AvocadOS: Joint Orchestration of Disaggregated CPUs and FPGAs in the Cloud

Pratyush Patel Katie Lim Johan Vikström

University of Washington

Abstract

In this work we research abstractions for disaggregated FPGAs to support them as first-class compute citizens in the datacenter. In doing so, we determine how CPUs and FPGAs should be presented to application developers to maximize convenience, and how cloud providers should orchestrate these increasingly heterogeneous compute resources.

1 Introduction

Most of the worldwide computation today takes place in datacenters run by cloud providers. Clouds rely on the economy of scale achieved by consolidating computing resources and sharing them among numerous clients [8]. This provides notable benefits to both cloud providers and users. Cloud providers profit by packing user compute and data on shared hardware, thereby maximizing resource utilization at amortized operational costs. In return, cloud users benefit from the abstraction of a large, flexible pool of computing resources that they can use and pay for on demand [5].

By offering flexible abstractions, the cloud computing model has fundamentally shaped how users think about and deploy applications at scale. For example, serverless computing has decoupled the notion of application code from the hardware that it executes on [30]. This is done by virtualizing the pool of datacenter hardware resources and automating system and network configurations, which frees developers from low-level details and allows them to focus on application logic. Developers no longer need to worry about resource elasticity, because they can easily scale up or down provisioned resources depending on their application needs. They can also offload fault tolerance concerns to cloud providers who guarantee highly available services by specifying Service Level Agreements (SLAs) [41] and automatically handle migration upon certain types of failures [14].

These flexible cloud abstractions have been shown to be successful since their inception on CPU-based servers [5]. However, in recent years, cloud providers are increasingly incorporating accelerators such as FPGAs and ASICs into their datacenters [47, 24, 13]. This is driven by hardware and application requirements. On the hardware side, CPU performance has stalled due to the end of Dennard scaling and energy dissipation concerns [54, 27, 28, 18]. Simultaneously on the applications side, modern workloads such as machine learning (ML) and deep learning (DL) continue to scale up [43, 38], with computation needs that can only be effectively met by accelerators [31, 47, 19].

As of today, accelerators are exposed in relatively restricted ways to cloud users. For example, Microsoft Catapult FPGAs are directly attached to the datacenter network, but are kept hidden from cloud users as they are used internally to accelerate cloud services [47, 11, 21]. Amazon and Alibaba provide FPGA-attached virtual machine instances that can be manually programmed via the host CPU, but with reduced access to datacenter services such as the networking infrastructure [2, 3]. Google's TPUs are exposed using a restricted TensorFlow programming interface, which limits their usability for arbitrary user workloads [31, 24, 16].

With accelerators becoming another compute cornerstone next to CPUs [34, 39], we believe that cloud providers need to rethink the accelerator abstractions that are presented to users. We posit that effectively exposing accelerators at scale to cloud users will require flexible abstractions similar to CPU-based clouds upon which large-scale user applications can be easily built and managed.

In this work, we explore and build flexible cloud abstractions to expose network-attached FPGAs as first-class compute citizens in the datacenter alongside CPUs. We choose to focus on FPGAs because they have been adopted by several cloud providers due to their programmability and versatility benefits over ASICs [2, 3, 20, 13]. We envision that users should be able to specify high-level application logic that can transparently use either target platform. That is, FPGAs should be able to replace CPUs for workload compute if appropriate or necessary, thereby reducing developer efforts in deploying efficient applications. In addition, applications should be able to uniformly access all the necessary datacen-

ter resources, and automatically scale or migrate upon load variations and failures, regardless of the target platform.

While existing frameworks already support such operations with CPUs [30, 14], no prior work provides all these abstractions for FPGAs. Therefore, we develop mechanisms to perform automatic resource provisioning, SDN-based network configuration, resource virtualization, autoscaling, and fault tolerance on disaggregated FPGAs. To realize these abstractions, we build a node-level runtime for individual FPGA nodes and incorporate FPGA support in a datacenter-scale compute orchestrator.

Each FPGA node in our system consists of the FPGA accelerator chip, a microcontroller, and local resources such as storage and memory. The cloud provider programs the FPGA with a trusted shell that abstracts low-level hardware details from user applications. Users implement untrusted applications as FPGA roles. The shell provides: **(1)** a uniform interface to all roles to access local node resources such as storage and memory, **(2)** an on-chip TCP networking stack to communicate with and access distributed datacenter resources [37], and **(3)** a multiplexing and isolation framework to safely share the FPGA chip among multiple roles. The control processor is attached to the FPGA chip and periodically communicates with the datacenter-wide compute orchestrator. It is used to reconfigure the FPGA, setup network connections, perform network and system configurations, and for load and health monitoring.

The compute orchestrator is logically centralized and coordinates the execution of all user-visible CPUs and FPGAs in the datacenter. It periodically communicates with CPU servers and control processors on FPGA nodes to obtain a global view of the datacenter system. It provides the building blocks to deploy and manage large-scale user applications. This includes provisioning virtualized CPU and FPGA resources, allocating and scheduling user jobs, setting up network configurations between nodes via SDN, making autoscaling decisions based on monitored load, and handling faults by performing migrations or restarts.

We evaluate the functionality and efficiency of our proposed abstractions by running various microbenchmarks. At the individual node-level, these microbenchmarks show that the shell implementation has a low resource utilization footprint, the TCP stack can communicate between compute nodes at line rate, and that our isolation framework successfully allows multiple users to share the FPGA. At the orchestrator level, our microbenchmarks validate that our provided building blocks are functional at low overheads.

