Improving Disaggregated System Evaluation with Modular End-to-End Simulation

Bin Gao* Hejing Li† Jialin Li* Antoine Kaufmann†

*National University of Singapore** *Max Planck Institute for Software Systems†*

Abstract

Research on disaggregated architectures and the systems built on them is challenging to evaluate. Such work by definition involves radical changes to performance critical system components (e.g., memory subsystem) with complex impact on overall system performance at scales ranging from racks to complete data centers. We examine the challenges faced in evaluation for work in this space, along with the inherent trade-offs chosen.

We then describe modular full system simulation with SimBricks as a means of evaluating disaggregated systems end-to-end. Simulation in SimBricks offers the required flexibility to modify and re-combine components, while maintaining control over performance across components. Looking forward, we discuss future steps to further improve and streamline disaggregated system simulation.

1 Introduction

Resource disaggregation holds the promise of improving resource utilization, reducing cost, independent scaling, and finer-grain fault isolation. This conceptually simple approach opens up a broad design space for systems disaggregating resources at different levels and with different approaches, often requiring new or modified hardware components. As a result, much of the work in this domain cannot be easily evaluated in a realistic physical testbed; prior work often compromise by using emulation, resulting in numerous caveats about the functionality and performance characteristics actually measured.

When a full system "end-to-end" evaluation in a physical testbed is out of reach, we argue that the next best thing is often a full system evaluation in simulation. The alternative of emulating functionality in software or with FPGAs or a non-full-system evaluation all risk missing essential hard-to-predict behaviors due to interactions between components or their relative performance characteristics. Unfortunately full system evaluation with existing simulators often requires an unreasonable amount of effort, or are outright incapable of fully simulating such systems.

In this paper we instead advocate for using modular full system simulation, combining multiple simulator instances for different hardware components. We use the SimBricks [28] simulation framework for network systems as a starting point. SimBricks defines fixed interfaces for different simulator types to enable modular composition, and includes mechanisms for scalable communication and synchronization of simulations with thousands of components. We discuss necessary changes for simulating a broad range of disaggregated system designs, including adding new component interfaces for memory disaggregation, and missing component simulators. Finally, we present preliminary results demonstrating the feasibility of this approach.

2 Evaluation Challenges

Multiple compounding factors make work on disaggregated systems challenging to evaluate.

Radical changes to hardware. First off, work in this space proposes a wide range of radical changes to today's standard server hardware architectures. Memory disaggregation, typically requires fundamental changes to the memory subsystem. Examples include only using local DRAM into cache [40], moving MMU out of processor [40, 15], or inspecting and interposing on cache coherence state [7]. Computational accelerators are similarly being decoupled from compute, by enabling accelerators to directly communicate over the network [11] and even with each other [5]. Even I/O devices such as network cards are re-imagined, e.g., by integrating them directly into DRAM [2] or the processor [17]. Hardware for such new proposals is by definition not available off the shelf.

Complex effects on full system. These radical changes, and disaggregation more generally, also increase both the number of cross-component interactions and their impact on overall system behavior, and introduces new opportunities for things to go wrong. For example, in a non-disaggregated system local execution is not affected by behavior on other nodes (barring controlled interaction through I/O devices). In a system with memory disaggregation, some memory accesses will travel across the shared network and may be processed in shared memory modules. And even minor disruptions in one component might cascade forward to other components, e.g., a memory access delayed by network congestion may in turn stall the processor and thereby delay OS timer interrupts preventing timely context switches. But such interactions only emerge if evaluating a complete end-to-end system.

Increased system scale. Similarly, the required scale for even minimal experimental setups that still result in meaningful results, is also significantly larger. Scale might

Resource	System	Cur. HW	Emu.	Sim.
Memory	[25, 30, 14, 3]	OS		
	[9, 39, 45]	App		
	[36]	FPGA		
	[40]		OS	
	[15]		FPGA	
	[22]			Partial
	[23]			Full
	[33]		VMM	Full
Storage	[40, 49]		OS	
Accelerators	[43, 12, 11, 24]	FPGA		

Tab. 1: Overview of work on resource disaggregation systems and the evaluation approaches employed.

range from a few machines [33] to racks [20, 22, 36, 41, 27] or whole clusters in data centers [48, 25]. Disaggregation by design is about flexibly sharing resources, and determining behavior and performance under realistic sharing conditions are usually the main evaluation goals.

