

Ossim: OS-Driven Live Simulation for Cluster-Scale Full-Stack Evaluation

Yiliang Wan¹, Haifeng Sun^{1*}, Yihan Yang¹, Jonas Kaufmann²,
Antoine Kaufmann², Jialin Li¹

¹National University of Singapore

²Max Planck Institute for Software Systems

Abstract

Cluster-scale full-stack simulation is essential for evaluating distributed software stacks and emerging hardware components before deployment. Such simulation must achieve both full-stack fidelity for the unmodified production stack and the simulation performance required for iterative configuration exploration. However, no existing method achieves both. We present Ossim, an OS-level approach to cluster-scale full-stack simulation built on top of the Linux virtualization stack. Ossim comprises four subsystems: simulation-oriented scheduling, live memory hierarchy management, simulation-aware IPC, and distributed simulation orchestration. Together, they coordinate live and modeled components under shared simulated time while controlling interference among co-located live hosts. These mechanisms point toward *simulation-native OS support*, where simulation control and orchestration become core OS responsibilities.

CCS Concepts

• **Software and its engineering** → **Operating systems**;
• **Computing methodologies** → **Modeling and simulation**.

Keywords

full-stack simulation, live simulation, virtualization, distributed systems

ACM Reference Format:

Yiliang Wan, Haifeng Sun, Yihan Yang, Jonas Kaufmann, Antoine Kaufmann, Jialin Li. 2026. Ossim: OS-Driven Live Simulation for Cluster-Scale Full-Stack Evaluation. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '26)*, September 17–18, 2026, Bangkok, Thailand. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3838177.3841722>

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.

APSys '26, Bangkok, Thailand

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2912-6/26/09

<https://doi.org/10.1145/3838177.3841722>

1 Introduction

Cluster-scale full-stack simulation is essential for end-to-end evaluation of distributed systems and emerging hardware components before deployment. Cluster operators rely on it to choose deployment configurations for production workloads such as cloud platforms [36, 47] and big-data frameworks [4, 5]. System designers rely on it to evaluate emerging accelerators [14, 22, 44], memory systems [45], and interconnects [1, 2, 9, 34] before they are available at the target scale.

A simulator that supports these use cases must satisfy two properties at once. The first is *full-stack fidelity*: the simulator runs the production stack without source modification and preserves the cross-layer behavior it exhibits across the OS, runtime, application, distributed coordination layers, and underlying hardware. The second is *simulation performance*: the simulator completes in a timely fashion to support iterative configuration exploration.

Existing simulation methods satisfy some but not all of the properties required for cluster-scale full-stack simulation. Discrete-event simulators [16, 21, 41, 51] provide high fidelity within a single simulation component, but do not run the unmodified distributed software stack across cluster nodes. Modular composition frameworks [27, 28] recover fidelity by stitching simulators together, but their simulation performance is limited by their slowest components when running complex workloads. Live simulation shows that native execution under simulated time can provide high simulation performance while preserving fidelity, but existing systems [31, 39] remain workload-specific and do not run the full unmodified end-host stack.

We identify that the high performance of live simulation and the full-stack fidelity of modular composition together suggest a path toward cluster-scale full-stack simulation. Our key observation is that the Linux virtualization stack serves as a promising live-execution foundation for this approach. In particular, KVM [49], QEMU [37], and libvirt [46] provide functional emulation of the unmodified stack at near-native speed. However, the stack lacks four capabilities that large-scale live simulation requires: (1) control over when live components advance under simulated time, (2) performance isolation across live components on shared hardware, (3) communication between live and modeled components, and

(4) coordination among multiple physical hosts for large-scale simulation. All these capabilities sit below the user-space boundary and therefore call for OS-level mechanisms.

