2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud

Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee,
Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang,
Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, Krste Asanović

Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
{*sagark, zhemao, dgkim, biancolin, alonamid, dayeol, nathanp, amaro, colins, adichopra,*
qijing.huang, kylekovacs, bora, randy, jrb, krste}*@eecs.berkeley.edu*

***Abstract*—We present FireSim, an open-source simulation platform that enables cycle-exact microarchitectural simulation of large scale-out clusters by combining FPGA-accelerated simulation of silicon-proven RTL designs with a scalable, distributed network simulation. Unlike prior FPGA-accelerated simulation tools, FireSim runs on Amazon EC2 F1, a public cloud FPGA platform, which greatly improves usability, provides elasticity, and lowers the cost of large-scale FPGA-based experiments. We describe the design and implementation of FireSim and show how it can provide sufficient performance to run modern applications at scale, to enable true hardware-software co-design. As an example, we demonstrate automatically generating and deploying a target cluster of 1,024 3.2 GHz quad-core server nodes, each with 16 GB of DRAM, interconnected by a 200 Gbit/s network with 2 microsecond latency, which simulates at a 3.4 MHz processor clock rate (less than 1,000x slowdown over real-time). In aggregate, this FireSim instantiation simulates 4,096 cores and 16 TB of memory, runs ˜14 billion instructions per second, and harnesses 12.8 million dollars worth of FPGAs—at a total cost of only ˜$100 per simulation hour to the user. We present several examples to show how FireSim can be used to explore various research directions in warehouse-scale machine design, including modeling networks with high-bandwidth and low-latency, integrating arbitrary RTL designs for a variety of commodity and specialized datacenter nodes, and modeling a variety of datacenter organizations, as well as reusing the scale-out FireSim infrastructure to enable fast, massively parallel cycle-exact single-node microarchitectural experimentation.**

***Keywords*-Data centers; Computer simulation; Field programmable gate arrays; Computer networks; Distributed computing; Performance analysis; Scalability; Computer architecture**

I. Introduction

The demand for ever more powerful warehouse-scale computers (WSCs) continues to grow, to support new compute-intensive applications deployed on billions of edge devices as well as conventional computing workloads migrating to the cloud. While the first few generations of WSCs were built with standard servers, hardware trends are pushing datacenter architects towards building warehouse-scale machines that are increasingly specialized and tightly integrated [1]–[3].

The end of general-purpose processor performance scaling has pushed cloud providers towards increased specialization, through custom silicon (e.g. Google's TPU [4]), FPGAs (e.g. Microsoft's Brainwave FPGA-based deep learning serving platform [5]), or GPUs (e.g. Amazon's G3 instances). Datacenter network performance has continued to scale, in stark contrast to the stagnation of individual server performance. Datacenter operators are currently deploying Ethernet networks with 100s of Gbit/s of bandwidth and latencies below 10s of microseconds. On the horizon is the potential for silicon photonic networks to push Terabit-per-second bandwidths straight to server processor dies [6]. New memory technologies, like 3D XPoint and HBM, also have the potential to fill interesting gaps in the datacenter memory hierarchy, but also further deepen and complicate the memory hierarchy, requiring detailed evaluation at scale. A large number of academic and industry groups have also pushed towards *disaggregated* datacenter architectures that combine all of these trends by splitting resources, including CPUs, high-performance storage, and specialized compute across a high-bandwidth, low-latency datacenter fabric [3], [7]–[13]. Following these hardware trends and the expectations of modern web-scale service users, application and systems framework developers are also beginning to expect the ability to deploy fine-grained tasks, where task runtime and latency expectations are measured in microseconds [14].

These trends push the boundaries of hardware-software co-design at-scale. Architects can no longer simply simulate individual nodes and leave the issues of scale to post-silicon measurement. Additionally, the introduction of custom silicon in the cloud means that architects must model emerging hardware, not only well-understood processor microarchitectures. Hardware-software co-design studies targeting next-generation WSCs are hampered by a lack of a scalable and performant simulation environment. Using microarchitectural software simulators modified to scale out [15], [16] is fundamentally bottlenecked by the low simulation speeds (5–100 KIPS) of the underlying single-server software simulator. Fast custom-built simulation hard-

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00014

ware has been proposed [17], but is difficult to modify and expensive ($100k+) to acquire, which limits access by most academic and industrial research teams.

In this work, we present FireSim[1,2], an open-source, cycle-exact, FPGA-accelerated simulation framework that can simulate large clusters, including high-bandwidth, low-latency networks, on a public-cloud host platform. Individual nodes in a FireSim simulation are automatically derived from synthesizable RTL and run realistic software stacks, including booting Linux, at 10s to 100s of MHz. High-performance C++ switch models coordinate simulation globally and provide clean abstractions that hide the host system transport to allow users to define and experiment with their own switching paradigms and link-layer protocols. FireSim also automates the construction of a scale-out simulation—users define the network topology and number and types of server blades in the system being simulated, and FireSim builds and deploys high-performance simulations on Amazon EC2 F1 instances. Users can then treat the simulated nodes as if they were part of a real cluster—simulated datacenter nodes are visible on the network and can be accessed via SSH to run software while collecting performance data that is cycle-exact. Thus, a FireSim user is isolated from the complications of running FPGA-accelerated simulations. They write RTL (which can later be synthesized with CAD tools and possibly taped out) to customize their datacenter blades, C++ code to customize switch designs, and specify the topology and link characteristics of their network simulation to the simulation manager, which then builds and deploys a simulation. Only RTL changes require re-running FPGA synthesis—network latency, bandwidth, network topology, and blade selection can all be configured at runtime.

Section II describes recent hardware trends that allow us to build a fast, usable, and cost-effective hardware simulation environment. Section III describes the FireSim simulation environment, including the target designs FireSim is capable of simulating, and our high-performance network simulation infrastructure. Section IV validates data collected by software in FireSim simulations against parameters set in the FireSim configuration, as well as reproducing scale-out system phenomena from recent literature. Section V discusses simulation performance and scale, including a 1024-node simulation. Section VI describes some early work in warehouse-scale hardware-software co-design that uses the FireSim simulation platform and discusses preliminary results. Section VII discusses prior work in the area of scale-out system simulation and contrasts it with FireSim. Finally, Section VIII outlines ongoing work to improve FireSim performance and features.

[1]https://fires.im
[2]https://github.com/firesim

II. FPGAs in the Public Cloud

Architects experience many challenges when building and using FPGA-accelerated simulation platforms. FPGA platforms are unwieldy, especially when compared to software simulators that run on commodity compute platforms. Traditional FPGA platforms are constrained by high prices, individual platform quirks that make reproducibility difficult, the need to provision for maximum utilization at time of initial purchase, and long build times, where parallelism is limited by the number of build licenses and build servers available. Even when FPGA pricing is insignificant to a project, building a custom rack of large FPGAs requires significant systems management and operations experience and makes it extremely difficult to share and reproduce research prototypes.

But recently, many cloud providers have begun integrating FPGAs into their cloud services, including Amazon [18], Microsoft [19], [20], Huawei [21], and Alibaba [22]. In particular, Amazon makes FPGAs available as part of its *public* cloud offering, allowing developers to directly design FPGA-based applications that run in the cloud. Using an FPGA-enabled public cloud platform such as EC2 F1 addresses many of the traditional issues with FPGA-based hardware simulation platforms and provides many of the same benefits to computer architects that the cloud brought to distributed systems builders. At the dawn of the cloud era, systems researchers identified several changes to the economics of obtaining compute: (1) the new illusion of infinite computing resources, (2) the elimination of up-front commitment towards hardware resources, and (3) the ability to scale resources on-demand [23]. Given that prices of large FPGAs are much higher than even the most expensive general-purpose-compute servers, these advantages are magnified for developers and users of FPGA-based simulation platforms.

Since EC2 F1 is a relatively recent offering, we summarize some of the key features of the service to explain why it forms a natural platform on which to build the scalable FireSim environment. Amazon's EC2 F1 offering provides two new EC2 instance types, `f1.2xlarge` and `f1.16xlarge`, which consist of a powerful host instance (8 or 64 vCPUs, 122 or 976 GB of DRAM, 10 or 25 Gbit/s networking) attached to 1 or 8 Xilinx Virtex UltraScale+ FPGAs over PCIe. Furthermore, each FPGA contains 64 GB of DRAM onboard across 4 channels, making it an ideal platform for prototyping servers. The ability to provision any number of these instances on-demand and distribute work to them provides scalability. Section III-B3 describes FireSim's ability to automate the mapping and deployment of a simulation across a large number of `f1.2xlarge` and `f1.16xlarge` instances.