In summary, we often require new hardware, but should evaluate end-to-end, and require testbeds of realistic scale.

3 Approaches to Evaluation

This struggle for meaningful evaluation is apparent in published work in this space. We now examine different approaches to evaluation the research literature has taken, and classify them into three categories for discussion. Tab. 1 provides an overview.

3.1 Designing for Today's Hardware

The first category is conceptually easiest to evaluate: work targeting already available hardware architectures. Here, evaluation in a complete physical testbed is typically feasible, although the required scale may still be a challenge when special hardware is required. A broad range of work leverages existing host and networking hardware and implements resource disaggregation in the operating system [25, 30, 14, 3] or in an application runtime [9, 39, 45]. Other work leverages existing programmable switches in the network to offload coordination and management tasks for resource disaggregation into the network [26, 46]. Finally, some work designs systems to *specifically leverage FPGAs* [36, 43, 12] where reconfigurable fabrics are a better fit than general purpose processors.

3.2 Emulation

The second category is work that proposes new hardware but uses *emulation* with currently available hardware as a means of evaluation. Fidelity of the emulation heavily depends on the specific use-case, but generally emulation is easier if the proposed hardware design is closer to currently available hardware.

Software Emulation. Emulating new functionality in software is the most common instance of this strategy. For example, LegoOS [40] emulates a hardware-managed cache for disaggregated memory accessed over the network, through the system's virtual memory support. MIND [26] leverages Intel Pin [31] for dynamic binary instrumentation to emulate accesses to remote memory in unmodified applications. In both cases this enables full-system evaluation with real applications. However, such emulation typically comes at a significant performance cost, in terms of operation latency, throughput, and resource utilization. For example, a software page-fault handler initiating a network transfer takes orders of magnitude longer compared to the same operation in dedicated hardware, and also consumes processor cycles otherwise available to the workload. Emulation thus enables functional evaluation, but performance measurements typically only offer a lower bound and slow emulation can easily more than absorb any performance improvements offered by the system under evaluation.

FPGAs. When proposing new hardware components, FPGAs can serve as a more realistic evaluation platform compared to software emulation. For example, Clio [15] proposes an SoC with a specialized ASIC hardware pipeline for memory nodes, and leverages a Xilinx SoC+FPGA board as an emulation for evaluation. Unlike with systems specifically designed for FPGAs (cf. §3.1), using FPGAs for evaluating ASICs is an emulation. Performance properties, such as operating frequencies power consumption, and resource utilization, vary substantially between ASIC and FPGA. As a result, most papers using FPGAs in this way include discussions of various performance limitations due to emulation limits, e.g., Clio [15] incurs high memory latencies and low DRAM capacity because of limitations with the particular FPGA board.

FPGA emulation is particularly useful for standalone components with standard interfaces, such as new PCIe devices. It, however, falls short when emulating changes to existing processors, e.g., modifying an Intel Xeon's MMU. Research platforms such as Enzian [8] aim to mitigate this through generous resource provisioning, e.g., by including high DRAM capacity, large FPGAs, and ample I/O connectivity. But even then, performance differences remain, and many designs will still not fit because of timing or resource limitations.

3.3 Simulation

Finally, when functionality beyond existing hardware is required and emulation is not feasible, the last resort is typically simulation. The key advantage of simulation compared to emulation is the *increased flexibility* and *complete control over timing*. This comes at a cost of often painfully slow execution, e.g., soNUMA [33] reports a

few thousand instructions per second, and the need to first implement and validate a simulator with the desired functionality. In spite of this, especially when proposing modified or new hardware architectures, simulation is often the only means for a rigorous evaluation.

