A Cloud-Scale Acceleration Architecture

Adrian M. Caulfield Eric S. 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 Doug Burger

Microsoft Corporation

***Abstract*—Hyperscale datacenter providers have struggled to balance the growing need for specialized hardware (efficiency) with the economic benefits of homogeneity (manageability). In this paper we propose a new cloud architecture that uses reconfigurable logic to accelerate both network plane functions and applications. This Configurable Cloud architecture places a layer of reconfigurable logic (FPGAs) between the network switches and the servers, enabling network flows to be programmably transformed at line rate, enabling acceleration of local applications running on the server, and enabling the FPGAs to communicate directly, at datacenter scale, to harvest remote FPGAs unused by their local servers. We deployed this design over a production server bed, and show how it can be used for both service acceleration (Web search ranking) and network acceleration (encryption of data in transit at high-speeds). This architecture is much more scalable than prior work which used secondary rack-scale networks for inter-FPGA communication. By coupling to the network plane, direct FPGA-to-FPGA messages can be achieved at comparable latency to previous work, without the secondary network. Additionally, the scale of direct inter-FPGA messaging is much larger. The average round-trip latencies observed in our measurements among 24, 1000, and 250,000 machines are under 3, 9, and 20 microseconds, respectively. The Configurable Cloud architecture has been deployed at hyperscale in Microsoft's production datacenters worldwide.**

I. Introduction

Modern hyperscale datacenters have made huge strides with improvements in networking, virtualization, energy efficiency, and infrastructure management, but still have the same basic structure as they have for years: individual servers with multicore CPUs, DRAM, and local storage, connected by the NIC through Ethernet switches to other servers. At hyperscale (hundreds of thousands to millions of servers), there are significant benefits to maximizing homogeneity; workloads can be migrated fungibly across the infrastructure, and management is simplified, reducing costs and configuration errors.

Both the slowdown in CPU scaling and the ending of Moore's Law have resulted in a growing need for hardware specialization to increase performance and efficiency. However, placing specialized accelerators in a subset of a hyperscale infrastructure's servers reduces the highly desirable homogeneity. The question is mostly one of economics: whether it is cost-effective to deploy an accelerator in every new server, whether it is better to specialize a subset of an infrastructure's new servers and maintain an ever-growing number of configurations, or whether it is most cost-effective to do neither. Any specialized accelerator must be compatible with the target workloads through its deployment lifetime (e.g. six years: two years to design and deploy the accelerator and four years of server deployment lifetime). This requirement is a challenge given both the diversity of cloud workloads and the rapid rate at which they change (weekly or monthly). It is thus highly desirable that accelerators incorporated into hyperscale servers be programmable, the two most common examples being FPGAs and GPUs.

978-1-5090-3508-3/16/$31.00 © 2016 IEEE

Both GPUs and FPGAs have been deployed in datacenter infrastructure at reasonable scale without direct connectivity between accelerators [1], [2], [3]. Our recent publication described a medium-scale FPGA deployment in a production datacenter to accelerate Bing web search ranking using multiple directly-connected accelerators [4]. That design consisted of a rack-scale fabric of 48 FPGAs connected by a secondary network. While effective at accelerating search ranking, our first architecture had several significant limitations:

- The secondary network (a 6x8 torus) required expensive and complex cabling, and required awareness of the physical location of machines.
- Failure handling of the torus required complex re-routing of traffic to neighboring nodes, causing both performance loss and isolation of nodes under certain failure patterns.
- The number of FPGAs that could communicate directly, without going through software, was limited to a single rack (i.e. 48 nodes).
- The fabric was a limited-scale "bolt on" accelerator, which could accelerate applications but offered little for enhancing the datacenter infrastructure, such as networking and storage flows.

In this paper, we describe a new cloud-scale, FPGA-based acceleration architecture, which we call the *Configurable Cloud*, which eliminates all of the limitations listed above with a single design. This architecture has been — and is being — deployed in the majority of new servers in Microsoft's production datacenters across more than 15 countries and 5 continents. A Configurable Cloud allows the datapath of cloud communication to be accelerated with programmable hardware. This datapath can include networking flows, storage flows, security operations, and distributed (multi-FPGA) applications.

The key difference over previous work is that the accelera-

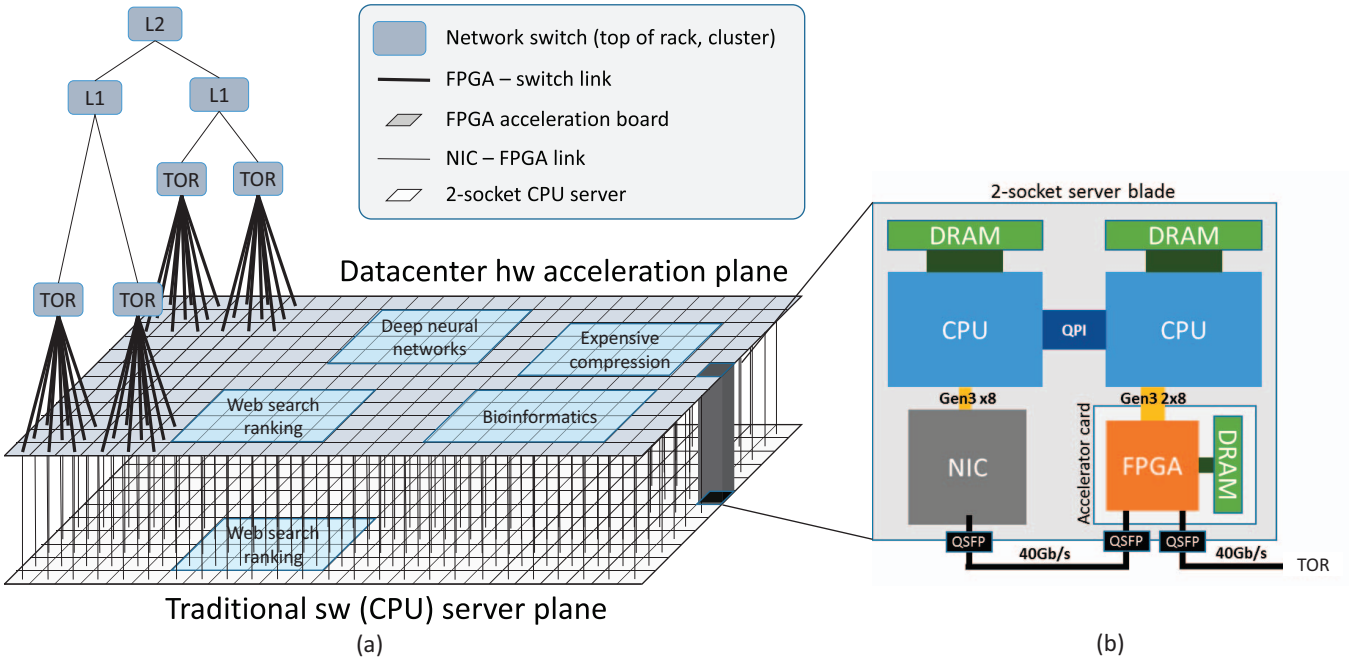

Fig. 1. (a) Decoupled Programmable Hardware Plane, (b) Server + FPGA schematic.

tion hardware is tightly coupled with the datacenter network—placing a layer of FPGAs between the servers' NICs and the Ethernet network switches. Figure 1b shows how the accelerator fits into a host server. All network traffic is routed through the FPGA, allowing it to accelerate high-bandwidth network flows. An independent PCIe connection to the host CPUs is also provided, allowing the FPGA to be used as a local compute accelerator. The standard network switch and topology removes the impact of failures on neighboring servers, removes the need for non-standard cabling, and eliminates the need to track the physical location of machines in each rack.

While placing FPGAs as a network-side "bump-in-the-wire" solves many of the shortcomings of the torus topology, much more is possible. By enabling the FPGAs to generate and consume their own networking packets independent of the hosts, each and every FPGA in the datacenter can reach every other one (at a scale of hundreds of thousands) in a small number of microseconds, without any intervening software. This capability allows hosts to use remote FPGAs for acceleration with low latency, improving the economics of the accelerator deployment, as hosts running services that do not use their local FPGAs can donate them to a global pool and extract value which would otherwise be stranded. Moreover, this design choice essentially turns the distributed FPGA resources into an independent computer in the datacenter, at the same scale as the servers, that physically shares the network wires with software. Figure 1a shows a logical view of this plane of computation.

