furthermore, certain users demand confidential computing where data center providers are not trusted to process highly-sensitive data. It is not clear how to achieve end-to-end security in a disaggregated setting.

4 Design and Implementation Considerations

We now take a closer look at every layer required to build a disaggregated data center and present our initial proposals. We start with the hardware infrastructure, which is characterized by disaggregated devices, the networking infrastructure, and a control plane to tie the two together.

4.1 Disaggregated Data Center Infrastructure

4.1.1 Devices

By design, disaggregated devices are directly attached to the network and are also programmable because of their built-in computation-capable hardware resources (§2.2). We identify three core functionalities that any disaggregated devices *should* have. We focus on high-level requirements and challenges rather than implementation details.

The first is **network connectivity**, the must-have feature. Many solutions are available. We categorize them into two types. The first type enables basic ("dummy") network connectivity. Its sole job is sending and receiving data packets.

Ethernet and PCIe fall into this category. The second type enables advanced ("smart") network connectivity. It could have built-in advanced transport and other higher-level services such as remote memory translation. Typical solutions are RoCE, CXL, and Gen-Z. Although the first type only exposes a bare-metal network interface, it allows flexible customization and co-design with other resources. The second type has a more consistent and easy-to-use interface across devices, but the hardened features discourage modifications and may limit scalability [62]. When deciding on a network connectivity solution, one must consider infrastructure availability. For instance, Ethernet is widely available but emerging fabrics are still in the development phase [19, 34].

The second is **multi-tenancy and virtualization support**, the one provides protection, isolation, and fairness for shared resources. Multi-tenancy is a long-standing subject and has been extensively studied on conventional computing resources (*e.g.*, CPU, GPU, and FPGA), but applying it to disaggregated devices poses new challenges. First, since a disaggregated device could have more than one computing resource, a correct multi-tenancy design therefore requires diligent coordination across resource boundaries. For instance, a recent disaggregated memory device has ASIC, FPGA, and CPU, its authors went great lens ensuring proper isolation and protection [62]. The second key challenge arises when novel computing resources are used, especially those with poor multi-tenancy support. For example, RMT, the core computing unit of a programmable switch, currently has no commercial support for multi-tenancy but only a few research proposals [52, 55, 56].

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. It secures the device from potentially unwarranted accesses and from using any tempted data. Unfortunately, security mechanisms required for this level are missing in many recently developed devices [46, 62]. The last two security levels are avoiding covert/side-channels and delivering confidential computing. Both ensure stronger security/privacy guarantees and grant users to offload highly-sensitive computation and data to disaggregated devices. Though both have been studied individually for resources like CPU [9], FPGA [60, 61], and GPU [24, 49], they share the same challenges when applied to disaggregated devices. For example, there is no trusted execution environment (TEE) in any programmable switches.

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.

4.1.2 Networking

Our proposed networking infrastructure is programmable with reconfigurable topologies, managed by a centralized SDN controller. Figure 2 shows its architecture.

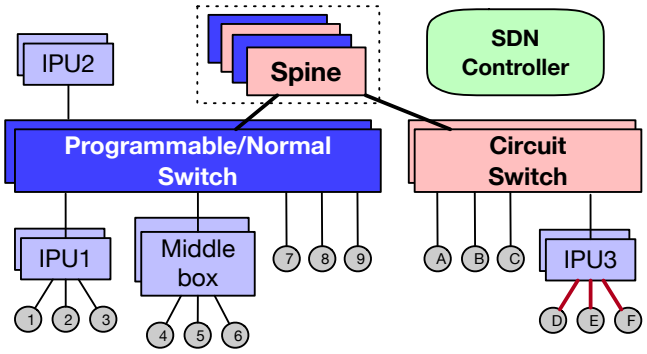

Figure 2: **Proposed Data Center Networking Infrastructure.** *The gray circles are disaggregated devices or servers. ToR can be programmable, circuit, or normal switch. Emerging network devices are usually directly attached to a ToR switch. Network programmability is enabled by all the blue-colored boxes. Dynamic topology is enabled by red-colored circuit switches **and** blue-colored devices with cut-through forwarding. Black links are standard Ethernet or Infiniband links. Red links are novel link solutions such as [19].*

Network Programmability. Network data plane's programmability is enabled by programmable switch and middlebox [7, 57] as well as emerging networking devices such as IPU [26] and SuperNIC [44]. Together, they make a rich set of resources such as RMT, FPGA, and CPU available to perform in-network computing. We treat the network as another first-class disaggregated resource. We view both networking switches/devices as regular disaggregated devices, following the same design principles discussed in §4.1.1. This unification simplifies and provides principled guidelines for future networking device design.