To evaluate the efficacy of our proposed abstractions, we perform two case studies. The first extends the serverless computing paradigm to support both CPU and FPGA-based function backends [30]. The second enables transparently deploying software or hardware-based Network Function Virtualization (NFV) pipelines to monitor the safety of datacenter packets [52]. These case studies show that our abstractions are generalizable, and can be used to support a wide variety of CPU and FPGA applications in datacenters.

We make the following contributions in this paper:

- We propose flexible abstractions to conveniently expose network-attached FPGAs to cloud users.
- We implement these abstractions by building per-node hardware platform for FPGAs and augmenting datacenter-wide compute orchestration to support both CPUs and FPGAs.
- We show that our abstractions can enable new functionalities for network-attached FPGAs in datacenters. We do so by extending serverless computing frameworks to incorporate FPGAs, and implementing hardware-agnostic NFV pipelines.

2 Background

2.1 Modern Cloud Abstractions

Over the years, cloud computing has undergone a shift towards a lighter, fine-grained execution model. This trend is centered around lowering application development complexity by offloading management concerns to the cloud provider. Traditionally, users deployed their long-running applications in virtual machines, and manually monitored resource usage and provisioned more VMs as needed [5]. As applications shifted from monolithic to microservices, lighter-weight containers became a more appealing model due to their ability to scale and their ease of deployment [9]. However, like VMs, containers also required some configuration and resource management from users. This prompted the development of container orchestration frameworks such as Kubernetes [46] and Apache Mesos [29] which provide finer-grained resource sharing and scheduling abstractions, and take orchestration burden off from cloud users.

Following in this trend, a recent cloud abstraction called serverless computing is emerging. Serverless computing is an even simpler execution model where users specify computation in the form of functions, and are not burdened with resource allocation or deployment scaling. This separates workloads from their hardware execution environment, lets the cloud provider manage the scheduling and assignment of functions to compute nodes, and enables fine-grained control of datacenter utilization. This model is supported by prominent cloud providers like Amazon [6], Google [25], and Microsoft [7] as well as many open-source frameworks such as OpenFaaS [44] and Knative [36].

Modern serverless deployments have three components that help support their flexible abstractions on CPU servers: **(1)** an encapsulation framework such as containers that allows interfacing the user code with system resources through a virtualization layer [9], **(2)** a global orchestrator such as Kubernetes for allocating and scheduling computation across

different nodes in the machine [46], and **(3)** a serverless engine that handles load balancing, autoscaling, and fault tolerance [44]. Our goal is to support similar abstractions for datacenter FPGAs.

2.2 Datacenter FPGAs

FPGAs are desirable in cloud settings due to their compute and energy efficiency over general-purpose CPUs, and programmability benefits over specialized ASICs [20].

2.2.1 Cloud Deployments and Orchestration

Microsoft was one of the first cloud providers to deploy FPGAs as a part of their Catapult program. These FPGAs were deployed either as PCIe-attached accelerators that communicate directly with the datacenter network [47], or as bump-in-the-wire networking stacks for their host CPUs [11]. Until recently, both these types of FPGAs were not directly exposed to users as a cloud service. Instead, they were managed and used internally by Microsoft for accelerating Bing PageRank [47], ML inference [21], and network encryption [11]. In 2019, Microsoft deployed user-customizable FPGAs as a part of Azure ML [40], which allow users to deploy a restricted set of DL models on a single Catapult FPGA and use them for low-latency inference serving. Catapult has also implemented an overlay network in FPGAs to obtain direct access to various datacenter resources [51].

In contrast, Amazon, Alibaba and IBM provide virtual machine instances with one or more PCIe-attached FPGAs to cloud users [2, 3, 13]. In this model, the cloud user submits an application in an RTL source code form to the cloud provider. The cloud platform merges the user RTL with a secure shell logic and generates the FPGA bitstream image.

Although there are minor differences in the way FPGA logic is deployed in each of the above cases, at a high level, the FPGA logic is divided into two components, the shell and the role. The shell is a trusted component that provides a uniform access abstraction by implementing interfacing logic to hardware resources such as the NIC, storage, memory, etc. on the same node or across the network. The role contains untrusted or user logic that must use the shell to perform resource access. However, beyond node-level support, existing datacenter FPGAs lack higher-level virtualization and orchestration frameworks that allow cloud users to easily program and manage them at scale, which is the focus of work.

2.2.2 Network Attachment

The above deployments have used various designs to support communication with accelerators. We specifically target the setting where FPGAs are attached directly to an Ethernet network, as in Microsoft's Catapult system [11].

We choose to support network-attached accelerators for several reasons. First, hardware network stacks have lower latency [11], which is important for microservice-style computation. Second, offloading the network stack to FPGA hardware allows CPU cycles to be used for other computation. Previous research [20, 37] has shown that there are total cost of ownership (TCO) benefits in offloading networking operations to hardware rather than burning CPU cores that can be sold to users. This is especially true for supporting high bandwidth networking, which is becoming more and more prevalent in the datacenter [22].

We choose to support communication over an Ethernet network rather than a custom accelerator network, so we can support disaggregated computation models. Additionally, a custom network limits scalability of deployments, because it is expensive to maintain a separate network. Microsoft Catapult v2 moved to attaching its FPGAs directly to the Ethernet network for these reasons [11].

To support this type of design, we need at minimum to have a networking stack to allow communication over Ethernet. For this, we leverage our TCP hardware stack used in previous work [37], with further modifications to fully support integration with an application.

3 Flexible Cloud Compute Abstractions

In this section, we describe the resource abstractions that must be provided by the datacenter compute orchestrator to network-attached CPUs and FPGAs in order to flexibly expose them to cloud users. We then discuss two application use cases that are made possible through our proposed abstractions, namely serverless computing and network function virtualization.