This model offers significant flexibility. From the local perspective, the FPGA is used as a compute or a network accelerator. From the global perspective, the FPGAs can be managed as a large-scale pool of resources, with acceleration services mapped to remote FPGA resources. Ideally, servers not using all of their local FPGA resources can donate those resources to the global pool, while servers that need additional resources can request the available resources on remote servers. Failing nodes are removed from the pool with replacements quickly added. As demand for a service grows or shrinks, a global manager grows or shrinks the pools correspondingly. Services are thus freed from having a fixed ratio of CPU cores per FPGAs, and can instead allocate (or purchase, in the case of IaaS) only the resources of each type needed.

Space limitations prevent a complete description of the management policies and mechanisms for the global resource manager. Instead, this paper focuses first on the hardware architecture necessary to treat remote FPGAs as available resources for global acceleration pools. We describe the communication protocols and mechanisms that allow nodes in a remote acceleration service to connect, including a protocol called LTL (Lightweight Transport Layer) that supports lightweight connections between pairs of FPGAs, with mostly lossless transport and extremely low latency (small numbers of microseconds). This protocol makes the datacenter-scale remote FPGA resources appear closer than either a single local SSD access or the time to get through the host's networking stack. Then, we describe an evaluation system of 5,760 servers which we built and deployed as a precursor to hyperscale production deployment. We measure the performance characteristics of the system, using web search and network flow encryption as examples. We show that significant gains in efficiency are possible, and that this new architecture enables a much broader and more robust architecture for the acceleration

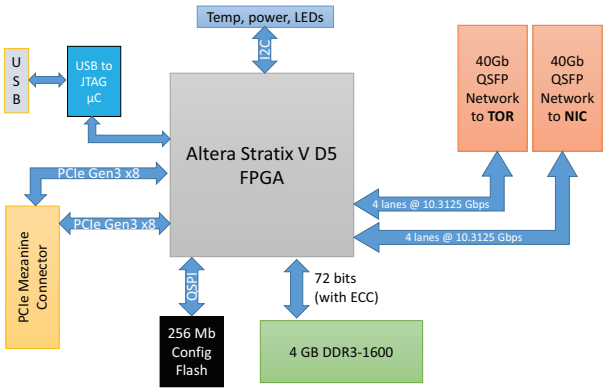

Fig. 2. Block diagram of the major components of the accelerator board.

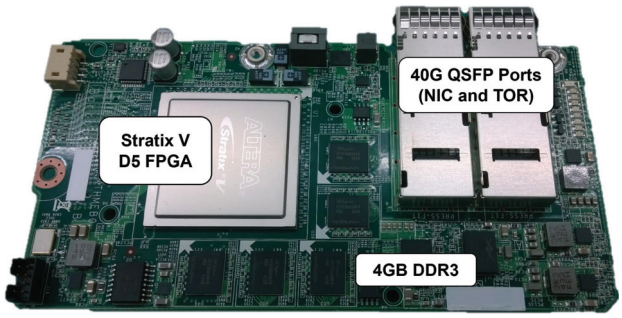

Fig. 3. Photograph of the manufactured board. The DDR channel is implemented using discrete components. PCIe connectivity goes through a mezzanine connector on the bottom side of the board (not shown).

of hyperscale datacenter services.

II. Hardware Architecture

There are many constraints on the design of hardware accelerators for the datacenter. Datacenter accelerators must be highly manageable, which means having few variations or versions. The environments must be largely homogeneous, which means that the accelerators must provide value across a plurality of the servers using it. Given services' rate of change and diversity in the datacenter, this requirement means that a single design must provide positive value across an extremely large, homogeneous deployment.

The solution to addressing the competing demands of homogeneity and specialization is to develop accelerator architectures which are programmable, such as FPGAs and GPUs. These programmable architectures allow for hardware homogeneity while allowing fungibility via software for different services. They must be highly *flexible* at the system level, in addition to being programmable, to justify deployment across a hyperscale infrastructure. The acceleration system we describe is sufficiently flexible to cover three scenarios: local compute acceleration (through PCIe), network acceleration, and global application acceleration, through configuration as pools of remotely accessible FPGAs. Local acceleration handles high-value scenarios such as search ranking acceleration where every server can benefit from having its own FPGA. Network acceleration can support services such as intrusion detection, deep packet inspection and network encryption which are critical to IaaS (e.g. "rental" of cloud servers), and which have such a huge diversity of customers that it makes it difficult to justify local compute acceleration alone economically. Global acceleration permits accelerators unused by their host servers to be made available for large-scale applications, such as machine learning. This decoupling of a 1:1 ratio of servers to FPGAs is essential for breaking the "chicken and egg" problem where accelerators cannot be added until enough applications need them, but applications will not rely upon the accelerators until they are present in the infrastructure. By decoupling the servers and FPGAs, software services that demand more FPGA capacity can harness spare FPGAs from other services that are slower to adopt (or do not require) the accelerator fabric.

In addition to architectural requirements that provide sufficient flexibility to justify scale production deployment, there are also physical restrictions in current infrastructures that must be overcome. These restrictions include strict power limits, a small physical space in which to fit, resilience to hardware failures, and tolerance to high temperatures. For example, the accelerator architecture we describe is the widely-used OpenCompute server that constrained power to 35W, the physical size to roughly a half-height half-length PCIe expansion card (80mm x 140 mm), and tolerance to an inlet air temperature of 70°C at 160 lfm airflow. These constraints make deployment of current GPUs impractical except in special HPC SKUs, so we selected FPGAs as the accelerator.

We designed the accelerator board as a standalone FPGA board that is added to the PCIe expansion slot in a production server SKU. Figure 2 shows a schematic of the board, and Figure 3 shows a photograph of the board with major components labeled. The FPGA is an Altera Stratix V D5, with 172.6K ALMs of programmable logic. The FPGA has one 4 GB DDR3-1600 DRAM channel, two independent PCIe Gen 3 x8 connections for an aggregate total of 16 GB/s in each direction between the CPU and FPGA, and two independent 40 Gb Ethernet interfaces with standard QSFP+ connectors. A 256 Mb Flash chip holds the known-good *golden image* for the FPGA that is loaded on power on, as well as one application image.

To measure the power consumption limits of the entire FPGA card (including DRAM, I/O channels, and PCIe), we developed a power virus that exercises nearly all of the FPGA's interfaces, logic, and DSP blocks—while running the card in a thermal chamber operating in worst-case conditions (peak ambient temperature, high CPU load, and minimum airflow due to a failed fan). Under these conditions, the card consumes 29.2W of power, which is well within the 32W TDP limits for a card running in a single server in our datacenter, and below the max electrical power draw limit of 35W.

The dual 40 Gb Ethernet interfaces on the board could allow for a private FPGA network as was done in our previous

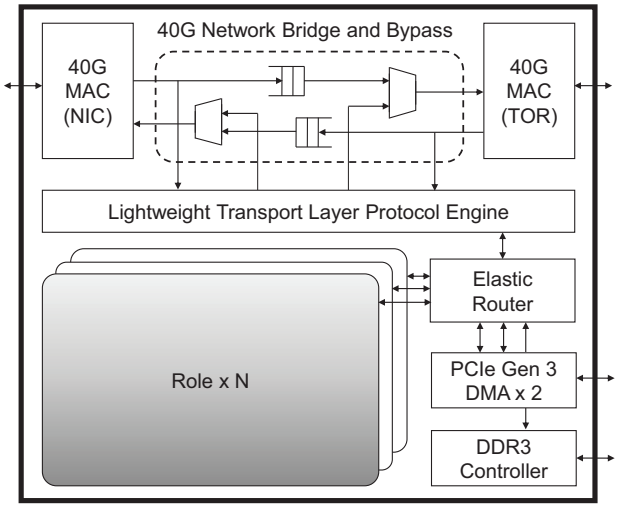

Fig. 4. The Shell Architecture in a Single FPGA.