We pose no restriction on how switches, networking devices, and other devices are interconnected. For instance, in Figure 2, `IPU2` is connected to a switch while `IPU1` is connected to both disaggregated devices and the switch. Besides standard Ethernet or Infiniband, we advocate using novel link layer solutions with features such as end-to-end flow control, lossless transmission, and incast prevention to break the rigorous boundaries in layered protocols [19, 20].

Dynamic Reconfigurable Topology. We enable dynamic reconfigurable topology on top of fixed physical network deployment. Specifically, we build conceptual point-to-point connections among selective devices to avoid any in-network buffering. Essentially, we use circuit-switching to create temporal links. We dynamically adjust temporal links to realize reconfigurable topologies. As Figure 2 shows, we use optical circuit switches and packet-based switches with cut-through forwarding for this purpose.

A key challenge in using any circuit-switching solution is to find an efficient scheduling algorithm [12, 45]. We therefore only change topologies when a scheduling plan would clearly benefit the running systems. We identify two potential use cases. The first is to build a logical monolithic server abstraction using a set of disaggregated devices by building

direct connections among them. The second is co-designing the topology with well-defined applications such as ML [54] or workloads with placement constraints [53].

We would use an SDN controller to virtually group all physically separated networking resources into one giant and logical abstraction made available to upper layer users. The controller manages routing, failure handling, etc [4, 15, 19].

4.1.3 Control Plane

The disaggregated control plane oversees all the disaggregated devices and the networking infrastructure. It is responsible for global resource management, health monitoring, load balancing, failure handling, etc. It is similar to the Global Resource Managers proposed by LegoOS [43].

For resource management, we use a *two-level approach* in which the control plane only does coarse-grained allocation while the specific devices do finer-grained ones. We expose a menu of abstractions for failure handling such as best-effort handling and transparent handling. The former exposes failures to applications while the latter hides them. Users could choose the one they see fit.

We observe that the control plane's most challenging task is to load balance computation, data, and packets among disaggregated devices at high speed during runtime. It involves a complicated choice matrix concerning device capability, data location, network topology, and network bandwidth. Traditional methods fall short in making such decisions fast to produce reasonable outcomes.

We identify two non-exclusive approaches to tackle this issue. First, we could utilize reinforcement learning to transform it into a learning problem. Similar methods are proven effective for OS [59], database [39], and data structures [11]. Second, we plan to *onload* this task from the control plane into the application layer by introducing explicit data movement and scheduling primitives in the IR layer (§4.2).

4.2 Intermediate Representation

To facilitate the use of heterogeneous devices and runtime migration, we leverage MLIR to decompose an application into smaller *codelets* and a companion DAG dictating the execution order of codelets. A codelet encapsulates data, computation, policies, network topology configurations, and scheduling primitives. We compile a codelet into binaries or bitstreams for distinct hardware targets together with generated APIs for codelet's state management

As Figure 3 shows, ① MLIR takes existing frameworks or new PL models as inputs. Within MLIR, there could be multiple domain-specific dialects for distinct inputs, *e.g.*, a *p4-dialect* for P4 programs. We also propose to use a shared layer to carry out common optimizations and security checks [8]. ② We would then lower optimized dialects onto a purposely-defined IR. We would package the IR instructions together with other policies and configurations into codelets. ③ Subsequently, the codelets are transformed onto various backends for final compilation. ④ We produce multiple binaries or bitstreams for each codelet together with generated APIs that can interact with the states within a codelet binary. The full IR definition is beyond the scope of this paper, we leave it for future work.

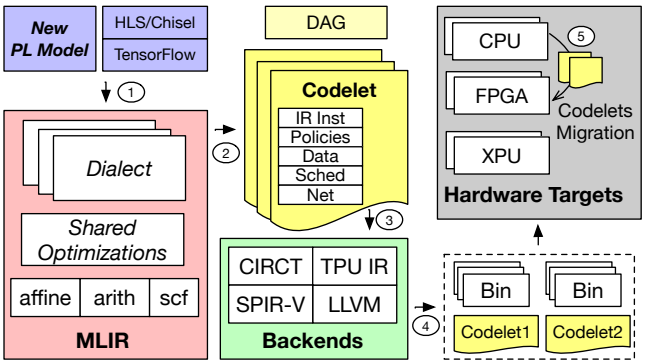

Figure 3: **IR and Compilation Flow.** *CIRCT is a backend for FPGA. SPIR-V is an IR for GPU. A codelet can be compiled into multiple executable binaries, or bitstreams.*