3.1 Heterogeneous Compute Orchestration

Our goal is for FPGAs to effectively supplement CPUs as first-class compute resources in the datacenter. To do so, the datacenter compute orchestrator must provide the following abstractions for virtualizing and managing both CPUs and FPGAs.

Resource Provisioning. The datacenter orchestrator should be able to allocate and deploy computation on either CPUs or FPGAs at a large scale. This requires compute resources to be allocated, configured, and programmed over the network.

Automatic Network Configuration. Newly provisioned compute resources must be made accessible to the user and to the rest of the datacenter hardware by assigning network configurations. Software-defined networking (SDN) controllers must be able to assign IP addresses to newly provisioned compute resources in addition to being able to configure routing tables and firewall settings based on user preferences. Similarly, for deallocated or failed resources, SDN controllers should be able to clean up prior network configurations to avoid requests hitting dead nodes.

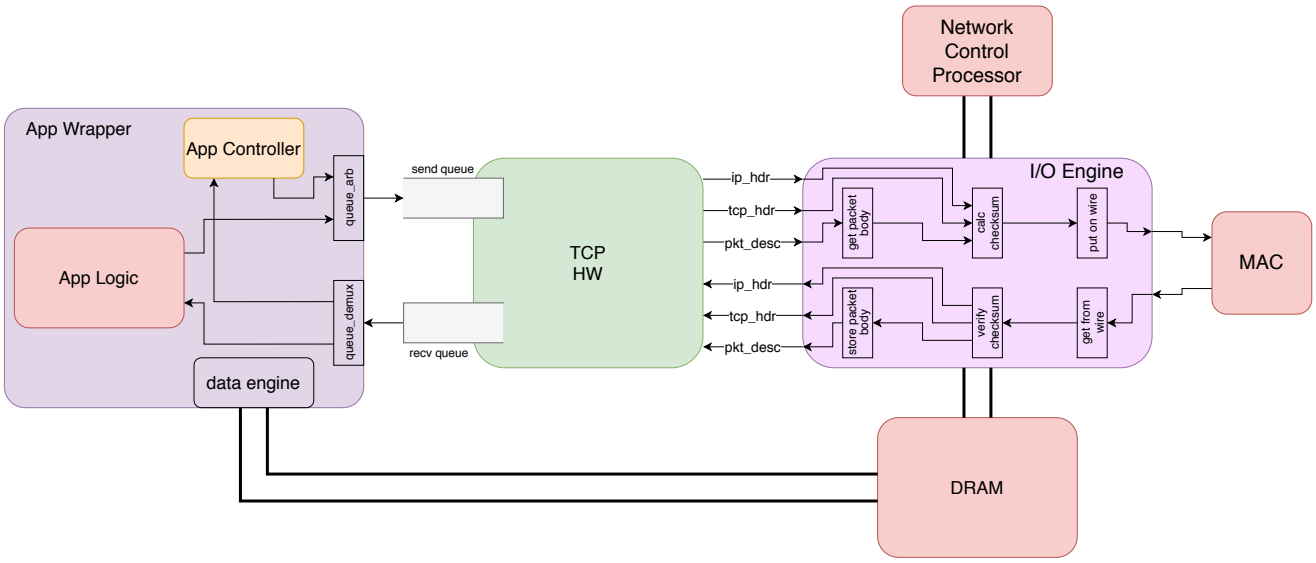

Figure 1: Overview of an FPGA node in our system running a single user application.

Autoscaling and Load Balancing. Each compute resource should expose an interface to the user to implement application-defined load monitoring, based upon which the resource can be scaled up or down. Replicated resources must be transparently load balanced to avoid overutilization of a single resource.

Migration. Each compute resource should expose interfaces to the user to specify application-specific checkpointing logic that can save computational state and restart it on a different node of the same type if need be. The orchestrator should also make sure that network configurations are replicated on the newly migrated node.

Fault Tolerance. Each compute resource should send out periodic heartbeats to verify that the resource is operational. In case of failures detected by the lack of timely heartbeats, the compute orchestrator should step in to resolve the issue, by notifying the user about the failure, cleaning up old resource configurations, and possibly migrating existing state to a newly allocated compute resource.

3.2 Use Cases

In this section, we describe two powerful use cases for datacenter CPUs and FPGAs that are enabled by the aforementioned resource abstractions.

3.2.1 Serverless Computing

We first envision a severless computing platform that can support stateless CPU and FPGA functions [30, 7, 6, 25]. This use case is motivated by the observation that developers primarily care about running their specified application logic, while avoiding low-level hardware details such as the type of hardware or the number of instances that the computation runs on [35]. Putting CPUs and FPGAs together into composable serverless pipelines would tremendously benefit such developers as it development burden and enables a pay-on-demand model for FPGA hardware. It also benefits the datacenter operator who can now perform hardware provisioning, network configuration and function orchestration in a way so as to maximize resource utilization.

In this new framework, developers can pipeline functions together using a high-level programming framework to form an application. To do so, they specify relevant function logic for both CPUs and FPGAs, or choose from a library of existing CPU and FPGA implementations of popular functions offered by the cloud provider. Functions in a pipeline are automatically configured and deployed by the serverless framework upon function invocation. The framework manages the runtime behaviour of the system under varying load and failures by transparently switching to FPGA-based deployments if available, and autoscaling if necessary.

3.2.2 Network Function Virtualization

The second use case for datacenter CPUs and FPGAs involves network function virtualization (NFV). Network functions implement some computation on network traffic, typically to improve performance or security (e.g., via deep packet inspection). They can be strung together to create distributed chains of computation. Network functions are often run on fixed function, dedicated hardware, but this creates management and scaling issues [50, 42], so NFV moves this functionality into general-purpose virtual machines to address management and scaling issues. However, packet processing performance is highly important for network functions [42], as they typi-

cally require line rate processing.