	ALMs	MHz
Role	55340 (32%)	175
40G MAC/PHY (TOR)	9785 (6%)	313
40G MAC/PHY (NIC)	13122 (8%)	313
Network Bridge / Bypass	4685 (3%)	313
DDR3 Memory Controller	13225 (8%)	200
Elastic Router	3449 (2%)	156
LTL Protocol Engine	11839 (7%)	156
LTL Packet Switch	4815 (3%)	-
PCIe DMA Engine	6817 (4%)	250
Other	8273 (5%)	-
Total Area Used	**131350 (76%)**	-
Total Area Available	172600	-

Fig. 5. Area and frequency breakdown of production-deployed image with remote acceleration support.

work [4], but this configuration also allows the FPGA to be wired as a "bump-in-the-wire", sitting between the network interface card (NIC) and the top-of-rack switch (ToR). Rather than cabling the standard NIC directly to the ToR, the NIC is cabled to one port of the FPGA, and the other FPGA port is cabled to the ToR, as we previously showed in Figure 1b.

Maintaining the discrete NIC in the system enables us to leverage all of the existing network offload and packet transport functionality hardened into the NIC. This simplifies the minimum FPGA code required to deploy the FPGAs to simple bypass logic. In addition, both FPGA resources and PCIe bandwidth are preserved for acceleration functionality, rather than being spent on implementing the NIC in soft logic.

Unlike [5], both the FPGA and NIC have separate connections to the host CPU via PCIe. This allows each to operate independently at maximum bandwidth when the FPGA is being used strictly as a local compute accelerator (with the FPGA simply doing network bypass). This path also makes it possible to use custom networking protocols that bypass the NIC entirely when desired.

One potential drawback to the bump-in-the-wire architecture is that an FPGA failure, such as loading a buggy application, could cut off network traffic to the server, rendering the server unreachable. However, unlike a torus or mesh network, failures in the bump-in-the-wire architecture do not degrade any neighboring FPGAs, making the overall system more resilient to failures. In addition, most datacenter servers (including ours) have a side-channel management path that exists to power servers on and off. By policy, the known-good golden image that loads on power up is rarely (if ever) overwritten, so power cycling the server through the management port will bring the FPGA back into a good configuration, making the server reachable via the network once again.

A. Shell architecture

Within each FPGA, we use the partitioning and terminology we defined in prior work [4] to separate the application logic (Role) from the common I/O and board-specific logic (Shell) used by accelerated services. Figure 4 gives an overview of this architecture's major shell components, focusing on the network. In addition to the Ethernet MACs and PHYs, there is an intra-FPGA message router called the Elastic Router (ER) with virtual channel support for allowing multiple Roles access to the network, and a Lightweight Transport Layer (LTL) engine used for enabling inter-FPGA communication. Both are described in detail in Section V.

The FPGA's location as a bump-in-the-wire between the network switch and host means that it must always pass packets between the two network interfaces that it controls. The shell implements a bridge to enable this functionality, shown at the top of Figure 4. The shell provides a tap for FPGA roles to inject, inspect, and alter the network traffic as needed, such as when encrypting network flows, which we describe in Section III.

Full FPGA reconfiguration briefly brings down this network link, but in most cases applications are robust to brief network outages. When network traffic cannot be paused even briefly, partial reconfiguration permits packets to be passed through even during reconfiguration of the role.

Figure 5 shows the area and clock frequency of the shell IP components used in the production-deployed image. In total, the design uses 44% of the FPGA to support all shell functions and the necessary IP blocks to enable access to remote pools of FPGAs (i.e., LTL and the Elastic Router). While a significant fraction of the FPGA is consumed by a few major shell components (especially the 40G PHY/MACs at 14% and the DDR3 memory controller at 8%), enough space is left for the role(s) to provide large speedups for key services, as we show in Section III. Large shell components that are stable for the long term are excellent candidates for hardening in future generations of datacenter-optimized FPGAs.

B. Datacenter Deployment

To evaluate the system architecture and performance at scale, we manufactured and deployed 5,760 servers containing this accelerator architecture and placed it into a production datacenter. All machines were configured with the shell described above. The servers and FPGAs were stress tested using the power virus workload on the FPGA and a standard burn-in test for the server under real datacenter environmental conditions. The servers all passed, and were approved for production use in the datacenter.

We brought up a production Bing web search ranking service on the servers, with 3,081 of these machines using the FPGA for local compute acceleration, and the rest used for other functions associated with web search. We mirrored live traffic to the bed for one month, and monitored the health and stability of the systems as well as the correctness of the ranking service. After one month, two FPGAs had hard failures, one with a persistently high rate of single event upset (SEU) errors in the configuration logic, and the other with an unstable 40 Gb network link to the NIC. A third failure of the 40 Gb link to the TOR was found not to be an FPGA failure, and was resolved by replacing a network cable. Given aggregate datacenter failure rates, we deemed the FPGA-related hardware failures to be acceptably low for production.

We also measured a low number of soft errors, which were all correctable. Five machines failed to train to the full Gen3 x8 speeds on the secondary PCIe link. There were eight total DRAM calibration failures which were repaired by reconfiguring the FPGA. The errors have since been traced to a logical error in the DRAM interface rather than a hard failure. Our shell scrubs the configuration state for soft errors and reports any flipped bits. We measured an average rate of one bit-flip in the configuration logic every 1025 machine days. While the scrubbing logic often catches the flips before functional failures occur, at least in one case there was a role hang that was likely attributable to an SEU event. Since the scrubbing logic completes roughly every 30 seconds, our system recovers from hung roles automatically, and we use ECC and other data integrity checks on critical interfaces, the exposure of the ranking service to SEU events is low. Overall, the hardware and interface stability of the system was deemed suitable for scale production.

In the next sections, we show how this board/shell combination can support local application acceleration while simultaneously routing all of the server's incoming and outgoing network traffic. Following that, we show network acceleration, and then acceleration of remote services.

III. Local Acceleration

As we described earlier, it is important for an at-scale datacenter accelerator to enhance local applications *and* infrastructure functions for different domains (e.g. web search and IaaS). In this section we measure the performance of our system on a large datacenter workload.

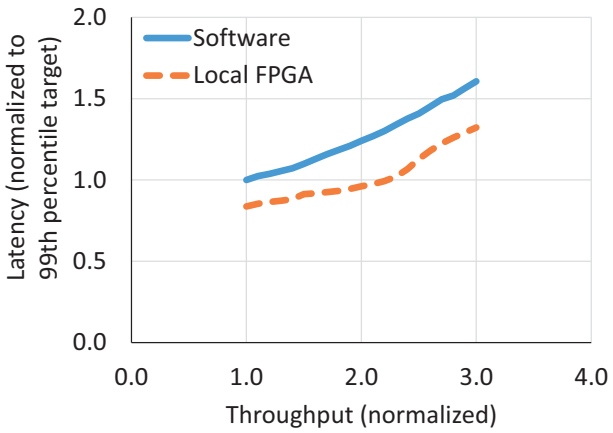

Fig. 6. 99% Latency versus Throughput of ranking service queries running on a single server, with and without FPGAs enabled.

A. Bing Search Page Ranking Acceleration

We describe Bing web search ranking acceleration as an example of using the local FPGA to accelerate a large-scale service. This example is useful both because Bing search is a large datacenter workload, and since we had described its acceleration in depth on the Catapult v1 platform [4]. At a high level, most web search ranking algorithms behave similarly; query-specific features are generated from documents, processed, and then passed to a machine learned model to determine how relevant the document is to the query.

Unlike in [4], we implement only a subset of the feature calculations (typically the most expensive ones), and neither compute post-processed synthetic features nor run the machine-learning portion of search ranking on the FPGAs. We do implement two classes of features on the FPGA. The first is the traditional finite state machines used in many search engines (e.g. "count the number of occurrences of query term two"). The second is a proprietary set of features generated by a complex dynamic programming engine.

We implemented the selected features in a *Feature Functional Unit* (FFU), and the *Dynamic Programming Features* in a separate DPF unit. Both the FFU and DPF units were built into a shell that also had support for execution using remote accelerators, namely the ER and LTL blocks as described in Section V. This FPGA image also, of course, includes the network bridge for NIC-TOR communication, so all the server's network traffic is passing through the FPGA while it is simultaneously accelerating document ranking. The pass-through traffic and the search ranking acceleration have no performance interaction.