The codelet is a powerful entity designed to fully exploit the underlying hardware infrastructure. First, it can reconfigure network topologies by orchestrating network switches and devices to create temporal link connections. Second, it has explicit intra-codelet data movement and scheduling primitives to overlap computation and communication within a codelet. For inter-codelet scheduling, we plan to add a specially defined DAG-codelet. Third, a codelet serves as the smallest logical unit of migration among devices. To migrate a codelet, we launch a previously compiled binary of this codelet onto the target hardware and then migrate states using the generated APIs.

4.3 Disaggregation-Native Design Methodology

As Section 3 points out, disaggregation-native design methodology's code idea is to co-design software with the heterogeneous infrastructure nature. Several prior works have explored building OS, indexes, trees, and databases for disaggregation [42, 43, 48, 50, 58, 63]. Following their insights, we propose the following design principles.

1. **Enable non-transparency and explicit code annotation**. Software should explicitly annotate their data and code to control placement, co-location, and security. This principle trades ease-of-programming for avoiding unwanted overhead. Our proposed IR can transform annotations into explicit scheduling primitives.
2. **Expose failures and enable configurable reliability**. We expect to expose device or network failures to software rather than masking them silently. Consequently, software can customize failure handling logic and choose the right level of reliability (*e.g.*, use erasure-coding rather than three-way replication).
3. **Enable network topology-aware designs**. We expect software to be topology-aware and can customize the network topologies to best fit their data access and computation offloading schemes. For example, distributed ML training could morph the topologies to fit its all-reduce

pattern. Our proposed IR can generate low-level network reconfiguration primitives. This practice leads to reduced network buffering hence better performance.

5 Conclusion

In this paper, we take a giant leap envisioning a fully disaggregated and programmable data center model and propose principled guidelines for data center designers to working towards this model. In particular, we propose a three-layer approach by laying out the device and network design requirements, an MLIR-based compilation flow, and disaggregation-native design principles.

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] David Chisnall, Deepak Garg, Catalin Hritcu, and Mathias Payer. Secure Compilation, Report from Dagstuhl Seminar 21481. 2022. https://nebelwelt.net/files/21DAGSTUHL.pdf.

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

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

[11] 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)*.

[12] 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.

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

[14] Yong Feng, Zhikang Chen, Haoyu Song, Wenquan Xu, Jiahao Li, Zijian Zhang, Tong Yun, Ying Wan, and Bin Liu. Enabling in-situ programmability in network data plane: From architecture to language. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[15] 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.

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

[17] 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)*.

[18] GenZ Consortium. http://genzconsortium.org/.

[19] 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.

[20] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E. Anderson. Backpressure flow control. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[21] 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.

[22] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651*, 2018.

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

[24] 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)*.

[25] Intel. Intel oneAPI. https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html.

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

[27] 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.

[28] 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.

[29] 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)*.

[30] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, 2016.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, 2019.

[36] 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)*.

[37] 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)*, 2020.

[38] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, 2019.

[39] 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.

[40] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. Floem: A programming system for NIC-Accelerated network applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018.

[41] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. Automated smartnic offloading insights for network functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, 2021.

[42] 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.

[43] 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)*.

[44] Yizhou Shan, Will Lin, Ryan Kosta, Arvind Krishnamurthy, and Yiying Zhang. Disaggregating and consolidating network functionalities with supernic. *arXiv preprint arXiv:2109.07744*, 2021.

[45] 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)*.

[46] 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.

[47] 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.

[48] 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.

[49] 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)*.

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

[51] 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)*, 2020.

[52] 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)*, 2020.

[53] 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.

[54] 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.

[55] Jiarong Xing, Kuo-Feng Hsu, Matty Kadosh, Alan Lo, Yonatan Piasetzky, Arvind Krishnamurthy, and Ang Chen. Runtime programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[56] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Hongyi Liu, Matty Kadosh, Alan Lo, Aditya Akella, Thomas Anderson, Arvind Krishnamurthy, TS Eugene Ng, et al. A vision for runtime programmable networks. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, 2021.

[57] Kaiyuan Zhang, Danyang Zhuo, and Arvind Krishnamurthy. Gallium: Automated software middlebox offloading to programmable switches. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, 2020.

[58] 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.

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

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

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

[62] 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.

[63] 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.