Amazon also provides an "FPGA Developer AMI", an OS image that contains all of the tooling and licenses

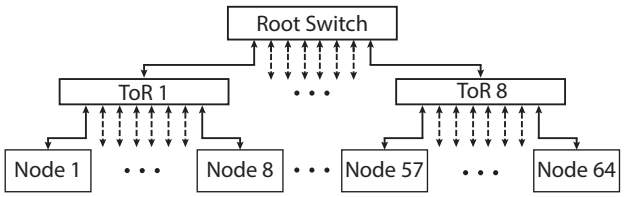

Figure 1. Target view of the 64-node topology simulated in Figure 2.

necessary to produce FPGA images for F1 instances pre-installed. As with FPGAs themselves, users can now scale to an essentially unlimited number of FPGA synthesis/P&R machines, making it possible to parallelize builds for design-space exploration and for constructing heterogeneous cluster simulations. Section III-B3 describes the FireSim infrastructure that can automatically distribute FPGA image builds based on a set of supplied node configurations.

In addition to F1 instances, FireSim also uses `m4.16xlarge` instances, which are "standard" EC2 instances that support high-performance (25 Gbit/s) networking. FireSim uses these instances to run aggregation and root switch models. All together, by taking advantage of the scalability of a cloud FPGA platform, we demonstrate the ability to automatically generate, deploy, and simulate a cluster of 1024 quad-core server nodes (for a total of 4096 cores) interconnected by a 200 Gbit/s network with 2 μs latency at 3.4 MHz. In aggregate, this simulation runs ˜14 billion instructions per second and harnesses 12.8 million dollars worth of FPGAs, at a total cost of only $100 per simulation hour to the user with no upfront capital expenditure. Section V details this example FireSim instantiation.

III. FireSim

FireSim models a target system containing a collection of server blades connected by some form of network. The target server blades are modeled using FAME-1 models [24] automatically derived from the RTL of the server SoCs and mapped onto FPGA instances, while the target network is modeled with high-performance, cycle-by-cycle C++ switch models running on host server instances. These two target components are interconnected by a high-performance simulation token transport that models target link characteristics and abstracts away host platform details, such as PCIe communication and host-to-host Ethernet communication. Figures 1 and 2 show the target topology and target-to-host mapping respectively for a 64-node simulation with 8 top-of-rack (ToR) switches and one root switch, which we use as an example throughout this section.

A. Server Blade Simulation

1) Target Server Design: FireSim compute servers are derived from the Rocket Chip SoC generator [26], which is an SoC generation library written in Chisel [27]. Rocket

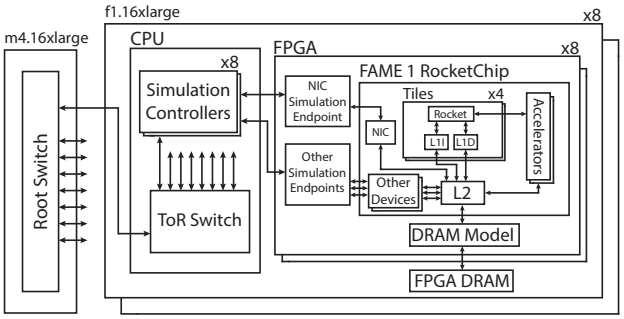

Figure 2. Example mapping of a 64-node simulation to EC2 F1 in FireSim.

Table I
SERVER BLADE CONFIGURATION.

Blade Component	RTL or Model
1 to 4 RISC-V Rocket Cores @ 3.2 GHz	RTL
Optional RoCC Accel. (Table II)	RTL
16 KiB L1I$, 16 KiB L1D$, 256 KiB L2$	RTL
16 GiB DDR3	FPGA Timing Model
200 Gbit/s Ethernet NIC	RTL
Disk	Software Model

Chip can produce Verilog RTL for a complete processor system, including the RISC-V Rocket CPU, L1 and L2 caches, custom accelerators (Table II), and I/O peripherals. Table I shows the Rocket Chip configurations we use throughout this work. When we refer to a particular frequency f for Rocket Chip, for example 3.2 GHz in Table I, this implies that all models that require a notion of target time in the simulation (e.g., the network) assume that 1 cycle is equivalent to $1/f$ seconds. The "FAME-1 Rocket Chip" box in Figure 2 provides a sample block diagram of a Rocket Chip server node.

2) Target Server Network Interface Controller: To build complete server nodes, we add two new peripherals to the Rocket Chip SoC. The first is a network interface controller (NIC) written in Chisel that exposes a top-level network interface on the SoC. The design of the NIC is shown in Figure 3. The NIC sends and receives Ethernet packets to/from the network switch. Recent papers have called for CPU and NIC to be integrated on the same die in order to decrease communication latency [28]. Our NIC implements this approach and connects directly to the on-chip network of the Rocket Chip SoC through the TileLink2 interconnect [29]. This allows the NIC to directly read/write data in/out of the shared L2 on the server SoC (Figure 2).

Table II
EXAMPLE ACCELERATORS FOR CUSTOM BLADES.

Accelerator	Purpose
Page Fault Accel.	Remote memory fast-path (Section VI)
Hwacha [25]	Vector-accelerated compute (Section VIII)
HLS-Generated	Rapid custom scale-out accels. (Section VIII)

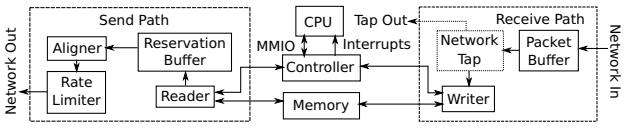

Figure 3. Network Interface Controller (NIC) design.

The NIC is split into three main blocks: the controller, the send path, and the receive path (Figure 3). The controller in the NIC is responsible for communicating with the CPU. It holds four queues: send request queue, receive request queue, send completion queue, and receive completion queue. The queues are exposed to the processor as memory-mapped IO registers. To send a packet out through the NIC, the CPU writes the memory address and length of the packet to the send request queue. To receive a packet from the NIC, the CPU writes the address of the receive buffer to the receive request queue. When the send/receive paths finish processing a request, they add an entry to their respective completion queues. The NIC also has an interrupt line to the CPU, which it asserts when a completion queue is occupied. The interrupt handler then reads the completion entries off of the queue, clearing the interrupt.

The send path in the NIC begins with the reader module, which takes a send request from the controller and issues read requests for the packet data from memory. Responses from memory are forwarded to the next stage, the reservation buffer. The reader sends a completion signal to the controller once all the reads for the packet have been issued.

The reservation-buffer module stores data read from memory while it waits to be transmitted through the network interface. Responses from the memory bus can arrive out-of-order, so the reservation buffer also performs some reordering so that the data is sent to the next stage in-order.

After the reservation buffer comes the aligner, which allows the send path to handle unaligned packets. The interface to the memory system is 64 bits wide, so the reader can only read data at an eight-byte alignment. If the starting address or ending address of the packet is not a multiple of eight, some extra data before or after the packet will be read. The aligner shifts data coming from the buffer so that the extra data is omitted and the first byte of the packet will be the first byte delivered to the destination.

The final module in the send path is the rate limiter, which allows NIC bandwidth to be limited at runtime using a token-bucket algorithm. Essentially, the limiter holds a counter that is decremented every time a network flit is sent and incremented by k every p cycles. Flits can be forwarded from input to output so long as the count is greater than zero. This makes the effective bandwidth $\frac{k}{p}$ times the unlimited rate. The values k and p can be set at runtime, allowing simulation of different bandwidths without resynthesizing the RTL. Unlike external throttling of requests, this internal throttling appropriately backpressures the NIC, so it behaves as if it actually operated at the set bandwidth.

The receive path begins at the network input with a packet buffer. Since we cannot back-pressure the Ethernet network, the buffer must drop packets when there is insufficient space. Packets are only dropped at full-packet granularity so that the operating system never sees incomplete packets.

The writer module takes packet data from the buffer and writes it to memory at the addresses provided by the controller. The writer sends a completion to the controller only after all writes for the packet have retired.

To interface between user-space software and the NIC, we wrote a custom Linux driver to allow any Linux-compatible networking software to run on FireSim.