We present results in a format similar to the Catapult results to make direct comparisons simpler. We are running this image on a full production bed consisting of thousands of servers. In a production environment, it is infeasible to simulate many different points of query load as there is substantial infrastructure upstream that only produces requests at the rate of arrivals. To produce a smooth distribution with repeatable results, we used a single-box test with a stream

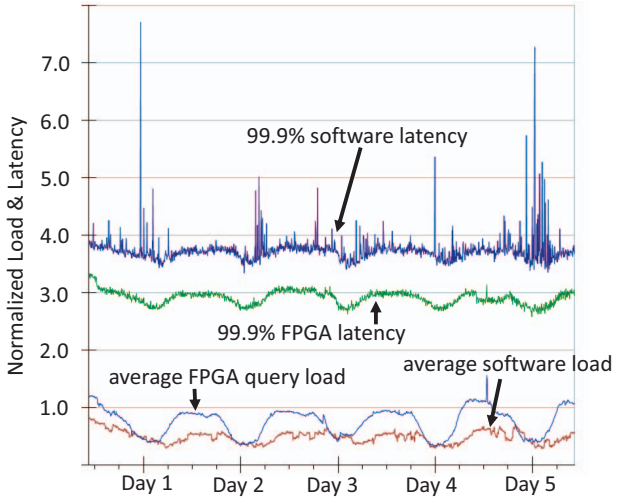

Fig. 7. Five day query throughput and latency of ranking service queries running in production, with and without FPGAs enabled.

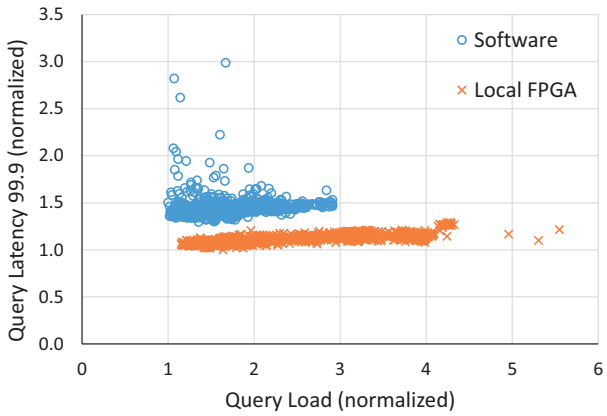

Fig. 8. Query 99.9% Latency vs. Offered Load.

of 200,000 queries, and varied the arrival rate of requests to measure query latency versus throughput. Figure 6 shows the results; the curves shown are the measured latencies of the 99% slowest queries at each given level of throughput. Both axes show normalized results.

We have normalized both the target production latency and typical average throughput in software-only mode to 1.0. The software is well tuned; it can achieve production targets for throughput at the required 99th percentile latency. With the single local FPGA, at the target 99th percentile latency, t he throughput can be safely increased by 2.25x, which means that fewer than half as many servers would be needed to sustain the target throughput at the required latency. Even at these higher loads, the FPGA remains underutilized, as the software portion of ranking saturates the host server before the FPGA is saturated. Having multiple servers drive fewer FPGAs addresses the underutilization of the FPGAs, which is the goal of our remote acceleration model.

Production Measurements: We have deployed the FPGA accelerated ranking service into production datacenters at scale and report end-to-end measurements below.

Figure 7 shows the performance of ranking service running in two production datacenters over a five day period, one with FPGAs enabled and one without (both datacenters are of identical scale and configuration, with the exception of the FPGAs). These results are from live production traffic, not on a synthetic workload or mirrored traffic. The top two bars show the normalized tail query latencies at the 99.9th percentile (aggregated across all servers over a rolling time window), while the bottom two bars show the corresponding query loads received at each datacenter. As load varies throughout the day, the queries executed in the software-only datacenter experience a high rate of latency spikes, while the FPGA-accelerated queries have much lower, tighter-bound latencies, despite seeing much higher peak query loads.

Figure 8 plots the load versus latency over the same 5-day period for the two datacenters. Given that these measurements were performed on production datacenters and traffic, we were unable to observe higher loads on the software datacenter (vs. the FPGA-enabled datacenter) due to a dynamic load balancing mechanism that caps the incoming traffic when tail latencies begin exceeding acceptable thresholds. Because the FPGA is able to process requests while keeping latencies low, it is able to absorb more than twice the offered load, while executing queries at a latency that never exceeds the software datacenter **at any load**.

IV. Network Acceleration

The bump-in-the-wire architecture was developed to enable datacenter networking applications, the first of which is host-to-host line rate encryption/decryption on a per flow basis. As each packet passes from the NIC through the FPGA to the ToR, its header is examined to determine if it is part of an encrypted flow that was previously set up by software. If it is, the software-provided encryption key is read from internal FPGA SRAM or the FPGA-attached DRAM and is used to encrypt or decrypt the packet. Thus, once the encrypted flows are set up, there is no load on the CPUs to encrypt or decrypt the packets; encryption occurs transparently from software's perspective, which sees all packets as unencrypted at the end points.

The ability to offload encryption/decryption at network line rates to the FPGA yields significant CPU savings. According to Intel [6], its AES GCM-128 performance on Haswell is 1.26 cycles per byte for encrypt and decrypt each. Thus, at a 2.4 GHz clock frequency, 40 Gb/s encryption/decryption consumes roughly five cores. Different standards, such as 256b or CBC are, however, significantly slower. In addition, there is sometimes a need for crypto hash functions, further reducing performance. For example, AES-CBC-128-SHA1 is needed for backward compatibility for some software stacks and consumes at least fifteen cores to achieve 40 Gb/s full duplex. Of course, consuming fifteen cores just for crypto is impractical on a typical multi-core server, as there would be that many fewer cores generating traffic. Even five cores of

savings is significant when every core can otherwise generate revenue.

Our FPGA implementation supports full 40 Gb/s encryption and decryption. The worst case half-duplex FPGA crypto latency for AES-CBC-128-SHA1 is 11 μs for a 1500B packet, from first flit to first flit. In software, based on the Intel numbers, it is approximately 4 μs. AES-CBC-SHA1 is, however, especially difficult for hardware due to tight dependencies. For example, AES-CBC requires processing 33 packets at a time in our implementation, taking only 128b from a single packet once every 33 cycles. GCM latency numbers are significantly better for FPGA since a single packet can be processed with no dependencies and thus can be perfectly pipelined.

The software performance numbers are Intel's very best numbers that do not account for the disturbance to the core, such as effects on the cache if encryption/decryption is blocked, which is often the case. Thus, the real-world benefits of offloading crypto to the FPGA are greater than the numbers presented above.

V. Remote Acceleration

In the previous section, we showed that the Configurable Cloud acceleration architecture could support local functions, both for Bing web search ranking acceleration and accelerating networking/infrastructure functions, such as encryption of network flows. However, to treat the acceleration hardware as a global resource and to deploy services that consume more than one FPGA (e.g. more aggressive web search ranking, large-scale machine learning, and bioinformatics), communication among FPGAs is crucial. Other systems provide explicit connectivity through secondary networks or PCIe switches to allow multi-FPGA solutions. Either solution, however, limits the scale of connectivity. By having FPGAs communicate directly through the datacenter Ethernet infrastructure, large scale and low latency are both achieved. However, these communication channels have several requirements:

- They cannot go through software protocol stacks on the CPU due to their long latencies.
- They must be resilient to failures and dropped packets, but should drop packets rarely.
- They should not consume significant FPGA resources.

The rest of this section describes the FPGA implementation that supports cross-datacenter, inter-FPGA communication and meets the above requirements. There are two major functions that must be implemented on the FPGA: the inter-FPGA communication engine and the intra-FPGA router that coordinates the various flows of traffic on the FPGA among the network, PCIe, DRAM, and the application roles. We describe each below.

A. Lightweight Transport Layer

We call the inter-FPGA network protocol LTL, for Lightweight Transport Layer. This protocol, like previous work [7], uses UDP for frame encapsulation and IP for routing packets across the datacenter network. Low-latency communication demands infrequent packet drops and infrequent packet reorders. By using "lossless" traffic classes provided in datacenter switches and provisioned for traffic like RDMA and FCoE, we avoid most packet drops and reorders. Separating out such traffic to their own classes also protects the datacenter's baseline TCP traffic. Since the FPGAs are so tightly coupled to the network, they can react quickly and efficiently to congestion notification and back off when needed to reduce packets dropped from incast patterns.