A more flexible deployment of cloud FPGAs would keep network functions implemented in hardware for performance at high bandwidths, but use our proposed compute resource abstractions to provide better virtualization to the hardware network functions to address the original issues of management and scaling. In such a system, each virtual network function (VNF) can be deployed on either on a CPU or on an FPGA. Depending on task complexity, several network function modules can be chained together into pipelines. Larger bandwidths and network traffic variation can be handled by autoscaling, thereby regulating NFV performance on both CPUs and custom hardware while minimizing manual developer configuration. Finally, faults are handled transparently which minimizes network downtime.

4 System Design for Disaggregated FPGAs

To support CPU-like cloud abstractions on disaggregated FPGAs, we develop two components. The first is a per-node FPGA hardware platform which provides a uniform interface to user applications and contains infrastructure to configure and manage FPGAs remotely. The second is the datacenter-wide compute orchestrator that can jointly manage CPU, FPGA and networking resources in the datacenter. We describe both these components in detail and explain how they help support our proposed resource abstractions.

4.1 FPGA Node Hardware Platform

A diagram of the hardware platform on a node in our system is shown in Figure 1. Each node contains an FPGA chip, a microcontroller, and optionally local resources such as storage and memory. We describe these components below.

4.1.1 FPGA Shell and Roles

Similar to existing datacenter FPGA deployments [47, 3], the FPGA platform consists of a cloud provider-trusted shell that abstracts low-level hardware details from user applications, and one or more roles, which are untrusted user applications that run on the FPGA. We reuse parts of shells built in prior work [57, 33].

The FPGA shell provides two key features. First, it provides a uniform resource abstraction to all roles to local and remote resources. This is done by implementing on-chip interfaces to the control processor, memory, storage and the network.

For storage device access, we implement the PCIe interface over which FPGA roles can read or write disk blocks. For networking, we implement an on-chip TCP/IP stack that can communicate over the datacenter network (described in Section 4.1.3).

Second, the shell provides a multiplexing and isolation framework to safely share the FPGA chip among multiple roles. We support both spatial and temporal multiplexing of roles on the FPGA chip [57, 33]. While spatial multiplexing is achieved by the FPGA synthesizing tool, temporal multiplexing requires application-specific logic to checkpoint and restore the application state, which we either provide or permit the developer to implement. The isolation stack also provides a virtualized resource access interface that is replicated for each role and multiplexes these on the actual resource interface implemented on the shell. This prevents interference between roles attempting to access the same resource at the same time.

4.1.2 Control Processor

The control processor is a low-resource CPU attached to the FPGA chip which periodically coordinates with the datacenter-wide compute orchestrator to perform a variety of management operations. It provides FPGA provisioning by obtaining FPGA bitstreams from the orchestrator and uses them to reconfigure a part of or the entire FPGA chip. It also performs local network and system configurations in case FPGA roles must be migrated or restarted. Finally, the control processor is also responsible for load and health monitoring of the FPGA node. It coordinates with on-chip load and health monitors present in the FPGA roles and aggregates them into a hearbeat message that is sent to the datacenter-wide compute orchestrator.

4.1.3 TCP/IP Networking Stack

To support network access on FPGA nodes, we build upon and integrate a TCP/IP stack from our prior work [37].

On the application side, the TCP stack exposes queues of packet payload descriptors that the application can write into using its flow ID. Each descriptor contains two fields, `packet_payload_addr` and `packet_payload_len`, that specify the address and length of the associated packet payload. Each application has a send queue, which it writes packet payload descriptors into, and a read queue, which it reads packet payload descriptors from. On the network side, the TCP block outputs TCP and IP headers and the associated packet payload descriptor and takes as input TCP and IP headers and the packet payload descriptor.

The I/O Engine is primarily responsible for directly interfacing with the MAC. On the receive side, it reads a packet from the MAC and parses the headers to separate the TCP and IP headers from the payload. It verifies the checksum and then copies the payload to DRAM, writing it to a preconfigured region of memory for the FPGA role and creating an associated packet payload descriptor. It passes the packet payload descriptor and the TCP and IP headers to the TCP block, which forwards it to the application. On the send side, the I/O engine takes TCP and IP headers and a packet payload descriptor from the TCP block. It uses the packet payload

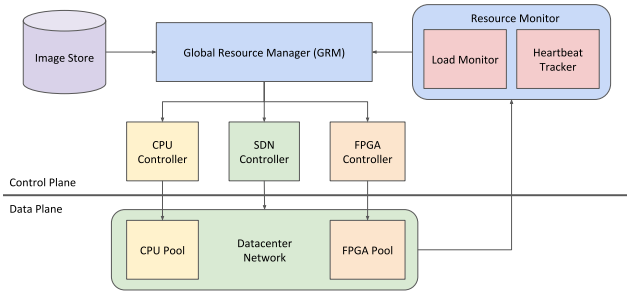

Figure 2: Overview of the compute orchestrator.

descriptor to retrieve the packet payload. It then calculates the checksum and outputs the whole packet to the MAC. The I/O engine is also responsible for forwarding TCP setup and teardown packets to the control processor managing the node.

4.2 Datacenter-Wide Compute Orchestration

The compute orchestrator is logically centralized and coordinates the execution of all user-visible CPUs and FPGAs in the datacenter. It forms the control plane of the datacenter resources and is responsible for appropriately configuring resources in the dataplane (i.e., the CPU and FPGA pools, and the datacenter network). Figure 2 shows a high-level diagram of the components of the compute orchestrator, which include the global resource manager (GRM), various per-resource controllers, a resource monitor and a global image store for CPU container and FPGA images.