The top-level interface of the NIC connects to the outside world through a FAME-1 decoupled interface—each cycle, the NIC must receive a token and produce a token for the simulation to advance in target time. Section III-B details cycle-accurate packet transport outside of the NIC.

3) Target Server Block Device Controller: We also add a block device controller to the server blades simulated in FireSim to allow booting of custom Linux distributions with large root filesystems. The block device controller contains a frontend module that interfaces with the CPU and one or more trackers that move data between memory and the block device. The frontend exposes Memory-Mapped I/O (MMIO) registers to the CPU, through which it can set the fields needed for a block device request. To start a block device transfer, the CPU reads from the allocation register, which sends a request to one of the trackers and returns the ID of the tracker. When the transfer is complete, the tracker sends a completion to the frontend, which records the ID of the tracker in the completion queue and sends an interrupt to the CPU. The CPU then reads from the completion queue and matches the ID with the one it received during allocation. The block device is organized into 512-byte sectors, so the controller can only transfer data in multiples of 512 bytes. The data does not need to be aligned at a 512-byte boundary in memory, but it does need to be aligned on the block device.

4) Cycle-Exact Server Simulations from RTL: We use the FAME-1 [24] transforms provided by the MIDAS/Strober frameworks [30], [31] to translate the server designs written in Chisel into RTL with decoupled I/O interfaces for use in simulation. Each target cycle, the transformed RTL on the FPGA expects a token on each input interface to supply input data for that target cycle and produces a token on each output interface to feed to the rest of the simulated environment. If any input of the SoC does not have an input token for that target cycle, simulation stalls until a token arrives. This allows for timing-accurate modeling of I/O attached to custom RTL on the FPGA. To provide a cycle-accurate DRAM model for our target servers, we use an existing synthesizable DRAM timing model provided with MIDAS,

attached directly to each host FPGA's on-board DRAM, with parameters that model DDR3. Other I/O interfaces (UART, Block Device, NIC) communicate with a software driver ("simulation controller" in Figure 2) on the host CPU core, which implements both timing and functional request handling (e.g. fetching disk blocks). Since in this work we are primarily interested in scaling to large clusters and network modeling, we focus on the implementation of the network token-transport mechanism used to globally coordinate simulation target time between the FAME-1-transformed server nodes.

5) Improving Scalability and Utilization: In addition to the previously described configuration, FireSim includes an additional "supernode" configuration, which simulates multiple complete target designs on each FPGA to provide improved utilization and scalability.

The basic target design described above utilizes only 32.6% of the FPGA LUT resources and one of the four memory channels. Only 14.4% of this utilization was for our custom server-blade RTL. As a result, the supernode configuration packs four simulated nodes on each FPGA, increasing FPGA LUT utilization for server blades to approximately 57.7% of the FPGA logic resources, and total FPGA LUT utilization to 76%. This optimization reduces the cost of simulating large clusters, at the cost of multiplexing network token transfer for four nodes over a single PCIe link. Section V-C describes how supernodes support the simulation of a large cluster with 1024 nodes.

This type of scalability is particularly important when attempting to identify datacenter-level phenomena, and reduces the dependency on the cloud-provider's instantaneous FPGA instance capacity. Furthermore, this capability allows for the simulation of smaller target network topologies, such as connecting a ToR switch to 32 simulated nodes, without an expensive host-Ethernet crossing for token transport.

B. Network Simulation

1) Target Switch Modeling: Switches in the target design in FireSim are modeled in software using a high-performance C++ switching model that processes network flits cycle-by-cycle. The switch models have a parametrizable number of ports, each of which interact with either a port on another switch or a simulated server NIC on the FPGA. Port bandwidth, link latency, amount of buffering, and switching latency are all parameterized and runtime-configurable.

The simulated switches implement store-and-forward switching of Ethernet frames. At ingress into the switch, individual simulation tokens that contain valid data are buffered into full packets, timestamped based on the arrival cycle of their last token, and placed into input packet queues. This step is done in parallel using OpenMP threads, with one thread per-port. The timestamps are also incremented by a configurable minimum switching latency to model the minimum port-to-port latency of datacenter switches. These timestamps are later used to determine when a packet can be released to an output buffer. A global switching step then takes all input packets available during the switching round, pushes them through a priority queue that sorts them on timestamp, and then drains the priority queue into the appropriate output port buffers based on a static MAC address table (since datacenter topologies are relatively fixed). In this step, packets are duplicated as necessary to handle broadcasts. Finally, in-parallel and per-port, output ports "release" packets to be sent on the link in simulation token form, based on the switch's notion of simulation time, the packet timestamp, and the amount of available space in the output token buffer. In essence, packets can be released when their release timestamp is less than or equal to global simulation time. Since the output token buffers are of a fixed size during each iteration, congestion is automatically modeled by packets not being able to be released due to full output buffers. Dropping due to buffer sizing and congestion is also modeled by placing an upper bound on the allowable delay between a packet's release timestamp and the global time, after which a packet is dropped. The switching algorithm described above and assumption of Ethernet as the link-layer is not fundamental to FireSim—a user can easily plug in their own switching algorithm or their own link-layer protocol parsing code in C++ to model new switch designs.

2) High-performance Token Transport: Unlike simulating "request-response" style channels (e.g. AXI channels) in cycle-accurate simulation platforms, the decoupled nature of datacenter networks introduces new challenges and prevents many optimizations traditionally used to improve simulation performance, especially when simulated nodes are distributed as in FireSim. In this section, we describe our network simulation and how we map links in simulation to the EC2 F1 host platform.

From the target's view, endpoints on the network (either NICs or ports on switches) should communicate with one another through a link of a particular latency and bandwidth. On a simulated link, the fundamental unit of data transferred is a token that represents one target cycle's worth of data. Each target cycle, every NIC expects one input token and produces one output token in response. Each port on every switch also behaves in the same way, expecting one input token and generating one output token every cycle. For a link with link latency of N cycles, N tokens are always "in-flight" on the link at any given time. That is, if a particular network endpoint issues a token at cycle M, the token arrives at the other side of the link for consumption at cycle $M+N$.

Network tokens in FireSim consist of two components: the target payload field and a "last" metadata field that indicates to the transport that a particular token is the end of a packet, without having to understand a particular link-layer protocol. In the case of our Ethernet model, the target payload field

contains two fields: the actual data being transmitted by the target design during that cycle and a valid signal to indicate that the input data for this cycle is legitimate data (as opposed to an empty token, which corresponds to a cycle where the endpoint received nothing from the network). To simulate the 200 Gbit/s links we use throughout this paper, the width of the data field is set to 64 bits, since we assume that our simulated processor frequency is 3.2 GHz. In a distributed simulation as in FireSim, different host nodes are decoupled and can be executing different target cycles at the same time, but the exchange of these tokens ensures that each server simulation computes each target cycle deterministically, since all NICs and switch ports are connected to the same network and do not advance unless they have input tokens to consume.

In a datacenter topology, there are two types of links that need to be modeled: links between a NIC and a switch port and links between two switch ports on different switches. Since we model switches in software and NICs (and servers) on FPGAs, these two types of links map to two different types of token transport. Transport between NICs and switch models requires two hops: a token must first cross the PCIe interface to an individual node's simulation driver, then be sent to a local switch model through shared memory or a remote switch model over a socket.

As discussed in prior work on distributed software-based cluster simulation [16], batching the exchange of these tokens improves host bandwidth utilization and hides the data movement latency of the host platform—both PCIe and Ethernet in the case of EC2 F1. Tokens can be batched up to the target's link latency, without any compromise in cycle accuracy. Given that the movement of network tokens is the fundamental bottleneck of simulation performance in a FireSim simulation, we always set our batch size to the target link latency being modeled.

We utilize three types of physical transports to move tokens. Communication between FPGAs and host CPUs on EC2 F1 happens over PCIe. For high-performance token transport, we use the Amazon Elastic DMA (EDMA) interface to move batches of tokens (one link latency's worth) between token queues on the FPGA and the simulation controller on the host. Transport between the simulation controller and a ToR switch or between two switches can be done either through a shared memory port (to effectively provide zero-copy transfer between endpoints) or a TCP socket when endpoints are on separate nodes. Since we are focused on simulating low-latency target networks, our primary bottleneck in simulation is the host latency of these transport interfaces. Since latency is the dominant factor, we also do not employ any form of token compression. This also means that simulation performance is stable (workload independent), apart from the cost of the switching phase inside a switch. We explore the trade-off between target-link latency and simulation performance in Section V.