Individual components. Limiting simulation to individual components is a common strategy to reduce complexity and simulation time. For example, dReDBox [1, 22] uses Intel Pin to record application memory traces in emulation, and then replay these through a simulator of just the memory interconnect and memory controller [4, 35]. Unfortunately, "piecemeal" simulations inherently do not capture full system behavior. For example, longer memory latency due to memory disaggregation would lead to changes in memory accesses, e.g., because thread switches occur at different points or delayed network transfers cascade to delayed future requests.

Full system. Full system simulation aims to simulate the complete system, including processor, memory subsystem, devices, networks, and the complete software stack of OS, libraries, and applications. soNUMA [33] uses the Flexus [47] simulator for simulating systems with NUMA interconnects extending across machines. Kommareddy et al. [23] use the structural simulator toolkit (SST) [38] from the HPC community to simulate 8 compute nodes connected to a central memory node. Finally, nanoPU [17] uses the FPGA-accelerated Firesim [21] simulator for simulating the complete RTL for an SoC architecture optimized for low-latency networking. For all three examples, simulation enables evaluation of the complete system, despite radical changes to hardware.

However, full system simulations typically require significant effort and computational resources. All three of these examples involve significant (one-off) modifications to different simulation frameworks. Further, these simulations require ample computational resources, be it hours of simulation on processors for Flexus and SST, or many FPGAs for Firesim. Finally, such one-off modifications also require long validation process against baselines and physical testbeds to ensure meaningful results.

We argue that simulation is and will remain an essential component for work on resource disaggregation systems and architectures. Combining general trends in the post-Moore era [16] with work in a nascent area, new hardware proposals and systems leveraging them will remain prevalent. The combination of evaluation challenges we outlined (cf. §2), imply that an end-to-end evaluation is necessary but challenging without a physical testbed and at the necessary scale. Prior work has demonstrated that emulation, in software or FPGAs, even when feasible often comes with limitations. Simulations, especially of the full system, can avoid these shortcomings by providing necessary flexibility and control over relative performance of system components. In the rest of this paper, we discuss how to address the challenges to full system simulation of disaggregated systems.

4 Modular Full System Simulation

Modular full system simulation with the SimBricks framework for simulating network systems [28], can also enable end-to-end evaluation of disaggregated systems.

4.1 SimBricks Summary

SimBricks assembles simulated full system testbeds from instances of simulators for different system components. Components in SimBricks correspond to natural boundaries in physical systems and they interconnect through the corresponding interfaces: PCIe between host simulators and devices such as NICs or NVMe SSDs, and Ethernet links between NICs, switches and network-attached accelerators. For each type of interconnect, SimBricks defines a message interface that conforms to the standard. This enables SimBricks to provide strong modularity: Integrating a component simulator only requires developing an adapter that implements the message interface, while the rest of the simulator is kept unmodified.

Component simulator instances in SimBricks run as independent processes that communicate through message passing over optimized shared memory queues. This loose coupling eases integration effort, as components are treated as black-boxes that communicate through asynchronous messages. It also allows component simulators to run on different processor cores *in parallel*, enabling SimBricks to scale up to larger simulations, and to scale out using distributed simulations. To ensure accurate evaluation results, SimBricks implements a time synchronization protocol that guarantees simulation timing and scales to hundreds or even thousands of components.

SimBricks scales to simulate 1000 hosts in distributed simulation, and can run unmodified OS, applications, and even RTL hardware designs and drivers. The open-source framework [42] already integrates a range of popular simulators, including gem5 [6], qemu [37], ns-3 [34], omnet++ [44], and Intel's Tofino Simulator [19]. The modular approach reduces effort for adapting a simulator configuration for a new use-case, as known-working configurations for unmodified components can simply be copied in. The authors also speculate that modular simulation configuration may reduce validation effort.

As such SimBricks provides a starting point for full system simulation of disaggregated systems and architectures.