The GRM is responsible for resource provisioning, affecting autoscaling decisions, and handling migrations upon failures. It specifies provisioning decisions to the CPU and FPGA controllers which carry them out on actual CPU and FPGA resources. The GRM also manages the SDN controller which assigns IP addresses to nodes and sets up routing tables and firewalls between heterogeneous clusters of user nodes. The resource monitor is responsible for tracking heartbeats and monitoring load across the datacenter by communicating with per-node runtimes. The global image store is a large distributed store that contains all the cloud-provided and user-submitted container images and FPGA bitstreams that can be deployed on datacenter hardware. These components interoperate to provide the following abstractions to virtualize CPU and FPGA compute resources in the cloud.

4.2.1 Resource Provisioning

The main component responsible for provisioning the pool of CPU and FPGA resources in the datacenter are the GRM and the per-resource controllers. When a workload is run for the first time, the GRM notifies the controllers to provision the corresponding target hardware platform (either CPU or FPGA) based on resource availability and user preferences. Subsequently, the GRM informs the SDN controller to set up network configurations for the newly provisioned node. To do so, the SDN controller assigns an IP address to the node, and sets up routing tables for the set of compute resources that the node communicates with. After finishing systems and network configurations, the GRM obtains the workload image from the central image store and deploys it on the provisioned hardware platform and returns the IP address of the provisioned node to the datacenter user.

4.2.2 Autoscaling

The initial provisioning may later be modified by the GRM in response to variations in load. Application load is continuously monitored at a per-node level and reported to the datacenter-wide load monitor. As explained in Section 2, CPU compute is already provisioned in reaction to load by modern orchestration frameworks such as Kubernetes [46] and OpenFaaS [44]. Similarly, for our abstractions to function seamlessly across hardware platforms, the FPGA provisioned under our framework must also dynamically react to load. As we cannot modify the amount of resources allocated to FPGAs in a granular manner similar to CPUs, we choose to perform FPGA autoscaling at the granularity of FPGA roles. For example, if a user is frequently using the functionality provided by specific bitfile, the framework should scale up the number of FPGAs programmed with that bitfile to handle the load. This means that the GRM may increase or decrease copies of the deployed FPGA roles. A load balancer, which is allocated on a separate CPU node, is then used to distribute task requests among replicated compute instances. Provisioning of autoscaled instances is handled in a similar manner as in Section 4.2.1.

4.2.3 Automatic Network Configuration

The SDN controller is responsible for configuring the datacenter network. There are three scenarios where the SDN controller must modify network configurations. Upon resource provisioning, the SDN controller assigns an appropriate IP address to the new node, and sets up routing tables leading to and originating at the node. Depending on the user's preferences, the assigned IP may be a part of a private virtualized network, or public to the internet. The IP address and associated routes are cleared up upon resource deallocation. Finally, when a failure is detected, the SDN controller immediately reprograms the network to prevent traffic from going to the failed node until a new one is brought up in its place. Alternatively, this traffic can be sent to previously replicated instances of the failed node if they exist.

4.2.4 Migration

The GRM handles migration of computation between two identical compute resources. To guarantee transparency to the user, the GRM ensures that the same network configuration as before is made available on the node that is migrated to, in addition to transferring application state. As application state on FPGAs is difficult to capture without explicit user interfaces, we rely on the user to provide checkpointing logic for FPGAs. In case of CPUs, we additionally provide the option of using existing checkpointing frameworks [15, 4]. The checkpointed state is saved in the image store, and is forwarded to a newly provisioned node by the GRM upon migration.

4.2.5 Fault Tolerance

The heartbeat tracker module in the resource monitor periodically receives heartbeats from each deployed application. On FPGA nodes, such heartbeats are sent by the control processor, which may obtain it from FPGA roles. In case a heartbeat is not received within a certain duration, the tracker times out and corresponding node is considered to have failed. The node is also considered failed upon explicit failure notification by the node-level control processor. In these cases, the heartbeat tracker notifies the GRM about the failure, which in turn informs the SDN controller to unassign old network configurations, and then attempts to restart or migrate the most recent application state on a different node after re-provisioning it with the same image and network configuration as before.

5 Evaluation

We plan to deploy our framework on Microsoft Catapult FPGAs run on an internal cluster [47, 11, 21].

Our per-FPGA node infrastructure is built on top of Catapult's shell, which already provides interfacing with various datacenter resources [51]. However, Catapult FPGAs do not have low-resource control processors, so we expect to provide the same functionality using the FPGA's host CPUs.

Each component of the compute orchestrator will be deployed on separate CPU nodes in our system via OpenStack [45], or Microsoft's equivalent cluster management framework.

5.1 Short Term

We would like to answer the following questions in the short term:

1. Can we get Microsoft Catapult to work?
2. What is the resource utilization of our shell when basic accelerator support is implemented?
3. What are basic bandwidth/latency numbers for the networking components of our shell?

5.2 Long Term

We plan to perform the following benchmarks:

1. How much FPGA logic does the full shell consume on Catapult FPGAs? What is the FPGA clock frequency with a shell-only design?
2. What is the application runtime overhead of the shell's isolation framework? That is, does running a program alone perform similarly as multiplexed accelerators?
3. What is the performance (latency distribution/bandwidth) of applications running in our framework versus their CPU-based "equivalents"?
4. How does the system behave under varying load? Does autoscaling work?
5. How does performance degrade when there are failures?

5.3 Case Studies

We examine the general applicability of our cloud compute resource abstractions by performing the following case studies.