Figure 9 gives a block diagram of the LTL protocol engine used to support the inter-FPGA network protocol. At the endpoints, the LTL protocol engine uses an ordered, reliable connection-based interface with statically allocated, persistent connections, realized using send and receive connection tables. The static allocation and persistence (until they are deallocated, of course) reduces latency for inter-FPGA and inter-service messaging, since once established they can communicate with low latency. Reliable messaging also reduces protocol latency. Although datacenter networks are already fairly reliable, LTL provides a strong reliability guarantee via an ACK/NACK based retransmission scheme. Outgoing packets are buffered and tracked in an unacknowledged frame queue until their receipt is acknowledged by the receiver (see Ack Generation and Ack Receiver in Figure 9). Timeouts trigger retransmission of unACKed packets. In some cases, such as when packet reordering is detected, NACKs are used to request timely retransmission of particular packets without waiting for a timeout. Timeouts can also be used to identify failing nodes quickly, if ultra-fast reprovisioning of a replacement is critical to the higher-level service. The exact timeout value is configurable, and is currently set to 50 μsec.

Datacenter networks handle multiple traffic classes and protocols, some of which expect near-lossless behavior. FPGAs routing traffic between the server's NIC and TOR, as a bump-in-the-wire, must not interfere with the expected behavior of these various traffic classes. To that end, the LTL Protocol Engine shown in Figure 4 allows roles to send and receive packets from the network without affecting–and while supporting–existing datacenter protocols.

To achieve both requirements, the tap supports per-flow congestion management, traffic class based flow control, and bandwidth limiting via random early drops. It also performs basic packet classification and buffering to map packets to classes. Our LTL implementation is capable of generating and responding to Priority Flow Control [8] frames to pause traffic on lossless traffic classes. LTL also implements the DC-QCN[9] end-to-end congestion control scheme. In combination, these features allow the FPGA to safely insert and remove packets from the network without disrupting existing flows and without host-side support.

We measured the end-to-end round-trip latency to go from one FPGA to another, where both FPGAs are attached to the same TOR to be 2.88 μs. This delay is comparable to the average latencies in our Catapult v1 system [4], where nearest neighbor (1-hop) communication had a round-trip latency of approximately 1 μs. However, worst-case round-trip

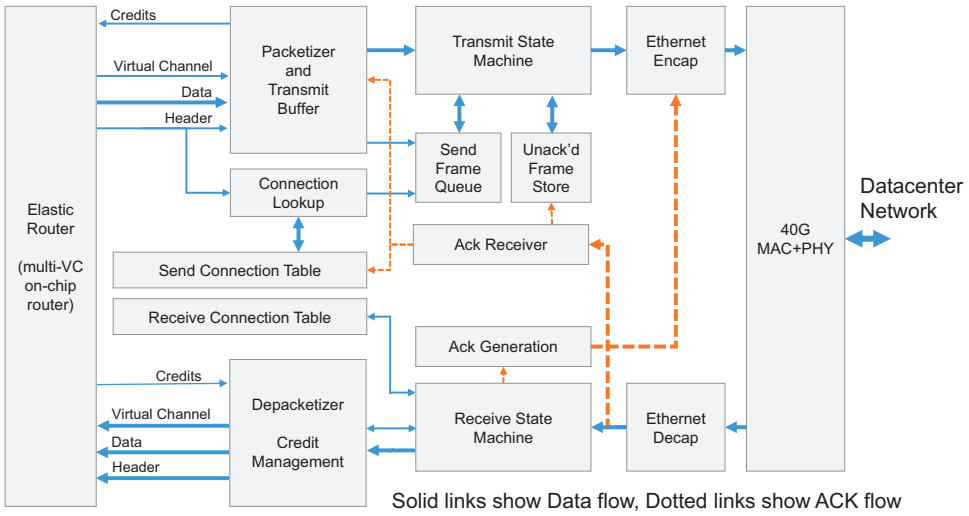

Fig. 9. Block diagram of the LTL Protocol Engine.

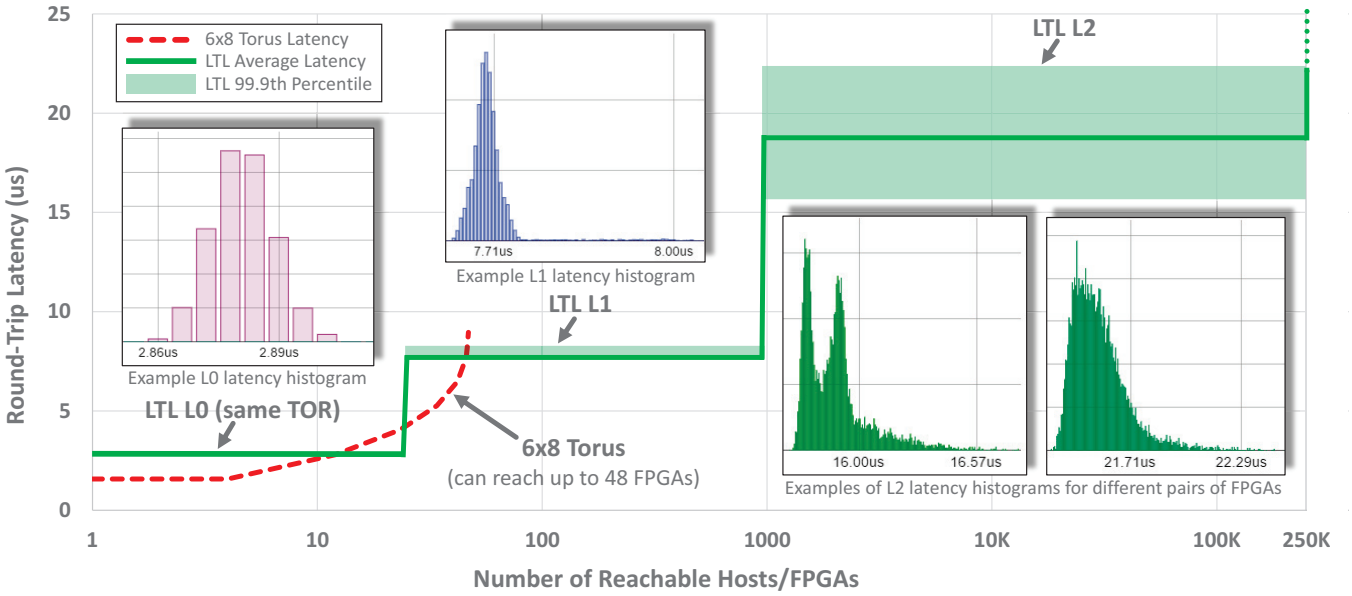

Fig. 10. Round-trip latency of accesses to remote machines with LTL compared to the 6x8 torus from [4]. Shaded areas represent range of latencies up to the 99.9th percentile. LTL enables round-trip access to 100,000+ machines in under 23.5 μs.

communication in the torus requires 7 μsec, slower than LTL. These latencies are also comparable to RDMA read latencies that we measured on the same types of systems. A complete evaluation follows in section V-C.

B. Elastic Router

The Elastic Router is an on-chip, input-buffered crossbar switch designed to support efficient communication between multiple endpoints on an FPGA across multiple virtual channels (VCs). This section describes the architectural and microarchitectural specifications of ER, its design rationales, and how it integrates with other components.

The Elastic Router was developed to support intra-FPGA communication between Roles on the same FPGA and inter-FPGA communication between Roles running on other FPGAs through the Lightweight Transport Layer. In an example single-role deployment, the ER is instantiated with 4 ports: (1) PCIe DMA, (2) Role, (3) DRAM, and (4) Remote (to LTL). The design can be fully parameterized in the number of ports, virtual channels, flit and phit sizes, and buffer capacities.

Any endpoint can send a message through the ER to any other port including itself as U-turns are supported. In addition, multiple ERs can be composed to form a larger on-chip network topology, e.g., a ring or a 2-D mesh.

The ER supports multiple virtual channels, virtualizing the physical links between input and output ports. The ER employs credit-based flow control, one credit per flit, and is an input-buffered switch. Unlike a conventional router that allocates a static number of flits per VC, the ER supports an elastic policy

that allows a pool of credits to be shared among multiple VCs, which is effective in reducing the aggregate flit buffering requirements.