To provide a concrete example to tie together the physical and logical layers in link modeling and token transport, let us trace the progress of a packet moving between two servers (labeled A and B) connected by a single ToR switch in simulation. For simplicity, we assume that the packet consists of only one token, however this description naturally extends to longer packets. We assume that links have a latency of l cycles and that we batch token movement by always moving l tokens at a time across host PCIe/network links. We also assume that the network is completely unloaded and that switches have a minimum port-to-port latency of n cycles. Within a server, cycle-accuracy is maintained by virtue of directly running FAME-1-transformed RTL on an FPGA, so we focus on the path between the top-level I/O of the two server NICs that are communicating:

1. Suppose that all simulated components (the switch and the two servers) begin at cycle $t = 0$, with each input token queue initialized with l tokens.

2. Each simulated component can independently advance to cycle $t = l$ by consuming the input tokens. Suppose that server A's NIC has a valid packet to send at cycle $t = m < l$.

3. This packet is written into the output token buffer in the NIC Simulation Endpoint (see Figure 2) on the FPGA at index m. When server A has completely filled the buffer, the buffer is copied first to the software simulation controller over PCIe and then to the ToR switch via shared memory.

4. In the meantime, since the switch was also initially seeded with l tokens per port, its simulation time has also advanced to cycle l, before it receives this buffer.

5. Now, when the switch "ticks" cycle-by-cycle through the newly received buffer and reaches simulation time $t = l + m$, it will "receive" the packet sent by server A.

6. Next, the switch will write the packet to an output buffer after a minimum switching delay, n, at cycle $t = l + m + n$. For the sake of argument, assume $m + n < l$. Then, this packet will be written to the next buffer sent out by the switch.

7. In the meantime, server B will have received two rounds of input buffers, so it will have advanced to cycle $2l$ when it receives the buffer containing the packet. Since the packet is at index $m + n$ in this buffer, it will arrive at the input of the server's NIC at cycle $2l + m + n$, or two times the link latency, plus the switching latency, plus the time at which server A sent the packet.

This decoupled simulation technique allows us to scale easily to large clusters. Furthermore, simulations generally map well onto our host platform, since we are in essence simulating large target clusters on large host clusters. Unlike software simulators, the power of our host CPUs can be dedicated to fast switching, while FPGAs handle the computationally expensive task of modeling the low-level details of the processors and the rest of the server blades.

3) Deploying/Mapping Simulation to EC2 F1: At this point, we have outlined each component necessary to build a

```
m4_16xlargeIPs = [...]
f1_16xlargeIPs = [...]
root = SwitchNode()
level2switches = [SwitchNode() for x in range(8)]
servers = [[ServerNode("QuadCore")
                                for y in range(8)]
                                for x in range(8)]
root.add_downlinks(level2switches)

for l2switchNo in range(len(level2switches)):
    switch = level2switches[l2switchNo]
    servers = servers[l2switchNo]
    switch.add_downlinks(servers)
```

Figure 4. Example Simulation Configuration. This instantiates a simulation of the cluster topology shown in Figure 1 with quad-core servers.

large-scale cluster simulation in FireSim. However, without automation, the task of stitching together all of these components in a reliable and reproducible way is daunting. To overcome this challenge, the FireSim infrastructure includes a simulation manager that automatically builds and deploys simulations given a programmatically defined datacenter topology. That is, a user can write a configuration in Python that describes a particular datacenter topology and server types for each server blade as shown in Figure 4. The FireSim cluster manager takes this configuration and automatically runs the desired RTL through the FPGA build flow and generates the high-performance switch models and simulation controllers with the appropriate port implementations (shared memory, socket, PCIe transport). In particular, based on the given topology, simulated servers are automatically assigned MAC and IP addresses and the MAC switching tables in the switch models are automatically populated for each switch in the simulation. Once all component builds are complete, the manager flashes FPGAs on each F1 instance with the desired server configurations, deploys simulations and switch models as described by the user, and finally boots Linux (or other software) on the simulated nodes. At the root switch, a special port can be added to the network that allows for direct ingress into the simulated datacenter network over SSH. That is, a user can *directly ssh* into the simulated system from the host machine and treat it as if it were a real cluster to deploy programs and collect results. Alternatively, a second layer of the cluster manager allows users to describe jobs that *automatically* run on the simulated cluster nodes and *automatically* collect result files and host/target-level measurements for analysis outside of the simulation. For example, the open release of FireSim includes reusable workload descriptions used by the manager to automatically run various versions of SPECint, boot other Linux distributions such as Fedora, or reproduce the experiments described later in this paper, among others.

IV. Validation

In this section, we validate the FireSim simulation environment, in particular our high-performance, cycle-accurate network simulation, using several benchmarks.

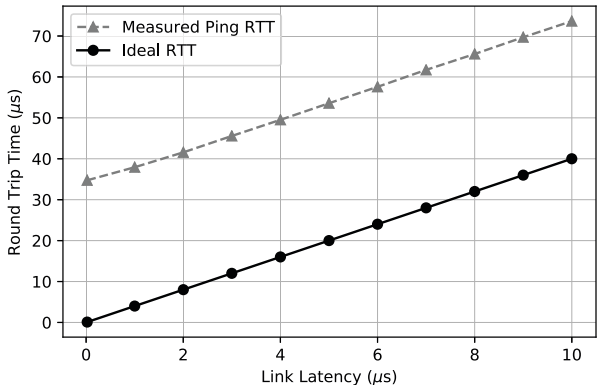

Figure 5. Ping latency vs. configured link latency.

A. Network Latency Benchmarking

We benchmark end-to-end latency between simulated nodes by collecting samples from the ping utility in Linux, while varying the target link latency set in simulation. This experiment boots Linux on a simulated 8-node cluster connected by a single ToR switch and collects the results of 100 pings between two nodes. We ignore the first ping result on each boot, since this includes additional latency for an ARP request for the node to find the MAC address of the node being pinged. We then vary the configured link latency for the simulation and record the average RTT reported by ping. Figure 5 shows the results of this benchmark. The bottom line represents the "Ideal" round trip time (link latency times four, plus a fixed port-to-port switching latency of 10 cycles times two). As expected for a correct network simulation, the measured results parallel the ideal line, with a fixed offset that represents overhead in the Linux networking stack and other server latency. The $\approx$34 μs overhead we observe matches results widely reported in the OS literature [28].

B. Network Bandwidth Benchmarking: `iperf3` on Linux

Next, we run `iperf3`, a standard network benchmarking suite, on top of Linux on the simulated nodes and measure the bandwidth between two nodes connected by a ToR switch. In this benchmark, we measure an average bandwidth over TCP of 1.4 Gbit/s. While this is much lower than our nominal link bandwidth of 200 Gbit/s, we suspect that the bulk of this mismatch is due to the relatively slow single-issue in-order Rocket processor running the network stack in software on an immature RISC-V Linux port with our Linux network driver.

C. Bare-metal node-to-node bandwidth test

To separate out the limits of the software stack from our NIC hardware and simulation environment, we implemented

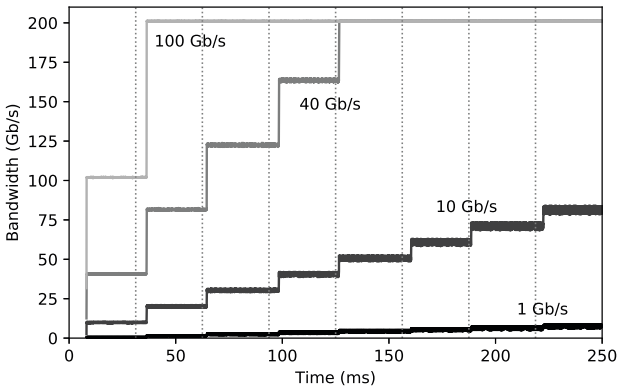

Figure 6. Multi-node bandwidth test. Dotted grey lines mark the entry points of individual senders.