5.3.1 Serverless Computing

In this case study, we extend the typically CPU-only severless computing paradigm to support both CPU and FPGA functions [30, 7, 6, 25]. That is, our serverless framework allows using either hardware resource to run user-specified cloud functions. Applications consist of pipelines of CPU containers and FPGA application roles, each of which performs the task of a single cloud function. Each function has well-defined input-output specifications (e.g., a function to decode a video or to classify an image), and can be chained together to form larger applications. Such pipelines have been shown to be a good fit to perform video, image, and sound processing [32], which we evaluate in this case study.

During steady state, we expect our hardware-enabled serverless pipelines to have better latency and throughput than its CPU-only counterpart. This will be partially due to enabling the use of accelerators for application logic and partially due to our shell streamlining the management of the accelerators. When adding capacity, we expect our performance to be similar to a CPU-based system. It has been shown that serverless computing performance depends on whether a function has been run recently or if a new container must be started for the function [49], called a cold start. However, our framework must similarly deal with FPGA reprogramming latencies and initializing routing tables.

5.3.2 Network Function Virtualization

Our second case study implements network function virtualization on our resource orchestration framework framework,

and will specifically focus on supporting network functions for deep packet inspection. Packet analysis hardware will be deployed in a node as a CPU or as an FPGA accelerator. Depending on the complexity of the analysis, several packet analysis modules can be chained together, including both traditional CPU-based network functions as well as specialized FPGA functions. We will show that our framework will be able to autoscale the deployment in reaction to traffic load to maintain line rate.

6 Related Work

In addition to the real-world FPGA deployments described in Section 2, several other prior works provide varying types and degrees of cloud abstractions for FPGAs. Most prior works use OpenStack [45] to implement their systems.

Chen et al. use OpenStack to virtualize FPGAs [12]. They couple PCIe-attached FPGAs with a KVM-based software hypervisor, and establish coordination between OpenStack and the software hypervisor to manage the FPGAs. However, they do not consider direct network attachment, which is essential to support large-scale applications, especially with increasing datacenter bandwidths [37].

Byma et al. propose to deploy network-attached FPGAs as OpenStack resources [10]. In their case, the node-level hypervisor is programmed into hardware and communicates to the OpenStack controller over the network. The FPGA application region is split into four smaller regions programmed via partial reconfiguration, which allows multiple users to share a single FPGA device. However, this infrastructure is only built for a 1 Gbps network port and only targets node-level support for cloud FPGAs.

Tarafdar et al. develop a framework to create FPGA clusters in an OpenStack-managed cloud [53]. Their framework generates OpenStack calls needed to reserve the compute devices, configures the network connections (with MAC addresses), and creates a ready-to-use cluster network device that can interact with any other network device in the data center. However, their framework requires users to provide a cluster description and FPGA mappings, which prevents true virtualization of FPGA resources. Our work enables cloud providers to automatically allocate and configure cloud FPGAs alongside CPUs, in addition to other orchestration abstractions, making the underlying hardware transparent to cloud users.

Building upon their previous work, Tarafdar et al. also present a framework for creating pipelines of heterogeneous virtualized network functions, using either CPUs or FPGAs [52]. They build a service chain scheduler and extend OpenStack's SDN controller to allocate and connect CPU and FPGA resources, allowing for incremental hardware deployment of virtual network functions. However, they do not examine other cloud abstractions such as autoscaling and fault tolerance.

Eskandari et al. implement modular communication layers (including MPI) on both CPUs and FPGAs to achieve a common communication interface for tasks that can run either hardware platform [17]. This is orthogonal to our work, but we can adopt it for greater flexibility in deploying our FPGA applications.

IBM's research on cloudFPGAs shares many goals with our work. They have experimented with network-attached FPGA prototypes that target hyperscale datacenters [56, 1, 55]. These FPGAs are provisioned via OpenStack, can be configured via OpenStack's SDN controller, and can be operated by cloud users who obtain an IP address to the target FPGA.

IBM recently extended this system with partial reconfiguration over the network to enable direct programming of FPGA nodes [48]. However, they still embed a tiny control processor alongside each FPGA which is used for low-level management such as resetting the FPGA, setting IP addresses, or to program the on-chip flash memory. Their FPGAs expose a RESTful API that supports performing remote bitfile-based FPGA reconfiguration, obtaining node status, and configuring routing information in order manage FPGAs. They run an MPI workload to show FPGA-CPU interoperability, albeit with some modifications. All in all, the abstractions they provide are very similar to our work. However, their FPGA networking stack can only support 10Gbps TCP/IP networking. Furthermore, they do not provide transparent autoscaling or fault tolerance monitoring, which we believe are important abstractions to fully virtualize datacenter FPGAs.

Finally, resource orchestration mechanisms are also available for other accelerators such as GPUs and TPUs. For example, GPUs are typically made available over the PCIe bus similar to Amazon F1 FPGAs [23]. However, this does not disaggregate GPUs as individual compute units on the datacenter network, which is a key focus of our work. On the Google Cloud Platform, CPUs, GPUs and TPUs can be programmed using the TensorFlow framework [23], which allows composing multiple CPUs and TPUs to run deep learning inference and training models. These resources can be scaled up or down based on load via the Google Kubernetes Engine [26]. However, unlike our framework, TensorFlow is a relatively restricted programming interface which is specific to deep learning applications. Our work targets FPGAs, enables easy interoperatbility between general-purpose CPUs and FPGAs, supports multi-tenancy, and is amenable to a wide variety of workload types.

7 Conclusion and Future Work

In this work, we proposed flexible resource abstractions to treat network-attached FPGAs as first-class compute citizens in the datacenter alongside CPUs. Our next steps are to implement these abstractions on Microsoft Catapult FPGAs and to evaluate their efficacy by developing a serverless framework and composing network function virtualization pipelines.