The LTL and ER blocks are crucial for allowing FPGAs to (1) be organized into multi-FPGA services, and (2) to be remotely managed for use as a remote accelerator when not in use by their host. The area consumed is 7% for LTL and 2% for ER (Figure 5.) While not insubstantial, services using only their single local FPGA can choose to deploy a shell version without the LTL block. Services needing a multi-FPGA accelerator or services not using their local FPGA (to make it available for global use) should deploy shell versions with the LTL block.

With this communication protocol, it is now possible to manage the datacenter's accelerator resources as a global pool. The next sections provide an overview of LTL performance, describe an example service running remotely, and give a brief overview of how hardware services are managed.

C. LTL Communication Evaluation

Our datacenter network is organized into three tiers. At the bottom tier (L0), each top-of-rack (TOR) switch connects directly to 24 hosts. The next tier of switches (L1) form pods of 960 machines. The final layer (L2) connects multiple pods together that can connect more than a quarter million of machines. Each layer of the hierarchy introduces more over-subscription such that the node-to-node bandwidth is greatest between nodes that share a L0 switch and least between pairs connected via L2.

Figure 10 shows LTL round-trip latency results for FPGAs connected through the different datacenter network tiers described above, namely L0, L1, and L2. For each tier, lines represent average latency while the shaded areas capture the range of latencies observed up to the 99.9th percentile. Note that the x-axis is logarithmic. Results were obtained through cycle-level measurements across multiple sender-receiver pairs and capture idle LTL round-trip latency from the moment the header of a packet is generated in LTL until the corresponding ACK for that packet is received in LTL. For each tier we also include sample latency distributions from individual runs. As a point of comparison, we also show results from our previously published 6x8 torus network [4] that is, however, limited to 48 FPGAs. Note that even though we generated LTL traffic at a very low rate to obtain representative idle latencies, L1 and L2 results are inevitably affected by other datacenter traffic that is potentially flowing through the same switches.

Average round-trip latency for FPGAs on the same TOR (L0) is 2.88 μs. L0 distributions were consistently very tight across all runs with the 99.9th percentile at 2.9 μs. L1 average latency is 7.72 μs with a 99.9th percentile latency of 8.24 μs indicating a tight distribution for the majority of L1 traffic. However, in this case there is also a small tail of outliers—possibly packets that got stuck behind other traffic going through the same L1 switch—that encounter up to roughly half a microsecond of additional latency. L2 average latency is 18.71 μs with a 99.9th percentile of 22.38 μs. Even though

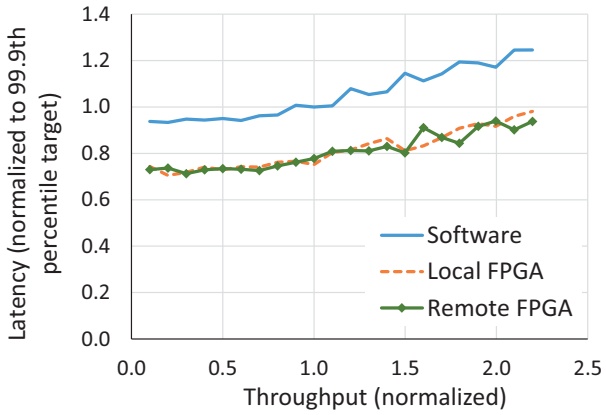

Fig. 11. Latencies of software ranking, locally accelerated ranking, and remotely accelerated ranking. All data are normalized to software 99.9th percentile latency target.

L2 latencies and distributions can vary significantly, as is highlighted by the two example L2 histograms, it is worth noting that L2 latency never exceeded 23.5 μs in any of our experiments. The higher L2 latency variability and noise does not come as a surprise as L2 switches can connect up to hundreds of thousands of hosts. In addition to oversubscription effects, which can become much more pronounced at this level, L2 latencies can be affected by several other factors that range from physical distance and cabling to transient background traffic from other workloads and even L2 switch internal implementation details, such as multi-pathing and ASIC organization.

Compared to LTL, our previous 6x8 torus, which employs a separate physical inter-FPGA network, offers comparable round-trip latencies at low FPGA counts. However, communication is strictly limited to groups of 48 FPGAs and the separate dedicated inter-FPGA network can be expensive and complex to cable and maintain. Similarly, failure handling in the torus can be quite challenging and impact latency as packets need to be dynamically rerouted around a faulty FPGA at the cost of extra network hops and latency. LTL on the other hand shares the existing datacenter networking infrastructure allowing access to hundreds of thousands of hosts/FPGAs in a fixed number of hops. Failure handling also becomes much simpler in this case as there is an abundance of spare accessible nodes/FPGAs.

D. Remote Acceleration Evaluation

To study the end-to-end impact of accelerating production-level applications using remote FPGAs, we evaluate the search ranking accelerator from Section III running remotely over the network via LTL. Figure 11 shows the throughput of a single accelerator when accessed remotely compared to the software and locally attached accelerator. The data are all normalized to the 99.9th percentile latency target of ranking running in software mode. The data show that over a range of throughput targets, the latency overhead of remote accesses is minimal.

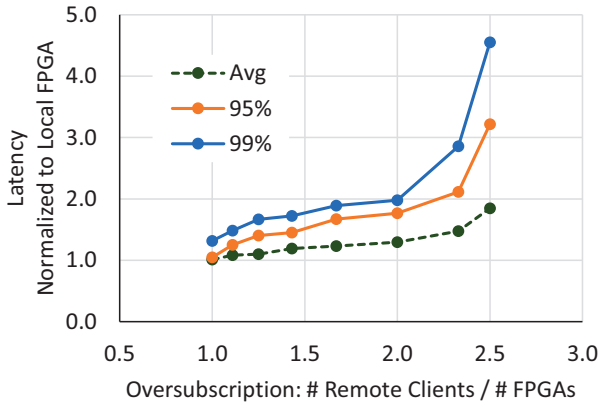

Fig. 12. Average, 95th, and 99th percentile latencies to a remote DNN accelerator (normalized to locally-attached performance in each latency category).

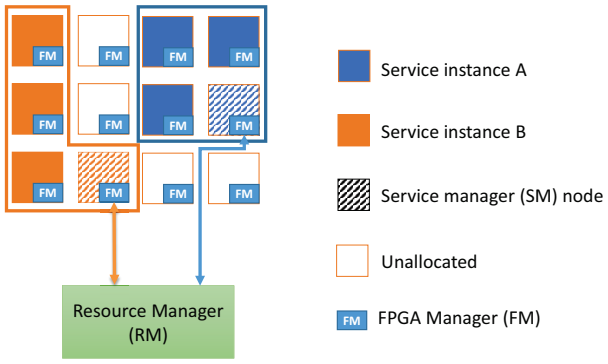

Fig. 13. Two Hardware-as-a-Service (HaaS) enabled hardware accelerators are shown running under HaaS. FPGAs are allocated to each service from Resource Manager's resource pool. Each service has a Service Manager node to administer the service on the allocated resources. Each FPGA has a lightweight FPGA Manager for managing calls from the Service Manager.

The impact on the host server while serving remote requests is minimal, because the FPGA directly handles the network and processing load. The host sees no increase in CPU or memory utilization, and only a small increase in overall power draw. Thus, FPGA-enabled servers can safely donate their FPGA to the global pool of hardware resources with minimal impact on software performance. One concern when sharing the FPGA with remote services is that network bandwidth can be reduced by the remote service. To prevent issues, LTL implements bandwidth limiting to prevent the FPGA from exceeding a configurable bandwidth limit.

E. Oversubscription Evaluation

One key motivation behind enabling remote hardware services is that the resource requirements for the hardware fabric rarely map 1:1 with the resource needs for the software fabric. Some services will need more FPGAs than the number of servers. Other services will have unused FPGA resources which can be made available to other services. Because the accelerators communicate directly rather than through the CPU, a service borrowing an FPGA can do so with minimal impact to the performance of the host server.

To evaluate the impact of remote service oversubscription, we deployed a small pool of latency-sensitive Deep Neural Network (DNN) accelerators shared by multiple software clients in a production datacenter. To stress the system, each software client sends synthetic traffic to the DNN pool at a rate several times higher than the expected throughput per client in deployment. We increased the ratio of software clients to accelerators (by removing FPGAs from the pool) to measure the impact on latency due to oversubscription.