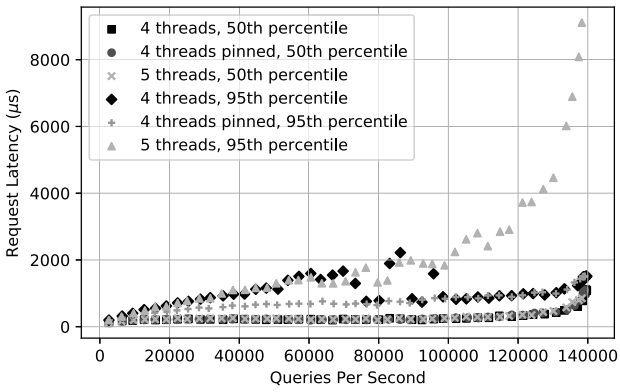

Figure 7. Reproducing the effect of thread imbalance in memcached on tail latency.

a bare-metal bandwidth benchmarking test that directly interfaces with the NIC hardware. The test constructs a sequence of Ethernet packets on one node and sends them at maximum rate to another node. On the other node, we verify that the received data is correct and then send an acknowledgement to the sender to indicate test completion. With this test, a single NIC is able to drive 100 Gbit/s of traffic onto the network, confirming that our current Linux networking software stack is a bottleneck.

D. Saturating Network Bandwidth

Because our current target server-blade designs are not capable of saturating the 200 Gbit/s network links that we are modeling even using bare-metal programs, we demonstrate the ability to saturate our network simulation by running concurrent streams across a cluster. We simulate a 16-node cluster with two ToR switches and one root switch. Each server attached to the first ToR switch sends data to the corresponding server on the other ToR switch (through the root switch). We then measure the aggregate bandwidth over time at the root switch. Figure 6 shows the results of this benchmark. We performed this test with different rate limits set on the sending nodes to simulate the standard Ethernet bandwidths of 1, 10, 40, and 100 Gbit/s. Each sender node starts a fixed unit of time after the previous sender began, so that traffic at the ToR switch ramps up over time and eventually saturates in the cases of 40 and 100 Gbit/s senders.

In the 100 Gbit/s test, each sender is able to saturate the full bandwidth of its link, so the total bandwidth at the switch jumps up by 100 Gbit/s for each sender until it saturates at 200 Gbit/s after the second sender enters. In the 40 Gbit/s test, the total bandwidth increases by 40 Gbit/s for each additional sender until it saturates after five senders enter. In the 1 and 10 Gbit/s tests, the root switch's bandwidth is not saturated and instead maxes out at 8 and 80 Gbit/s, respectively.

E. Reproducing `memcached` QoS Phenomena from Deployed Commodity Clusters in FireSim

As an end-to-end validation of FireSim running a realistic datacenter workload, we run the memcached key-value store and use the mutilate distributed memcached load-generator from Leverich and Kozyrakis [32] to benchmark our simulated system. While much simulation work has focused on reproducing well-known phenomena like the long-latency tail, we go further to validate a finer-grained phenomenon: thread imbalance in memcached servers when memcached is instructed to use more threads than the number of cores in the system. Reproducing this result involves interaction between the core microarchitecture, operating system, and network. Under thread imbalance, a sharp increase in tail latency was shown while median latency was relatively unchanged [32]. To replicate this result, we simulate an 8-node cluster in FireSim interconnected by a 200 Gbit/s, 2 μs latency network, where each simulated server has 4 cores. We provision one server in the simulation as a memcached host. We also cross-compile mutilate and its dependencies to run on RISC-V servers and run it on the remaining seven simulated blades to generate a specified aggregate load on the memcached server. On the serving node, we configure memcached to run with either 4 or 5 threads and report median and tail (95th-percentile) latencies based on achieved queries per second. Figure 7 shows the results of this experiment. As expected from the earlier work [32], we observe thread imbalance when running with 5 threads—the tail latency is significantly worsened by the presence of the extra thread, while median latency is essentially unaffected. In the 4-thread case, we also notice an interesting phenomenon at low to medium load—the 95th percentile line for 4-threads tracks the 5-thread case, until a certain load is reached. We suspect that this phenomenon is due to poor thread placement in that region of QPS, even when the number of threads equals the number of cores in the system [32]. To confirm our suspicion, we run an

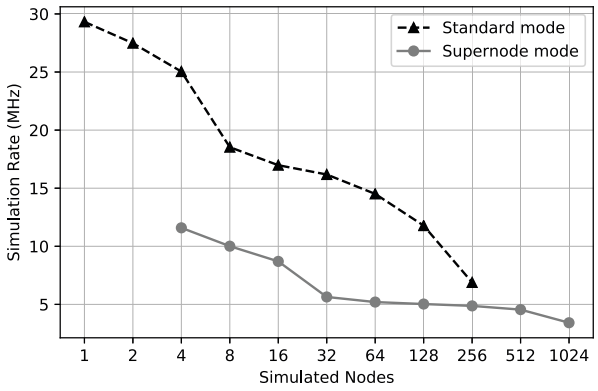

Figure 8. Simulation rate vs. # of simulated target nodes.

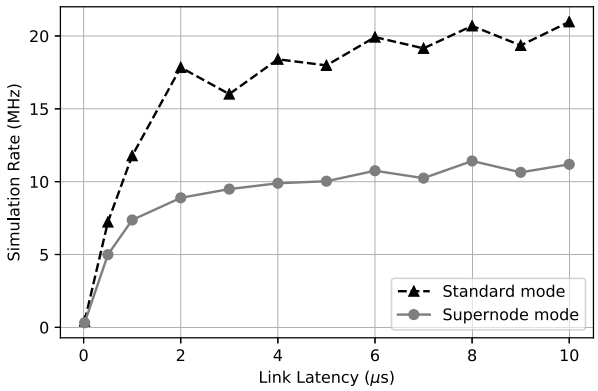

Figure 9. Simulation rate vs. simulated network link latency.

additional experiment where we drive the same load, but run a memcached server with 4 threads and pin one thread to each of the four cores in the processor ("4 threads pinned" in Figure 7). In this case, we again see the same curve for 50th-percentile latency. However, the 95th-percentile curve is smoothed-out relative to the no-pinning 4-thread 95th-percentile curve, but overlaps it at high load, where we suspect that the Linux scheduler automatically places threads as if they were pinned one-to-a-core, even in the no-pinning case.

V. Simulation Performance

In this section, we analyze the performance of FireSim simulations, using a variety of target configurations.

A. Performance vs. target scale

To show the overhead of token-based synchronization of all simulated nodes in clusters interconnected by a simulated 2 μs, 200 Gbit/s network as a function of cluster size, we run a benchmark that boots Linux to userspace, then immediately powers down the nodes in the cluster and reports simulation rate. This process does not exercise the network from the target perspective. However, as we do not yet perform any form of token compression, the number of tokens exchanged on the host platform is exactly the same as if there were network traffic (empty tokens are being moved between the target network endpoints). The only component of simulation overhead not included in this measurement is the overhead of packet switching inside the switch when there is traffic on the network. However this is primarily a function of the load on a single switch, rather than simulation scale. This benchmark shows the overhead of distributing simulations, first between FPGAs on one instance, then between FPGAs in different instances. Figure 8 shows the results of this benchmark, both for "standard" and "supernode" FPGA configurations.

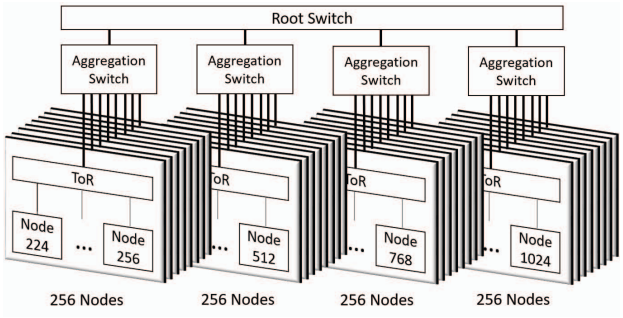

Figure 10. Topology of 1024-node datacenter simulation.

B. Performance vs. target network latency

As prior work has shown, moving tokens between distributed simulations is a significant bottleneck to scaling simulations [16]. Furthermore, as target link latency is decreased, simulation performance also decreases proportionally due to the loss of benefits of request batching. Throughout this work, we focus on a 2 μs link latency when benchmarking, since we consider this to be similar to latencies desired in experiments. Figure 9 shows simulation performance as a function of varying target link latency. As expected, simulation performance improves as the batch size of tokens increases.