References

[1] Francois Abel, Jagath Weerasinghe, Christoph Hagleitner, Beat Weiss, and Stephan Paredes. "An FPGA platform for hyperscalers". In: *Symposium on High-Performance Interconnects (HOTI)*. 2017 (p. 8).

[2] Alibaba. *Deep Dive into Alibaba Cloud F3 FPGA as a Service Instances*. Accessed: 2020-01-29. URL: https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057 (pp. 1, 3).

[3] Amazon. *Amazon EC2 F1 Instances*. Accessed: 2019-10-11. URL: https://aws.amazon.com/ec2/instance-types/f1/ (pp. 1, 3, 5).

[4] Jason Ansel, Kapil Arya, and Gene Cooperman. "DMTCP: Transparent checkpointing for cluster computations and the desktop". In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. 2009 (p. 7).

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. "A view of cloud computing". In: *Communications of the ACM* (2010) (pp. 1, 2).

[6] "AWS Lambda". In: (). https://aws.amazon.com/lambda/ (pp. 2, 4, 7).

[7] "Azure Functions". In: (). https://azure.microsoft.com/en-us/services/functions/ (pp. 2, 4, 7).

[8] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. "The datacenter as a computer: Designing warehouse-scale machines". In: *Synthesis Lectures on Computer Architecture* 13.3 (2018), pp. i–189 (p. 1).

[9] David Bernstein. "Containers and cloud: From LXC to Docker to Kubernetes". In: *IEEE Cloud Computing* (2014) (p. 2).

[10] Stuart Byma, J Gregory Steffan, Hadi Bannazadeh, Alberto Leon Garcia, and Paul Chow. "FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack". In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2014 (p. 8).

[11] Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. "A Cloud-Scale Acceleration Architecture". In: *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Oct. 2016 (pp. 1, 3, 7).

[12] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. "Enabling FPGAs in the cloud". In: *Conference on Computing Frontiers*. 2014 (p. 8).

[13] IBM Research – China. *OpenPOWER Cloud – Accelerating Cloud Computing*. 2019. URL: https://www.research.ibm.com/labs/china/supervessel.html (pp. 1, 3).

[14] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. "Live migration of virtual machines". In: *Symposium on Networked Systems Design and Implementation (NSDI)*. 2005 (pp. 1, 2).

[15] *CRIU: Checkpoint-Restore in Userspace*. Accessed: 2020-02-05. URL: https://www.criu.org/Main_Page (p. 7).

[16] David Patterson. *Domain Specific Architectures for Deep Neural Networks: Three Generations of Tensor Processing Units (TPUs)*. Allen School Distinguished Lecture, https://www.youtube.com/watch?v=VCScWh966u4. 2019 (p. 1).

[17] Nariman Eskandari, Naif Tarafdar, Daniel Ly-Ma, and Paul Chow. "A modular heterogeneous stack for deploying FPGAs and CPUs in the data center". In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2019, pp. 262–271 (p. 8).

[18] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. "Power limitations and dark silicon challenge the future of multicore". In: *ACM Transactions on Computer Systems (TOCS)* 30.3 (2012), pp. 1–27 (p. 1).

[19] Facebook. *Accelerating Facebook's infrastructure with application-specific hardware*. Accessed: 2019-10-11. URL: https://engineering.fb.com/data-center-engineering/accelerating-infrastructure/ (p. 1).

[20] Daniel Firestone, Andrew Putnam, Hari Angepat, Derek Chiou, Adrian Caulfield, Eric Chung, Matt Humphrey, Kalin Ovtcharov, Jitu Padhye, Doug Burger, Dave Maltz, Albert Greenberg, Sambhrama Mundkur, Alireza Dabagh, Mike Andrewartha, Vivek Bhanu, Harish Kumar Chandrappa, Somesh Chaturmohta, Jack Lavier, Norman Lam, Fengfen Liu, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Kushagra Vaid, and David A. Maltz. "Azure Accelerated Networking: SmartNICs in the Public Cloud". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Apr. 2018 (pp. 1, 3).

[21] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steve Reinhardt, Adrian Caulfield, Eric Chung, and Doug Burger. "A Configurable Cloud-Scale DNN Processor for Real-Time AI". In: *Proceedings of the 45th International Symposium on Computer Architecture, 2018*. ACM, June 2018 (pp. 1, 3, 7).

[22] FS. *MTP/MPO Cabling System: A Panacea for Data Center*. Accessed: 2020-02-24. URL: https://community.fs.com/blog/mtpmpo-cabling-system-a-panacea-for-data-center.html (p. 3).

[23] Google. *Cloud GPUs*. Accessed: 2020-02-06. URL: https://cloud.google.com/gpu (p. 8).

[24] Google. *Google's scalable supercomputers for machine learning, Cloud TPU Pods, are now publicly available in beta*. Accessed: 2019-10-11 (p. 1).

[25] "Google Cloud Functions". In: (). https://cloud.google.com/functions/ (pp. 2, 4, 7).

[26] *Google Kubernetes Engine TPUs*. Accessed: 2020-02-05. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/tpus (p. 8).

[27] Nathan Goulding, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Jonathan Babb, Michael B Taylor, and Steven Swanson. "GreenDroid: A mobile application processor for a future of dark silicon". In: *2010 IEEE Hot Chips 22 Symposium (HCS)*. IEEE. 2010, pp. 1–39 (p. 1).

[28] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. "Toward dark silicon in servers". In: *IEEE Micro* 31.4 (2011), pp. 6–15 (p. 1).

[29] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center". In: *Networked Systems Design and Implementation (NSDI)*. 2011 (p. 2).

[30] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. "Cloud programming simplified: a berkeley view on serverless computing". In: *arXiv preprint arXiv:1902.03383* (2019) (pp. 1, 2, 4, 7).