Figure 12 shows the average, 95th and 99th percentile request latencies as the ratio of clients to FPGAs (oversubscription) increases. These figures plot end-to-end request latencies, measuring the time between when a request is enqueued to the work queue and when its response is received from the accelerator. To expose the latencies more clearly, these results do not include end-to-end service and software latencies as the ranking data in Figure 11 do. In the no oversubscription (1 to 1) case, remotely accessing the service adds 1% additional latency to each request on average, 4.7% additional latency at the 95th percentile, and 32% at the 99th percentile. As expected, contention and queuing delay increases as oversubscription increases. Eventually, the FPGA reaches its peak throughput and saturates, causing latencies to spike due to rapidly increasing queue depths. In this particular case study, each individual FPGA has sufficient throughput to sustain 2-2.5 software clients (operating at very high rates several times the expected throughput in production) before latencies begin to spike prohibitively. This shows that conservatively half to two-thirds of the FPGAs can be freed up for other functions.

F. Hardware-as-a-Service Model

While a complete overview of the management of our hardware fabric is beyond the scope of this paper, we provide a short overview of the Hardware-as-a-Service (HaaS) platform here. HaaS manages FPGAs in a manner similar to Yarn [10] and other job schedulers. Figure 13 shows a few services running under the HaaS model. A logically centralized Resource Manager (RM) tracks FPGA resources throughout the datacenter. The RM provides simple APIs for higher-level Service Managers (SM) to easily manage FPGA-based hardware Components through a lease-based model. Each Component is an instance of a hardware service made up of one or more FPGAs and a set of constraints (locality, bandwidth, etc.). SMs manage service-level tasks such as load balancing, inter-component connectivity, and failure handling by requesting and releasing Component leases through RM. A SM provides pointers to the hardware service to one or more end users to take advantage of the hardware acceleration. An FPGA Manager (FM) runs on each node to provide configuration and status monitoring for the system.

VI. Related Work

There are many possible options for incorporating accelerators into large-scale systems, including the type of accelerator, how it interfaces with the CPUs, and how the accelerators can

communicate with one another. Below we describe a taxonomy that uses those three categories.

1) CPU-Accelerator memory integration. The closer the integration with the CPU, the finer-grain the problem that can be beneficially offloaded to the accelerator. Possibilities include:
   - (C) - Coherent accelerators, where data movement is handled by a memory coherence protocol.
   - (I) - I/O level, where data movement is done via DMA transfers, and
   - (N) - Network level, where data movement is done via Ethernet (or other) packets.
2) Accelerator connectivity scale. The scale at which accelerators can directly communicate without CPU intervention. Possibilities include:
   - (S) Single server / single appliance (i.e. CPU management is required to communicate with accelerators on other servers)
   - (R) Rack level
   - (D) Datacenter scale
3) Accelerator type. Possibilities include:
   - (F) FPGAs
   - (G) GPUs
   - (A) ASICs

In this section we describe a subset of previously proposed datacenter or server accelerators, organized by this taxonomy, with an emphasis on FPGA-based accelerators. While most designs fit into one category, our proposed design could fit into two. Our accelerator could fit as an ISF design when doing local acceleration since there are no additional FPGAs (and hence no inter-FPGA connectivity) within the same server. However, the design is an NDF design when doing network acceleration since connected FPGAs (including the local one) can all be accessed via Ethernet packets. LTL ties together local acceleration and network acceleration scenarios, so altogether we view it as an NDF architecture. All of the systems that we survey below fall into other categories.

NSF: Network/Single/FPGA: The announced Mellanox hybrid NIC [5] enables one to build a bump-in-the-wire architecture where the FPGA is in-line with the NIC, although there has not been a published large-scale deployment of the Mellanox hardware. Other FPGA boards, such as NetFPGA [11], high-speed trading appliances, and network appliances are specifically designed to augment or replace standard network interface cards for specific applications. While these can each be considered bump-in-the wire architectures, there is little to no communication between or aggregation of accelerators. As such, they are limited to problem sizes that fit on within a single accelerator or appliance.

In addition to commercial network acceleration designs, there has been considerable research into FPGA-based bump-in-the-wire architectures. For example, FPGA-accelerated Memcached designs lend themselves naturally to directly attaching FPGA to the network [12], [13], [14], [15] and programming the FPGA to handle requests some directly from the network without software interference.

IRF: IO/Rack/FPGA: Catapult v1 is an IRF system, where FPGAs are connected by a dedicated network at rack scale [4] accessible through PCIe.

The scale of that communication is limited to a single rack, which similarly limits the scope of acceleration services that can be supported. In addition, these 2D torus network topologies suffer from resiliency challenges since the failure of one node affects neighboring nodes.

The newer version of Novo-G, called Novo-G# [16] also falls into this category with a three-dimensional 4x4x4 torus. Another such system is the Cray XD-1 [17], which places up to six FPGAs in a single chassis. These FPGAs attach directly to the dedicated rack-scale RapidArray network, which is distinct from the Ethernet network that connects multiple racks of up to 12 chassis. Another example is Maxwell [18], which also provides rack-scale direct FPGA-to-FPGA communication.

ISF: IO/Single/FPGA: The Baidu SDA [3], Novo-G [19], Maxeler MPC [20], Convey HC-2 [21], BeeCube BEE4 [22], and SRC MAPStation [23] are all large or multi-FPGA appliances, many with high connectivity between FPGAs within the appliance, but the CPU manages communication beyond a single box.

ISFG: IO/Zero/FPGA+GPU: The QP [24] system is designed to handle larger HPC-style problems with a large number of FPGAs and GPUs. However, all communication between accelerators has to be managed by the CPU, which increases the latency of inter-FPGA communication and limits the scale of problems that can be profitably accelerated by multiple FPGAs.

CSF: Coherent/Single/FPGA: Pulling the FPGA into the coherence domain of the CPU improves the granularity of communication between the accelerator and CPU, and can increase the scope of applications that can be profitably offloaded to the accelerator. One of the most prominent examples is IBM, who is shipping Power8 [25] systems with a coherent accelerator interface called CAPI [26]. Researchers have demonstrated several applications on CAPI such as bioinformatics [27] and large matrix processing [28].

Intel's hybrid CPU/FPGA [29] is another example. FPGA boards that included Intel FSB [30], QPI [31], or coherent HyperTransport [32], [33] are also included. While coherence helps within a limited scope, it does not scale across servers, and hence does not significantly differ on a datacenter scale from I/O integrated accelerators, as the CPU manages all FPGA-to-FPGA communication.

ISG: IO/Single/GPU: GPUs are to-date the most successful architecture for accelerating CPU-based systems. Multiple GPUs are commonly used for large problems, and the addition of technologies like NVLink [34] have enabled multiple GPUs to talk directly. However, the scale of NVLink is still limited to a single box, and there is no integration of GPUs with the network, so the scale of GPU acceleration achievable without CPU intervention is still relatively small. Some compelling recent work looked at using GPUs to offer DNNs as a service in the datacenter [35], with a subset of the datacenter's servers containing GPUs, and in a disaggregated design, servers with low-end CPUs feeding many GPUs in a single box. In all configurations, the GPU was accessed by the

CPU over PCIe, and all inter-GPU communication occurred within a single server.

ISA: IO/Single/ASIC: ASICs, whether in the form of stand-alone accelerators or as custom blocks integrated into conventional CPUs, are another approach that has worked well in client systems and for some server infrastructure functions. However, the general-purpose nature and rapid pace of change of applications makes single-function ASICs challenging to deploy at scale. Counterexamples may be crypto or compression blocks; however, the economics at scale of doing crypto and compression in soft vs. hard logic are not yet clear. In the long term, we expect that some acceleration functions–particularly for system offload functions–will first be implemented on FPGAs, and then after long-term stability of that function has been demonstrated, the function could be moved into hardened logic. Machine learning is one application that is sufficiently important to justify as a domain-specific ASIC at scale. One such example is the DianNao family of deep learning accelerators [36].

Other taxonomies: Fahmy and Vipin create a service taxonomy based on how FPGAs are exposed to external customers from a datacenter environment [37]. The remote acceleration model that we propose in this paper focuses not on whom the accelerators are exposed to, but the architecture, deployment, and performance of services on a global pool of accelerators. The remote acceleration model can support all three models proposed by Fahmy and Vipin, including Vendor Acceleration, Accelerators as a Service, and Fabric as a Service.