4.2 Alternative Simulators

Before diving into necessary changes to SimBricks, we discuss simulators previously used for full system simulation of disaggregated systems. The *structural*

simulation toolkit (SST) [38] also composes modular component simulators. Because of SST's super computing background, it misses important components for data center systems. More importantly, disaggregated system work leveraging SST [23] has reported issues scaling past 8 simulated nodes without drastically increasing simulation time. soNUMA [33] leveraged a modified version of the Flexus simulator [47], but a slow-down of six orders of magnitude. Finally, nanoPU [17] leveraged Firesim [21] for FPGA-accelerated simulation of modified RISC-V hosts. Firesim requires a full RTL implementation of the simulated system, (except for the network), drastically limiting what can be easily simulated.

4.3 Support for Disaggregated Systems

While SimBricks can already simulate disaggregated systems that only require regular Linux network connectivity, the majority of work in the space needs additional functionality not yet in SimBricks. Changes range from adding new component simulators to adding new interfaces and abstractions in SimBricks.

4.3.1 Component Simulators

RDMA NIC. Much work in resource disaggregation [40, 26, 39, 45] relies on RDMA for network communication.SimBricks does not include RDMA NIC simulators. While much of this work targets current hardware and can be evaluated in physical testbeds, follow-up work with hardware changes should still be able to compare to it in an apples-to-apples setting. Thus in ongoing work, we are developing a component simulator for the Intel e810, a data-center 100 Gbps Ethernet NIC with available register-level data sheet and open source drivers.

Scalable compute node simulator. SimBricks includes simulators like gem5 and qemu that can simulate multi-cores. However, these are sequential simulators (in configurations that model timing), resulting in simulation times increasing proportionally with the number of simulated processors. To address this, we plan to explore and integrate other simulators that leverage parallel execution, such as Intel/Windriver Simics [10].

System-specific components. In §5, we extend SimBricks with components for simulating specific systems.

4.3.2 Interfaces

We propose to extend SimBricks with new interfaces for disaggregated components for work on memory disaggregation. This could be implemented in individual simulators, e.g., by adding a network-attached memory controller to gem5 directly. However, adding dedicated interfaces greatly increases modularity and flexibility for re-combining different component simulators.

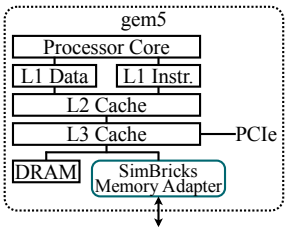

Fig. 1: SimBricks memory adapter in gem5 system.

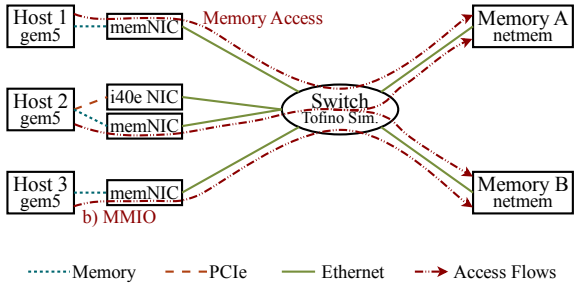

Fig. 2: SimBricks simulation of a disaggregated system with 3 compute, 2 memory, and 2 accelerator nodes.

Memory interface. We have implemented a memory interface in SimBricks along with an adapter in gem5 to connect components directly to the memory bus. This interface asynchronously carries read and write requests from gem5 to a memory component simulator and completions back. In gem5, the adapter implements the gem5 memory interface, and can thus be flexibly integrated into various gem5 configurations. Fig. 1 shows an example configuration.

Cache interface. As future work, we plan to introduce an interface for connecting components to caches including coherence. This requires extending the memory protocol with additional control messages for coherence (e.g., snoops). The key challenge is to find a balance of complexity and features. The protocol should cover a broad range of use-cases, allow for integration with different compute node simulators such as gem5 or Simics, and make it as easy to implement components using the interface.

CXL. CXL [13, 29, 32] has emerged in industry as the interconnect of choice for disaggregated hardware components. CXL extends PCIe with functionality for devices to efficiently expose device memory to the host (`CXL.mem`) and maintaining coherent on-device caches (`CXL.caches`). While commercial CXL hardware, including processors, mainboards, FPGAs, are becoming available [18], simulation still provides much greater flexibility for experimentation and extensions. As such, we plan to extend SimBricks with a CXL interface and corresponding adapters for compute node simulators.