C. Thousand-Node Datacenter Simulation

To demonstrate the scale achievable with FireSim, we run a simulation that models 1024×3.2 GHz quad-core nodes, with 32 top-of-rack switches, 4 aggregation switches, and

Table III
1024-node `memcached` experiment latencies and QPS.

	50th Percentile (μs)	95th Percentile (μs)	Aggregate Queries-Per-Second
Cross-ToR	79.26	128.15	4,691,888.0
Cross-aggregation	87.10	111.25	4,492,745.6
Cross-datacenter	93.82	119.50	4,077,369.6

one root switch, all interconnected by a 2 μs, 200 Gbit/s network and arranged in a tree-topology, at a simulation rate of 3.42 MHz. This design represents a more realistic target design point than the simpler design used as an example in Section III, since we make use of FireSim's "supernode" feature to pack four simulated nodes per FPGA, giving a total of 32 simulated nodes attached to each ToR switch. Figure 10 shows this topology in detail. Each ToR switch has 32 downlinks to nodes and one uplink to an aggregation switch. Each aggregation switch has eight downlinks, each to one ToR switch and one uplink to the root switch. Finally, the root switch has 4 downlinks to the 4 aggregation switches in the target topology. This topology is specified to the FireSim simulation manager with around 10 lines of configuration code. More complicated topologies, such as a fat-tree, can similarly be described in the manager configuration, but we do not explore them in this work. Compared to existing software simulators, this instantiation of FireSim simulates an order of magnitude more nodes, with several orders of magnitude improved performance.

To map this simulation to EC2, we run 32 `f1.16xlarge` instances, which host ToR switch models and simulated server blades, and 5 `m4.16xlarge` instances to serve as aggregation and root-switch model hosts. The cost of this simulation can be calculated for 2 different EC2 pricing models: spot instances (bidding on unused capacity) and on-demand (guaranteed instances). To calculate the spot price of this simulation, we use the longest stable prices in recent history, ignoring downward and upward spikes. This results in a total cost of ≈$100 per simulation hour. Using on-demand instances, which have fixed instance prices, this simulation costs ≈$440 per simulation hour. Using publicly listed retail prices of the FPGAs on EC2 (≈$50K each), this simulation harnesses ≈$12.8M worth of FPGAs. We expect that users will use cluster-scale experiments to prototype systems, with datacenter-scale experiments to analyze behavior at-scale once a system is already stable at cluster-scale.

As an example experiment that runs at this scale, we use the mutilate memcached load generator (as in Section IV) to generate load on memcached servers throughout the datacenter. In each experiment, there are 512 memcached servers and 512 mutilate load generator nodes, with each load generator targeting one server. We run these in 3 different configurations: one where all requests remain intra-rack, one where all requests cross an aggregation switch (but not the root switch), and one where all requests cross the root switch, by changing which servers and load generators are paired together. Table III shows average 50th percentile latency, average 95th percentile latency, and total QPS handled across all server-client pairs for these experiments. As expected, when we jump from crossing ToR switches only to crossing aggregation switches for each request, the 50th percentile latency increases by 4 times the link latency, plus switching latency, or approximately 8 μs. A similar increase is seen in the 50th percentile latency when moving from crossing aggregation switches to crossing root switches. On the other hand, in both cases, there is no predictable change in 95th percentile latency, since it is usually dominated by other variability that masks changes in number of hops for a request. Finally, we see that number of queries per second decreases. This decrease is not as sharp as one would expect, because we limit the generated load to ≈10,000 requests per server, since we are interested primarily in showing the effects of latency rather than OS-effects or congestion.

VI. Page-Fault Accelerator

In this section, as a case study to exemplify the cross-cutting architecture projects enabled by FireSim, we show preliminary performance results for a "Page-Fault Accelerator" that removes software from the critical path of paging-based remote memory.

There have been several recent proposals to disaggregate memory in warehouse-scale computers, motivated by the increasing performance of networks, and a proliferation of novel memory technologies (e.g. HBM, NVM) [8], [10]. In a system with memory disaggregation, each compute node contains a modest amount of fast memory (e.g. high-bandwidth DRAM integrated on-package), while large capacity memory or NVM is made available across the network through dedicated memory nodes.

One common proposal to harness the fast local memory is to use it as a large cache for the remote bulk memory. This cache could be implemented purely in hardware (e.g [33], [34]), which could minimize latency but may involve complicated architectural changes and would lack OS insights into memory usage. An alternative is to manage the cache purely in software with traditional paging mechanisms (e.g. [3], [35]). This approach requires no additional hardware, can use sophisticated algorithms, and has insight into memory usage patterns. However, our experiments show that even when paging to local memory, applications can be slowed significantly due to the overhead of handling page faults, which can take several microseconds and pollute the caches. In this case study, we propose a hybrid HW/SW cache using a new hardware device called the "page fault accelerator" (PFA).

The PFA works by handling the latency-critical page faults (cache-miss) in hardware, while allowing the OS to manage latency-insensitive (but algorithmically complex) evictions asynchronously. We achieve this decoupling with a queue of free page frames (*freeQ*) to be used by the PFA for fetched pages, and a queue of new page descriptors (*newQ*) that the OS can use to manage new page metadata. Execution then proceeds as follows:

- The OS allocates several page frames and pushes their addresses onto the *freeQ*. The OS experiences memory pressure and selects pages to evict to remote memory.

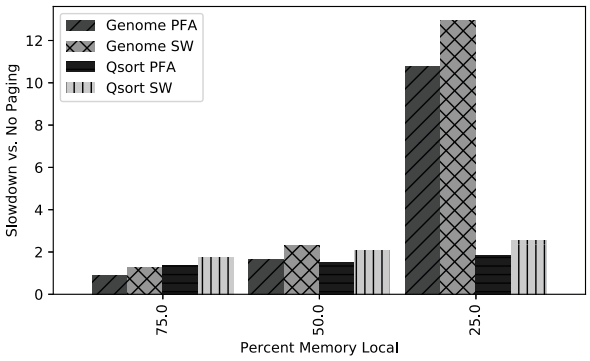

Figure 11. Hardware-accelerated vs. software paging.

It marks them as "remote" in the page tables and then provides them to the PFA for eviction.

- The application attempts to access a remote page, triggering the PFA to request the page from remote memory and place it in the next available free frame. The application is then resumed.
- Some time later (either through a background daemon, or through an interrupt due to full queues) the OS pops all new page descriptors off the *newQ* and records the (now local) pages in its metadata. The OS typically provides more free frames at this time.

We implemented the PFA in Rocket Chip and modified Linux to use it for all paging activity. For our baseline, we modified Linux to use the memory blade directly through its normal paging mechanisms (similar to Infiniswap [35]). The memory blade itself is implemented as another Rocket core running a bare-metal memory server accessed through a custom network protocol. Figure VI plots PFA performance on two benchmarks: Genome, a de-novo genome assembly benchmark that involves random accesses into a large hash table, and Qsort, a simple quicksort benchmark. Both benchmarks were tuned to have a peak memory usage of 64 MiB. Quicksort is known to have good cache behavior and does not experience significant slowdowns when swapping. Genome assembly, however, has unpredictable access patterns, leading to significant cache thrashing in low local memory configurations. In both cases, the PFA significantly reduces the overhead (by up to 1.4×). A more detailed analysis shows that while the number of evicted pages is the same in both cases, using the PFA leads to a 2.5× reduction in metadata management time on average. While the same code path is executed for each new page, the PFA batches these events, leading to improved cache locality for the OS, and fewer cache-polluting page-faults for the application. Future implementations could use a background daemon for new-page processing, further decoupling the application from new page management.

What we have presented here represents an initial design and evaluation of a disaggregated memory system. In the future, we plan to use FireSim to experiment with more sophisticated configurations. For example, we will scale the number of memory and application nodes to see how contention for network and memory resources affects overall performance. We are also investigating alternative remote memory protocols such as Gen-Z [36].

VII. Related Work

Computer architects have long used cycle-accurate simulation techniques to model machines and workloads, including servers in warehouse-scale machines. Simulation of machines at the scale of individual nodes was largely sufficient for 2010-era datacenters; the commodity Ethernet network used in most datacenters provided a natural decoupling point to draw a boundary on the effects of microarchitectural changes. Since most systems were built using commodity components, issues of scale could be measured in actual deployments and used to improve future designs. For example, in cases where scale-out workloads differed from traditional workloads in terms of fine-grained performance, performance-counter based mechanisms could be used to analyze microarchitectural performance issues on commodity servers at scale [37], [38].