Finally, while other promising programmable accelerator architectures exist, such as MPPAs and CGRAs, none have reached the level of commercial viability and availability to be ready for production datacenters.

VII. Conclusions

The slowing of Moore's Law, coupled with the massive and growing scale of datacenter infrastructure, makes specialized accelerators extremely important. The most important problem to solve is in the design of *scalable accelerators*, which are economically viable across a large infrastructure, as opposed to accelerators that enhance a specialized small or medium-scale deployment of machines. What has made research in this space challenging is the large number of possible design options, spanning the type of accelerator (FPGAs, GPUs, ASICs), their location in the infrastructure (coherent, PCIe space, or network path), and their scale (how many can communicate directly with one another).

This paper described Configurable Clouds, a datacenter-scale acceleration architecture, based on FPGAs, that is both scalable and flexible. By putting in FPGA cards both in I/O space as well as between a server's NIC and the local switch, the FPGA can serve as both a network accelerator and local compute accelerator. By enabling the FPGA to talk directly to the network switch, each FPGA can communicate directly with every other FPGA in the datacenter, over the network, without any CPU software. This flexibility enables ganging together groups of FPGAs into service pools, a concept we call Hardware as a Service (HaaS). We demonstrate a reliable communication protocol for inter-FPGA communication that achieves comparable latency to prior state of the art, while scaling to hundreds of thousands of nodes.

The architecture was demonstrated successfully across multiple datacenter scenarios: use as a local offload engine (for accelerating Bing web search), a local network acceleration engine (for network crypto), and as a remote acceleration service for web search. With the Configurable Clouds design, reconfigurable logic becomes a first-class resource in the datacenter, and over time may even be running more computational work than the datacenter's CPUs. There will still, of course, be a role for GPUs and ASICs, which can augment a subset of servers with capabilities to accelerate specific workloads. This reconfigurable acceleration plane, however, will be pervasive, offering large-scale gains in capabilities and enabling rapid evolution of datacenter protocols. In addition to being a compute accelerator, we anticipate that it will drive evolution of the datacenter architecture in the near future, including network protocols, storage stacks, and physical organization of components.

The Catapult v2 architecture has already been deployed at hyperscale and is how most new Microsoft data center servers are configured. The FPGAs accelerate both compute workloads, such as Bing web search ranking, and Azure infrastructure workloads, such as software-defined networks and network crypto.

Acknowledgements

The authors would like to acknowledge the many individuals who contributed to this project: Mike Andrewartha, Jack Lavier, Mark Shaw, Kushagra Vaid, and the rest of the CSI team; Shlomi Alkalay, Kyle Holohan, Raja Seera, Phillip Xiao, and the rest of the Bing IndexServe team; Daniel Firestone, Albert Greenberg, Dave Maltz, and the rest of the Azure CloudNet team; Stuart Byma, Alvin Lebeck, and Jason Thong.

References

[1] M. Staveley, "Applications that scale using GPU Compute," in *AzureCon 2015*, August 2015.

[2] J. Barr, "Build 3D Streaming Applications with EC2's New G2 Instance Type," Nov 2013.

[3] J. Ouyang, S. Lin, W. Qi, Y. Wang, B. Yu, and S. Jiang, "SDA: Software-Defined Accelerator for Large-Scale DNN Systems," in *HotChips 2014*, August 2014.

[4] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. R. Larus, E. Peterson, G. Prashanth, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *International Symposium on Computer Architecture (ISCA)*, 2014.

[5] Mellanox, "ConnectX-4 Lx EN Programmable Adapter Card. Rev. 1.1," 2015.

[6] S. Gulley and V. Gopal, "Haswell Cryptographic Performance," July 2013. Available at http://www.intel.com/content/www/us/en/communications/haswell-cryptographic-performance-paper.html.

[7] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala, "An fpga memcached appliance," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, (New York, NY, USA), pp. 245–254, ACM, 2013.

[8] IEEE, *IEEE 802.1Qbb - Priority-based Flow Control*, June 2011 ed., 2011.

[9] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), pp. 523–536, ACM, 2015.

[10] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, (New York, NY, USA), pp. 5:1–5:16, ACM, 2013.

[11] G. Gibb, J. Lockwood, J. Naous, P. Hartke, and N. McKeown, "NetFPGA–An Open Platform for Teaching How to Build Gigabit-Rate Network Switches and Routers," in *IEEE Transactions on Education*, 2008.

[12] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached," *SIGARCH Comput. Archit. News*, vol. 41, pp. 36–47, June 2013.

[13] M. lavasani, H. Angepat, and D. Chiou, "An fpga-based in-line accelerator for memcached," *Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2013.

[14] M. Blott and K. Vissers, "Dataflow architectures for 10gbps line-rate key-value-stores," in *HotChips 2013*, August 2013.

[15] E. S. Fukuda, H. Inoue, T. Takenaka, D. Kim, T. Sadashisa, T. Asai, and M. Motomura, "Caching mecached at reconfigurable network interface," in *Proceedings of the 24th International Conference on Field Programmable Logic and Applications*, 2014.

[16] A. G. Lawande, A. D. George, and H. Lam, "Novo-g: a multidimensional torus-based reconfigurable cluster for molecular dynamics," *Concurrency and Computation: Practice and Experience*, pp. n/a–n/a, 2015. cpe.3565.

[17] Cray, *Cray XD1 Datasheet*, 1.3 ed., 2005.

[18] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. Mccormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest, "Maxwell - a 64 FPGA Supercomputer," *Engineering Letters*, vol. 16, pp. 426–433, 2008.

[19] A. George, H. Lam, and G. Stitt, "Novo-g: At the forefront of scalable reconfigurable supercomputing," *Computing in Science Engineering*, vol. 13, no. 1, pp. 82–86, 2011.

[20] O. Pell and O. Mencer, "Surviving the end of frequency scaling with reconfigurable dataflow computing," *SIGARCH Comput. Archit. News*, vol. 39, pp. 60–65, Dec. 2011.

[21] Convey, *The Convey HC-2 Computer*, conv-12-030.2 ed., 2012.

[22] BEECube, *BEE4 Hardware Platform*, 1.0 ed., 2011.

[23] SRC, *MAPstation Systems*, 70000 AH ed., 2014.

[24] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, and W. Hwu, "Qp: A heterogeneous multi-accelerator cluster," 2009.

[25] J. Stuecheli, "Next Generation POWER microprocessor," in *HotChips 2013*, August 2013.

[26] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, "Capi: A coherent accelerator processor interface," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 7–1, 2015.

[27] M. J. Jaspers, *Acceleration of read alignment with coherent attached FPGA coprocessors*. PhD thesis, TU Delft, Delft University of Technology, 2015.

[28] C.-C. Chung, C.-K. Liu, and D.-H. Lee, "Fpga-based accelerator platform for big data matrix processing," in *Electron Devices and Solid-State Circuits (EDSSC), 2015 IEEE International Conference on*, pp. 221–224, IEEE, 2015.

[29] P. Gupta, "Xeon+fpga platform for the data center," 2015.

[30] L. Ling, N. Oliver, C. Bhushan, W. Qigang, A. Chen, S. Wenbo, Y. Zhihong, A. Sheiman, I. McCallum, J. Grecco, H. Mitchel, L. Dong, and P. Gupta, "High-performance, Energy-efficient Platforms Using In-Socket FPGA Accelerators," in *FPGA'09: Proceeding of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, (New York, NY, USA), pp. 261–264, ACM, 2009.

[31] Intel, "An Introduction to the Intel Quickpath Interconnect," 2009.

[32] D. Slogsnat, A. Giese, M. Nüssle, and U. Brüning, "An open-source hypertransport core," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, pp. 14:1–14:21, Sept. 2008.

[33] DRC, *DRC Accelium Coprocessors Datasheet*, ds ac 7-08 ed., 2014.

[34] "NVIDIA NVLink High-Speed Interconnect: Application Performance," Nov. 2014.

[35] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pp. 27–40, ACM, 2015.

[36] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM SIGPLAN Notices*, vol. 49, pp. 269–284, ACM, 2014.

[37] S. A. Fahmy and K. Vipin, "A case for fpga accelerators in the cloud," Poster at SoCC 2014.