5 Case-Study

5.1 OS Support for Disaggregation

LegoOS [40] is a new operating system that specifically targets disaggregated data centers. LegoOS proposes three concrete types of disaggregated resources and their respective monitors: a *pComponent* for processor hardware, a *mComponent* for memory nodes, and a *sComponent* for disaggregated storage nodes. Each component has a dedicated network-attached hardware controller. The pComponent contains CPU cores, caches, and limited amount of physical memory to both store kernel memory and serve as an extended cache (ExCache). Virtual memory translation and memory management are done by the mComponents. The ExCache on pComponents is managed by software – when the ExCache experiences a cache miss, the pComponent controller (CPU processors) issues a memory access request to the responsible mComponent over the network. The mComponent controller serves memory requests, performs address translation, and allocates/deallocates memory regions.

Unfortunately, the disaggregated hardware components proposed by LegoOS are not yet available. LegoOS therefore resorts to emulation by running all components on regular servers: pComponents are emulated by restricting the size of physical memory; a few processor cores are used to emulate controllers on mComponents and sComponents.

SimBricks offers a solution to properly evaluate a proposal like LegoOS. As shown in Fig. 1 and Fig. 2, we can simulate a pComponent in LegoOS using a combination of a host simulator like gem5 and a hardware network controller connected to the memory bus. Unlike the software emulation in LegoOS, SimBricks can accurately simulate a hardware memory adaptor directly connected to the L3 cache on the memory bus. Emulation on existing hardware limits LegoOS to process ExCache misses in software. With SimBricks, a researcher can explore a wider range options, for example, to handle ExCache misses directly in the adapter hardware, which may improve cache miss latency. The benefit of SimBricks-style simulation is more evident on mComponents. Instead of processing memory requests on CPU cores, SimBricks can simulate a hardware controller that is tightly coupled with component memory. This simulation setup closely matches the original disaggregated hardware proposal, which gives more realistic performance results. It also enables vast design space exploration – hardware design of the controller, latency and bandwidth between the controller and local memory — and test their impact on LegoOS performance.

5.2 In-Network Support for Disaggregation

One shortcoming of many memory disaggregation solutions, including LegoOS, is the lack of support for elastic scaling of processes to multiple compute nodes. The core of this limitation is that sharing writeable memory across compute nodes necessitate cache coherence, which can incur significant overhead. To address this issue, MIND [26] proposes to move memory and cache coherence management to the network fabric itself. Specifically, in a rack-scale deployment, the control plane of the ToR programmable switch is responsible for memory and coherence directory allocation, while coherence directory lookup, address translation, and coherence invalidation are done in the switch data plane.

SimBricks is fully capable of accurately simulating the MIND setup. SimBricks has already integrated a cycle-accurate Intel Tofino programmable switch model. As shown in Fig. 2, a set of memory nodes and compute nodes (combination of a host simulator and a memory NIC simulator) can be connected through a single Tofino simulator. The original MIND P4 and controller implementation can directly run in the Tofino model without any modification. Moreover, similar to our LegoOS discussion, SimBricks enables more realistic hardware simulation of the disaggregated compute and memory nodes – MIND still employs monolithic Linux servers to emulate both types of hardware components.

5.3 Preliminary Evaluation

To demonstrate that SimBricks can evaluate such systems, we have implemented a preliminary prototype, using simplified replicas of the required components as shown in Fig. 2. First off, we have implemented a network memory interface (`memNIC`), that connects to gem5 through our new SimBricks memory interface, and forwards accesses to the configured memory range, as UDP network packets on its outgoing SimBricks Ethernet port. Next, using the Intel Tofino simulator (as well as a simpler behavioral switch), we implement memory address translation and steering of requests to the corresponding memory node. We then implement a simple network-attached hardware memory node simulator processes our UDP network memory requests and responds to them. On the host side, we implement a simple Linux driver to register this memory region through the memory hotplug API into a separate (emulated) NUMA node.