Other tools for simulating rack-scale systems can broadly be divided into three categories—software-based simulators that provide flexibility but low performance, hardware-accelerated simulators that are expensive to deploy and difficult to use, but provide high-performance, and statistical models that analyze the big-picture of datacenter performance, but are not targeted towards modeling fine-grained system interactions. Below, we review this prior work.

A. Software Simulators

One approach to simulating warehouse-scale computers is to take existing cycle-accurate full-system software simulators and scale them up to support multiple nodes. One simulator that uses this approach is dist-gem5 [16], a distributed version of the popular architectural simulator gem5. This simulator networks together instances of gem5 running on different nodes by moving synchronization messages and target packets. The primary advantage of these software-based approaches is that they are extremely flexible. However, this flexibility comes at the expense of performance—these software-based solutions are several orders of magnitude slower than FPGA-accelerated simulation platforms like FireSim. Also, software models of processors are notoriously difficult to validate and calibrate against a real design [39], and do not directly provide reliable power and area numbers. By utilizing FPGAs in a cloud service, FireSim matches many of the traditional flexibility advantages of software simulators, excluding short build times. Software simulators have also traditionally had more sophisticated hardware models than FPGA-based simulators, however the recent

explosion in open-source hardware is providing realistic designs [40]–[42] that have the advantage of being truly cycle-exact and synthesizable to obtain realistic physical measurements.

Another prior software-based approach is reflected in the Wisconsin Wind Tunnel (WWT) [43] project, which used the technique of direct execution to achieve high simulation performance for individual nodes. In turn, WWT encountered similar performance bottlenecks as FireSim—network simulation has a significant impact on overall simulation performance. Follow-on work in the WWT project [44] explored the impact of several different network simulation models on the performance and accuracy of WWT. Similarly, we plan to explore more optimized network simulation models in FireSim in the future, which would trade-off accuracy vs. performance to support different user needs, using our current cycle-exact network model as the baseline. FireSim already supports the other extreme of the performance-accuracy curve—purely functional network simulation—which allows individual simulated nodes to run at 150+ MHz, while still supporting the transport of Ethernet frames between simulated nodes.

The Graphite simulator [45] takes a different software-based approach to simulating datacenters. Graphite can simulate thousands of cores in a shared-memory system at high simulation rate (as low as 41× slowdown), but only by dropping cycle accuracy and relaxing synchronization between simulated cores. Moreover, unlike full-system software simulators, Graphite only supports user applications and does not boot an OS.

A final software-based approach to simulating datacenters is to abstractly model the datacenter as a set of statistical events. This reduces simulation runtime, but sacrifices the fidelity provided by detailed microarchitectural models. This approach is used by datacenter-scale simulators like BigHouse [46] and MDCSim [47]. BigHouse models a datacenter using a stochastic queuing simulation, representing datacenter workloads as a set of tasks with a distribution of arrival and service times. The distribution is determined empirically by instrumenting deployed datacenter systems. These distributions are then used to generate a synthetic event trace that is fed into a discrete-event simulation. The simulation is distributed across multiple nodes by running a separate instance of the simulator with a different random seed on each node, then collecting the individual results into a final merged result. MDCSim separates the simulation into three layers: user, kernel, and communication. It then models specific services and systems in each layer, such as web and database servers, schedulers, and NIC, as M/M/1 queues. While these simulation tools are useful for providing the "big picture" of datacenter performance and can do so considerably faster than FireSim, FireSim instead focuses on modeling fine-grained interactions at the level of detailed microarchitecture changes between systems at-scale.

B. Hardware-accelerated Simulators

Several proprietary tools exist for hardware-accelerated system simulation, such as Cadence Palladium, Mentor Veloce, and Synopsys Zebu [48]. These systems are generally prohibitively expensive (≈millions of dollars) and thus unattainable for all but the largest industrial design groups.

Some existing projects use FPGAs to accelerate simulation of computer systems, including large multicore systems [49]. Most recently, projects in the RAMP collaboration [50] pushed towards fast, productive FPGA-based evaluation for multi-core systems [51]–[54]. However, most of these simulation platforms do not support simulation of scale-out systems. To our knowledge, the closest simulator to FireSim in this regard is DIABLO [17], [55]. Although DIABLO also uses FPGAs to simulate large scale-out systems, there are several significant differences between DIABLO and FireSim:

Automatically transformed RTL vs. Abstract Models. In DIABLO, servers are modeled using handwritten SystemVerilog abstract RTL models of SPARC cores. Authoring abstract RTL models is far more difficult than developing an actual design in RTL, and abstract RTL cannot be run through an ASIC flow to gain realistic power and area numbers. FireSim's simulated servers are built by directly applying FAME-1 transforms to the original RTL for a server blade to yield a simulator that has the exact cycle-by-cycle bit-by-bit behavior of the user-written RTL. Simulated switches in DIABLO are also abstract RTL models. In FireSim, users still write abstract switch models, but since switches are not the simulation bottleneck, switch models can be written in C++ making them considerably easier to modify. For detailed switch simulations, FireSim could be extended to also support FAME-1 transformations of real RTL switch designs.

Specialized vs. Commodity Host Platform. DIABLO uses a custom-built FPGA platform that cost ≈$100K at time of publication, excluding operation and maintenance costs and the requirement for sysops expertise. This choice of platform makes it very difficult for other researchers to use DIABLO and reproduce results. In contrast, FireSim is deployed in a public cloud environment where Amazon has made all platform-specific FPGA scripts open-source [56]. Similarly, the entire FireSim code base is open-source, which allows any user to easily deploy simulations on EC2 without the high CapEx of a DIABLO-like system. Furthermore, Amazon frequently gives substantial EC2 credit grants to academics for research purposes through their AWS Cloud Credits for Research program [57].

VIII. Discussion and Future Work

As shown in Section IV, FireSim is sufficiently mature to reproduce warehouse-scale workload performance phenomena. In this section, we outline some alternative use cases and ongoing work to further build out and improve FireSim.

Reproducible and scalable single-node experimentation. The large scale of FireSim experiments required us to build a simulation management framework to enable reliable and reproducible experimentation with a thousand nodes as described in Section III-B3. Unsurprisingly, this functionality is also immensely useful for running workloads like SPECint on single-node systems. Harnessing FireSim's ability to distribute jobs to many parallel single-node simulations, users can run the entire SPECint17 benchmark suite on Rocket Chip-like systems with *full reference* inputs, and obtain cycle-exact results in roughly one day. In the future, we plan to re-use the FireSim network simulation transport to support partitioning larger designs across many FPGAs.

The Berkeley Out-of-Order Machine (BOOM) [58] is an **OoO superscalar processor** that fits into the Rocket Chip ecosystem. At time of writing, the BOOM developers are in the process of validating RTL changes from their most recent updates to the RISC-V Privileged 1.10 and User 2.2 specifications, so we do not explore it in this work. However, integrating BOOM should require only a few lines of configuration change in FireSim and we plan to include it in a future release. Unlike software simulators, FireSim can integrate more complicated CPU models without sacrificing performance, as long as they fit on the FPGA and meet timing. From our early experiments, one BOOM core consumes roughly the same resources as a quad-core Rocket.

New storage technologies such as 3D XPoint are being evaluated for use in datacenter environments. In the near future, we are planning to replace our functional block device model with a timing-accurate model with pluggable timing mechanisms for various storage technologies (Disks, SSDs, 3D XPoint).

One way to further increase the number of simulated nodes is to use **FAME-5 multithreading** [24]. This maps multiple simulated cores onto each physical pipeline on the FPGA, at the cost of simulation performance and reduced physical memory per simulated core.

FireSim also supports attaching **accelerators** to Rocket Chip. One example of such an accelerator is the Hwacha data-parallel vector accelerator [25]. FireSim nodes can integrate Hwachas into a cluster, including simulating disaggregated pools of Hwachas. FireSim also contains a custom pass that can automatically transform Verilog generated from HLS tools into accelerators that plug into a simulation and can be simulated cycle-exact with the rest of the SoC.

IX. Conclusion

The open-source FireSim simulation platform represents a new approach to warehouse-scale architectural research, simultaneously supporting an unprecedented combination of *fidelity* (cycle-exact microarchitectural models derived from synthesizable RTL), *target scale* (4,096 processor cores connected by network switches), *flexibility* (modifiable to include arbitrary RTL and/or abstract models), *reproducibility*, target *software support*, and *performance* (less than 1,000× slowdown over real time), while using a *public FPGA-cloud platform* to remove upfront costs and provide large cluster simulations on-demand.