[31] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, and et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *SIGARCH Computer Architecture News* 45.2 (June 2017), pp. 1–12 (p. 1).

[32] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. "GrandSLAm: Guaranteeing SLAs for jobs in microservices execution frameworks". In: *European Conference on Computer Systems (EuroSys)*. 2019 (p. 7).

[33] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J Rossbach. "Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS". In: *Operating Systems Design and Implementation (OSDI)*. 2018 (p. 5).

[34] Moein Khazraee, Lu Zhang, Luis Vega, and Michael Bedford Taylor. "Moonwalk: NRE Optimization in ASIC Clouds". In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: ACM, 2017, pp. 511–526 (p. 1).

[35] Jaewook Kim, Tae Joon Jun, Daeyoun Kang, Dohyeun Kim, and Daeyoung Kim. "GPU Enabled Serverless Computing Framework". In: *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. 2018 (p. 4).

[36] "Knative". In: (). https://github.com/knative/ (p. 2).

[37] Katie Lim, Pratyush Patel, Tom Anderson, and Michael B. Taylor. *A Quantitative Analysis of Transport-LevelNetworking for Disaggregated Accelerators*. Tech. rep. University of Washington, Feb. 2020 (pp. 2, 3, 5, 8).

[38] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. "Parameter hub: a rack-scale parameter server for distributed deep neural network training". In: *Proceedings of the ACM Symposium on Cloud Computing*. 2018, pp. 41–54 (p. 1).

[39] Ikuo Magaki, Moein Khazraee, Luis Vega, and Michael Taylor. "ASIC Clouds: Specializing the Datacenter". In: *International Symposium on Computer Architecture (ISCA)*. 2016 (p. 1).

[40] Microsoft. *FPGA Web Service: Deploy Models on FPGAs*. Accessed: 2020-01-29. URL: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service (p. 3).

[41] Jeffrey C. Mogul and John Wilkes. "Nines are Not Enough: Meaningful Metrics for Clouds". In: *Workshop on Hot Topics in Operating Systems (HotOS)*. 2019 (p. 1).

[42] Leonhard Nobach, Oliver Hohlfeld, and David Hausheer. “New Kid on the Block: Network Functions Visualization: From Big Boxes to Carrier Clouds”. In: *SIGCOMM Comput. Commun. Rev.* 46.3 (July 2018) (p. 4).

[43] OpenAI. *AI and Compute*. Accessed: 2020-1-17. URL: https://openai.com/blog/ai-and-compute/ (p. 1).

[44] “OpenFaaS”. In: (). https://github.com/openfaas/faas (pp. 2, 3, 6).

[45] “OpenStack”. In: (). https://www.openstack.org/ (pp. 7, 8).

[46] “Production-Grade Container Orchestration”. In: (). https://kubernetes.io/ (pp. 2, 3, 6).

[47] Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, Eric Peterson, Aaron Smith, Jason Thong, Phillip Yi Xiao, Doug Burger, Jim Larus, Gopi Prashanth Gopal, and Simon Pope. “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services”. In: *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*. IEEE, June 2014, pp. 13–24 (pp. 1, 3, 5, 7).

[48] Burkhard Ringlein, Francois Abel, Alexander Ditter, Beat Weiss, Christoph Hagleitner, and Dietmar Fey. “System architecture for network-attached FPGAs in the Cloud using partial reconfiguration”. In: *International Conference on Field Programmable Logic and Applications (FPL)*. 2019 (p. 8).

[49] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. “Architectural Implications of Function-as-a-Service Computing”. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO ’52. 2019. ISBN: 9781450369381 (p. 7).

[50] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. “Making Middleboxes Someone Else’s Problem: Network Processing as a Cloud Service”. In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM ’12. 2012. ISBN: 9781450314190 (p. 4).

[51] Ran Shu, Peng Cheng, Guo Chen, Zhiyuan Guo, Lei Qu, Yongqiang Xiong, Derek Chiou, and Thomas Moscibroda. “Direct Universal Access: Making Data Center Resources Available to FPGA”. In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 127–140. ISBN: 978-1-931971-49-2 (pp. 3, 7).

[52] Naif Tarafdar, Thomas Lin, Nariman Eskandari, David Lion, Alberto Leon-Garcia, and Paul Chow. “Heterogeneous virtualized network function framework for the data center”. In: *International Conference on Field Programmable Logic and Applications (FPL)*. 2017, pp. 1–8 (pp. 2, 8).

[53] Naif Tarafdar, Thomas Lin, Eric Fukuda, Hadi Bannazadeh, Alberto Leon-Garcia, and Paul Chow. “Enabling flexible network FPGA clusters in a heterogeneous cloud data center”. In: *International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2017, pp. 237–246 (p. 8).

[54] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. “Conservation cores: reducing the energy of mature computations”. In: *ACM SIGPLAN Notices* 45.3 (2010), pp. 205–218 (p. 1).

[55] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner, and Andreas Herkersdorf. “Disaggregated fpgas: Network performance comparison against bare-metal servers, virtual machines and linux containers”. In: *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2016 (p. 8).

[56] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner, and Andreas Herkersdorf. “Enabling FPGAs in hyperscale data centers”. In: *International Conference on Cloud and Big Data Computing (CBDCom)*. 2015 (p. 8).

[57] Jiansong Zhang, Yongqiang Xiong, Ningyi Xu, Ran Shu, Bojie Li, Peng Cheng, Guo Chen, and Thomas Moscibroda. “The Feniks FPGA operating system for cloud computing”. In: *Proceedings of the 8th Asia-Pacific Workshop on Systems*. 2017 (p. 5).