We then run the `sysbench` memory benchmark pinned to that NUMA node for one second, and measure the simulation time, including Linux bootup and shutdown. To evaluate sensitivity of the simulation time to different latencies for the memory bus (which directly affect synchronization overhead), we measure with latencies of 20, 100, and 500 ns. The complete simulation takes 74, 77, and 78 min respectively, indicating that synchronization is not the bottleneck (likely gem5's instruction execution is). Even with a larger setup of 20 simulated hosts and 5 memory nodes, and 20 ns memory bus latency,

synchronization only increases to 114 min. Based on experiences reported in the SimBricks paper, we expect that at least half of the increase is due to thermal throttling of the processor as more cores are used.

6 Looking Forward

We now discuss remaining challenges and promising approaches to address them.

6.1 Open Challenges

The scale and diverse hardware in disaggregated systems give rise to challenges for modular end-to-end simulations.

Long simulation times. Many simulators incur high simulation times — taking hours to simulate few seconds. For many systems this severely limits what applications can feasibly be simulated. Many disaggregated systems require long initialization and warm-up periods before reaching steady state, incurring prohibitive simulation times.

High resource requirements. To make matters worse, SimBricks simulations require at least one core per component simulator instance for the full duration of the simulation. Due to busy-polled shared-memory queues, multiplexing multiple simulator processes onto a core, incurs excessive context switching and SimBricks time synchronization delays.

Validation. Finally, robust simulation results require validation of the simulation configuration and implementation, typically by comparing results against ground truth results from physical testbeds. However, ground truth results for novel disaggregated system architectures are typically hard or impossible to obtain, or even approximate through comparison to other similar physical testbeds.

6.2 Promising Directions

We now outline ideas toward addressing these challenges.

Decomposing component simulators. One way to accelerate slow and sequential component simulators is to parallelize. Preliminary results [28] suggest decomposing along natural boundaries and connecting and synchronizing the components through SimBricks mechanisms can effectively parallelize simulators. We are currently applying this to multi-core simulations in gem5 and network simulators, to address common bottlenecks in our simulations.

Hardware accelerated simulation. To further reduce simulation times, FireSim [21] leverages FPGAs to accelerate hardware RTL system simulations combined with software network simulations. First, we suggest integrating FireSim into SimBricks, by adapting it to SimBricks communication and synchronization mechanisms. Next, we expect that software simulations can also be significantly accelerated through hardware acceleration for parallelism and synchronization.

Mixed fidelity simulations. Many disaggregated systems simulation do not require the same level of detail throughout all components. For example, often we measure performance of applications with background tasks also using system components. Modular simulation can mix simulators with varying fidelity for different components, e.g. by reducing detail for the simulated background nodes while maintaining equivalent (visible) interactions with other system components. While simulation time is still bottlenecked by the slow detailed simulators for the components we measure, this reduces computational resources for the background nodes.

Composing validation. Finally, we expect modular simulations can also reduce validation effort. The open question is under which circumstances connecting individually validated simulator configurations together results in overall valid behavior. If this is possible, simulations could be assembled from pieces of pre-validated components. SimBricks accurately synchronizes and connects components, but errors in different components may accumulate. We propose empirical measurements with a broad range of disaggregated systems under different configurations. Additionally, we propose to design tools for detailed visibility into full system performance of systems simulated in SimBricks, taking inspiration from distributed tracing.

7 Conclusions

Modular simulation for disaggregated systems has the potential to significantly improve quality of our evaluations, while also providing additional flexibility and often reducing effort. SimBricks extended with a few additional interfaces and component quickly enables a host of work to be simulated without undue effort. Once researchers start adopting SimBricks for different disaggregated systems, the modular approach will enable follow-up work to easily reuse or adapt components and configurations.

SimBricks, including the extensions discussed in this paper, is open source and available at `https://simbricks.github.io`.