Acknowledgments

Research partially funded by DARPA Award Number HR0011-12-2-0016, RISE Lab sponsor Amazon Web Services, and ADEPT/ASPIRE Lab industrial sponsors and affiliates Intel, HP, Huawei, NVIDIA, and SK Hynix. Any opinions, findings, conclusions, or recommendations in this paper are solely those of the authors and do not necessarily reflect the position or the policy of the sponsors.

References

[1] B. C. Lee, "Datacenter Design and Management: A Computer Architect's Perspective," *Synthesis Lectures on Computer Architecture*, vol. 11, 2016.

[2] L. A. Barroso, J. Clidaras, and U. Hölzle, "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition," *Synthesis Lectures on Computer Architecture*, vol. 8, 2013.

[3] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal *et al.*, "Network Requirements for Resource Disaggregation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017.

[5] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengil *et al.*, "Accelerating Persistent Neural Networks at Datacenter Scale," in *Proceedings of the 29th IEEE HotChips Symposium on High-Performance Chips (HotChips 2017)*. IEEE, August 2017.

[6] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas *et al.*, "Single-chip microprocessor that communicates directly using light," *Nature*, vol. 528, December 2015.

[7] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated Memory for Expansion and Sharing in Blade Servers," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009.

[8] K. Asanović, "FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers," ser. FAST 2014.

[9] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos *et al.*, "Rack-scale disaggregated cloud data centers: The dReDBox project vision," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016.

[10] HP Labs, "The Machine," https://www.labs.hpe.com/the-machine, 2017.

[11] Huawei, "High throughput computing data center architecture - thinking of data center 3.0," www.huawei.com/ilink/en/download/HW_349607, 2014.

[12] Intel, "Intel rack scale design," https://www-ssl.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html, 2017.

[13] Facebook, "Disaggregated rack," http://www.opencompute.org/wp/wp-content/uploads/2013/01/OCP_Summit_IV_Disaggregation_Jason_Taylor.pdf, 2013.

[14] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, Mar. 2017.

[15] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "Scale-out NUMA," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014.

[16] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim, "dist-gem5: Distributed simulation of computer clusters," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2017.

[17] Z. Tan, Z. Qian, X. Chen, K. Asanović, and D. Patterson, "DIABLO: A Warehouse-Scale Computer Network Simulator Using FPGAs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015.

[18] Amazon Web Services, "Amazon EC2 F1 instances," https://aws.amazon.com/ec2/instance-types/f1/, 2017.

[19] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman *et al.*, "A Cloud-Scale Acceleration Architecture," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, October 2016.

[20] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme *et al.*, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*. IEEE Press, June 2014.

[21] Huawei, "Huawei releases the new-generation intelligent cloud hardware platform atlas," http://e.huawei.com/us/news/global/2017/201709061557, 2017.

[22] Alibaba, "Instance generations and type families," https://www.alibabacloud.com/help/doc-detail/25378.htm#f1, 2017.

[23] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski *et al.*, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[24] Z. Tan, A. Waterman, H. Cook, S. Bird, K. Asanović, and D. Patterson, "A Case for FAME: FPGA Architecture Model Execution," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010.

[25] Y. Lee, C. Schmidt, A. Ou, A. Waterman, and K. Asanović, "The Hwacha vector-fetch architecture manual, version 3.8.1," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-262, Dec 2015.

[26] K. Asanović, R. Avižienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio *et al.*, "The Rocket Chip Generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016.

[27] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis *et al.*, "Chisel: Constructing hardware in a Scala embedded language," in *DAC Design Automation Conference 2012*, June 2012.

[28] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, "It's time for low latency," in *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, ser. HotOS'13. Berkeley, CA, USA: USENIX Association, 2011.

[29] SiFive, Inc, "SiFive TileLink specification," https://static.dev.sifive.com/docs/tilelink/tilelink-spec-1.7-draft.pdf, 2017.

[30] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee *et al.*, "Strober: Fast and Accurate Sample-based Energy Simulation for Arbitrary RTL," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016.

[31] D. Kim, C. Celio, D. Biancolin, J. Bachrach, and K. Asanović, "Evaluation of RISC-V RTL with FPGA-Accelerated Simulation," in *First Workshop on Computer Architecture Research with RISC-V*, 2017.

[32] J. Leverich and C. Kozyrakis, "Reconciling high server utilization and sub-millisecond quality-of-service," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: ACM, 2014.

[33] S. Volos, D. Jevdjic, B. Falsafi, and B. Grot, "Fat caches for scale-out servers," *IEEE Micro*, vol. 37, Mar. 2017.

[34] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong *et al.*, "A Fully Associative, Tagless DRAM Cache," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015.

[35] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient Memory Disaggregation with Infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017.

[36] Gen-Z Consortium, "Gen-Z overview," Tech. Rep., 2016. [Online]. Available: http://genzconsortium.org/wp-content/uploads/2016/11/Gen-Z-Overview-V1.pdf

[37] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic *et al.*, "A Case for Specialized Processors for Scale-Out Workloads," *IEEE Micro's Top Picks*, 2014.

[38] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei *et al.*, "Profiling a Warehouse-scale Computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015.

[39] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons *et al.*, "Sources of error in full-system simulation," in *ISPASS*. IEEE Computer Society, 2014.

[40] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for RISC-V," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, Aug 2014.

[41] G. Gupta, T. Nowatzki, V. Gangadhar, and K. Sankaralingam, "Open-source Hardware: Opportunities and Challenges," *CoRR*, vol. abs/1606.01980, 2016.

[42] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov *et al.*, "OpenPiton: An open-source manycore research framework," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: ACM, 2016.

[43] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood, "The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers," in *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '93. New York, NY, USA: ACM, 1993. [Online]. Available: http://doi.acm.org/10.1145/166955.166979

[44] D. C. Burger and D. A. Wood, "Accuracy vs. performance in parallel simulation of interconnection networks," in *Proceedings of 9th International Parallel Processing Symposium*, Apr 1995.

[45] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio *et al.*, "Graphite: A distributed parallel simulator for multicores," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, Jan 2010.

[46] D. Meisner, J. Wu, and T. F. Wenisch, "BigHouse: A Simulation Infrastructure for Data Center Systems," in *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software*, ser. ISPASS '12. Washington, DC, USA: IEEE Computer Society, 2012.

[47] S. H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, "MDCSim: A multi-tier data center simulation, platform," in *2009 IEEE International Conference on Cluster Computing and Workshops*, Aug 2009.

[48] Jim Hogan, "Hogan compares palladium, veloce, eve zebu, aldec, bluespec, dini," http://www.deepchip.com/items/0522-04.html, 2013.

[49] H. Angepat, D. Chiou, E. S. Chung, and J. C. Hoe, "FPGA-Accelerated Simulation of Computer Systems," *Synthesis Lectures on Computer Architecture*, vol. 9, 2014.

[50] J. Wawrzynek, M. Oskin, C. Kozyrakis, D. Chiou, D. A. Patterson, S.-L. Lu *et al.*, "RAMP: A research accelerator for multiple processors," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-158, Nov 2006.

[51] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson *et al.*, "FPGA-Accelerated Simulation Technologies (FAST): Fast, full-system, cycle-accurate simulators," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, Dec 2007.

[52] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011.

[53] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, K. Mai, and B. Falsafi, "ProtoFlex: Towards scalable, full-system multiprocessor simulations using FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, Jun. 2009.

[54] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson *et al.*, "RAMP Gold: An FPGA-based architecture simulator for multiprocessors," in *Design Automation Conference*, June 2010.

[55] Z. Tan, "Using FPGAs to Simulate Novel Datacenter Network Architectures At Scale," Ph.D. dissertation, EECS Department, University of California, Berkeley, Jun 2013.

[56] Amazon Web Services, "Official repository of the AWS EC2 FPGA hardware and software development kit - github," https://github.com/aws/aws-fpga, 2017.

[57] Amazon Web Services, "AWS cloud credits for research," https://aws.amazon.com/research-credits/, 2017.

[58] C. Celio, D. A. Patterson, and K. Asanović, "The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, Jun 2015.