Acknowledgments

We would like to thank the anonymous reviewers for their comments and feedback. We also thank Jonas Kaufmann for his help with running experiments, and for developing the QEMU SimBricks memory protocol adapter. Jialin Li is supported by a MOE Tier 1 grant T1 251RES2104, an ODPRT grant A-0008089-00-00, and a Huawei industry grant A-8000079-00-00.

References

[1] Nikolaos Alachiotis, Andreas Andronikakis, Orion Papadakis, Dimitris Theodoropoulos, Dionisios Pnevmatikatos, Dimitris Syrivelis, Andrea Reale, Kostas Katrinis, George Zervas, Vaibhawa Mishra, et al. dReDBox: A disaggregated architectural perspective for data centers. In *Hardware Accelerators in Data Centers*, pages 35–56. Springer, 2019.

[2] Mohammad Alian and Nam Sung Kim. NetDIMM: Low-latency near-memory network interface architecture. In *52nd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO, 2019.

[3] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can far memory improve job throughput? In *15th ACM European Conference on Computer Systems*, EuroSys, 2020.

[4] Andreas E. Andronikakis. Memory system evaluation for disaggregated cloud data centers. Diploma thesis, Technical University of Crete, 2017.

[5] Nils Asmussen, Michael Roitzsch, and Hermann Härtig. M³x: Autonomous accelerators via Context-Enabled Fast-Path communication. In *2019 USENIX Annual Technical Conference*, ATC, 2019.

[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, August 2011.

[7] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *26nd International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2021.

[8] David Cock, Abishek Ramdas, Daniel Schwyn, Michael Giardino, Adam Turowski, Zhenhao He, Nora Hossle, Dario Korolija, Melissa Licciardello, Kristina Martsenko, Reto Achermann, Gustavo Alonso, and Timothy Roscoe. Enzian: An open, general, CPU/FPGA platform for systems software research. In *27th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2022.

[9] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2014.

[10] Jakob Engblom. Simics 5 is here — more paralllel than ever. https://blogs.windriver.com/rwoolley/2015/06/simics-5-is-here-more-parallel-than-ever/.

[11] Haggai Eran, Maxim Fudim, Gabi Malka, Gal Shalom, Noam Cohen, Amit Hermony, Dotan Levi, Liran Liss, and Mark Silberstein. FlexDriver: A network driver for your accelerator. In *27th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2022.

[12] Haggai Eran, Lior Zeno, Maroun Tork, Gabi Malka, and Mark Silberstein. NICA: An infrastructure for inline acceleration of network applications. In *2019 USENIX Annual Technical Conference*, ATC, 2019.

[13] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access,high-performance memory disaggregation with directcxl. In *2022 USENIX Annual Technical Conference*, ATC, 2022.

[14] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2017.

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

[16] John L. Hennessy and David A. Patterson. A new golden age for computer architecture. *ACM Transactions on Computer Systems*, 62(2):48–60, February 2019.

[17] Stephen Ibanez, Alex Mallery, Serhat Arslan, Theo Jepsen, Muhammad Shahbaz, Changhoon Kim, and Nick McKeown. The nanoPU: A nanosecond network stack for datacenters. In *15th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2021.

[18] Intel Corporation. Intel fpga compute express link (cxl). https://www.intel.sg/content/

www/xa/en/products/details/fpga/intellectual-property/interface-protocols/cxl-ip.html, 2022. Retrieved Oct 6, 2022.

[19] Intel Corporation. Intel Tofino series programmable switch ASIC. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, 2022. Retrieved Feb 2, 2022.

[20] Changyeon Jo, Hyunik Kim, Hexiang Geng, and Bernhard Egger. RackMem: A tailored caching layer for rack scale computing. In *2020 ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT, 2020.

[21] 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, and Krste Asanović. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *45th Annual International Symposium on Computer Architecture*, ISCA, 2018.

[22] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The DReDBox project vision. In *2016 Annual Design Automation Conference*, DATE, 2016.

[23] Vamsee Reddy Kommareddy, Simon David Hammond, Clayton Hughes, Ahmad Samih, and Amro Awad. Page migration support for disaggregated non-volatile memories. In *2019 International Symposium on Memory Systems*, MEMSYS, 2019.

[24] Youngeun Kwon and Minsoo Rhu. A disaggregated memory system for deep learning. *IEEE Micro*, 39(5):82–90, 2019.

[25] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. Software-defined far memory in warehouse-scale computers. In *24nd International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2019.

[26] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, and Abhishek Bhattacharjee. MIND: In-network memory management for disaggregated data centers. In *28th ACM Symposium on Operating Systems Principles*, SOSP, 2021.

[27] Sergey Legtchenko, Hugh Williams, Kaveh Razavi, Austin Donnelly, Richard Black, Andrew Douglas, Nathanael Cheriere, Daniel Fryer, Kai Mast, Angela Demke Brown, Ana Klimovic, Andy Slowey, and Antony Rowstron. Understanding rack-scale disaggregated storage. In *9th Workshop on Hot Topics in Storage and File Systems*, HotStorage, 2017.

[28] Hejing Li, Jialin Li, and Antoine Kaufmann. SimBricks: End-to-end network system evaluation with modular simulation. In *2022 ACM SIGCOMM Conference on Data Communication*, SIGCOMM, 2022.

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

[30] Shuang Liang, Ranjit Noronha, and Dhabaleswar K. Panda. Swapping to remote memory over InfiniBand: An approach using a high performance network block device. In *2005 IEEE International Conference on Cluster Computing*, 2005.

[31] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI, 2005.

[32] Pankaj Mehra and Tom Coughlin. Taming memory with disaggregation. *Computer*, 55(9):94–98, 2022.

[33] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. Scale-out NUMA. In *19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2014.

[34] nsnam. ns-3 — a discrete-event network simulator for internet systems. https://www.nsnam.org/, 2022. Retrieved Feb 2, 2022.

[35] Orion Papadakis. Memory system evaluation of disaggregated high performance parallel systems. Diploma thesis, Technical University of Crete, 2017.

[36] Christian Pinto, Dimitris Syrivelis, Michele Gazzetti, Panos Koutsovasilis, Andrea Reale, Kostas Katrinis, and H. Peter Hofstee. ThymesisFlow: A software-defined, HW/SW co-designed interconnect stack for rack-scale memory disaggregation. In *53nd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO, 2020.

[37] QEMU Authors. QEMU – the FAST! processor emulator. https://www.qemu.org/, 2022. Retrieved Feb 2, 2022.

[38] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, March 2011.

[39] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. AIFM: High-performance, application-integrated far memory. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2020.

[40] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2018.

[41] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2019.

[42] SimBricks. https://github.com/simbricks.

[43] Maroun Tork, Lina Maudlej, and Mark Silberstein. Lynx: A SmartNIC-driven accelerator-centric architecture for network servers. In *25nd International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2020.

[44] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools, 2008.

[45] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Semeru: A memory-disaggregated managed runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2020.

[46] Qing Wang, Youyou Lu, Erci Xu, Junru Li, Youmin Chen, and Jiwu Shu. Concordia: Distributed shared memory with in-network cache coherence. In *19th USENIX Conference on File and Storage Technologies*, FAST, 2021.

[47] Thomas F Wenisch, Roland E Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C Hoe. SimFlex: statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, August 2006.

[48] Qiao Zhang, Guo Yu, Chuanxiong Guo, Yingnong Dang, Nick Swanson, Xinsheng Yang, Randolph Yao, Murali Chintalapati, Arvind Krishnamurthy, and Thomas Anderson. Deepview: Virtual disk failure diagnosis and pattern detection for Azure. In *15th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2018.

[49] Qizhen Zhang, Xinyi Chen, Sidharth Sankhe, Zhilei Zheng, Ke Zhong, Sebastian Angel, Ang Chen, Vincent Liu, and Boon Thau Loo. Optimizing data-intensive systems in disaggregated data centers with teleport. In *2022 ACM SIGMOD International Conference on Management of Data*, SIGMOD, 2022.