To this end, we present Ossim¹, a live simulator for cluster-scale full-stack evaluation that combines full-stack fidelity with high simulation performance. Ossim adds OS-level live-simulation support to the Linux virtualization stack through four subsystems. *Simulation-oriented scheduling* synchronizes live and modeled components under shared virtual time. *Live memory hierarchy management* limits interference among co-located live components for performance fidelity. *Simulation-aware IPC* controls cross-component events under virtual time. Finally, *distributed simulation orchestration* composes per-host mechanisms across machines for large-scale simulation. Together, the four subsystems define a live-simulation substrate spanning servers in a distributed cluster.

Our preliminary prototype runs the TPC-C benchmark in 90.4 seconds, whereas a comparable modular gem5-based setup did not finish within a week. Across representative workloads, Ossim reproduces physical behavior with promising accuracy. This demonstrates Ossim’s feasibility for cluster-scale full-stack simulation. Beyond this setting, Ossim points toward *simulation-native OS support*, where simulation control and orchestration become central OS abstractions rather than user-space services above a simulation-agnostic kernel.

2 Background and Motivation

2.1 Cluster-Scale Full-Stack Simulation

Cluster-scale full-stack simulation is essential before deployment because building and repeatedly reconfiguring large physical testbeds is costly and often impractical. By observing end-to-end behavior across software, hardware, and workload interactions, it provides a lightweight way to guide both near-term deployment tuning and longer-term hardware planning. This helps operators revisit server scale, network layout, and software configuration as workloads evolve. Designers can use it to evaluate future components, such as accelerators and memory tiers, within a full distributed system before such components are available at scale.

Compared with conventional domain-specific simulators (e.g., network simulators [41, 51]), a cluster-scale full-stack simulator must satisfy two properties.

R1. Full-stack fidelity. The simulator must faithfully reproduce the software–hardware interactions that determine cluster-scale behavior. We decompose this requirement into two aspects: *full-stack coverage* and *compatibility*.

- Full-stack coverage requires capturing interactions among the OS, runtime, application, distributed coordination logic, hardware resources, and deployment configuration. These

interactions determine cluster-level behavior, so modeling one layer in isolation or replaying recorded traces can miss the effects that shape end-to-end performance.

- Compatibility requires the simulator to run the production stack as deployed, without source modification. This property is important for fidelity since modifying or porting the stack can introduce artifacts that change the behavior being evaluated. It also allows production binaries and configurations to be evaluated directly as the stack evolves.

R2. Simulation performance. The simulator must run fast enough to support iterative exploration. Because the configuration space of cluster-scale systems is large and workload behavior changes over time, operators cannot rely on a single decisive run. Instead, they must evaluate a candidate configuration, observe its end-to-end behavior, refine the configuration, and repeat. Therefore, simulated-time progress must proceed at a rate compatible with end-to-end workload duration. Otherwise, if one minute of simulated workload takes days to complete, the simulator becomes impractical for interactive exploration.

2.2 Related Simulation Methods

Discrete-event simulation (DES). DES models the target system as a sequence of events processed in simulated time. Different tools in this family target different layers. Host-side simulators such as gem5 [16] and Simics [21] target cycle-level microarchitectures and provide full software compatibility. Network-side simulators such as ns-3 [41] and OMNeT++ [51] target protocols and topologies. These tools are mature and widely adopted, and they achieve high fidelity for studies within their target layer.

However, their layer specialization prevents them from satisfying the two requirements in §2.1. First, they lack full-stack fidelity. Host-side simulators miss end-to-end network behavior, while network-side simulators replace the OS, runtime, and application layers with traffic generators. Second, they sacrifice simulation performance since high-fidelity DES usually requires fine-grained event processing. For example, simulating one second of a 1,024-server fat-tree DCTCP workload in ns-3 can take over ten hours [6].

Modular composition. Modular frameworks [27, 28] address the full-stack fidelity limitation by composing independent host, switch, accelerator, and other component simulators into a unified multi-layer setup. By coordinating events and time across components, they recover full-stack fidelity that no single simulator provides. However, these frameworks still rely mainly on DES-based components, and thus can only run as fast as the slowest simulator used.

Live simulation. Live simulation addresses the simulation performance limitation by running parts of the system on

¹Ossim is open source at <https://github.com/ossim-project/ossim>.

Table 1: Comparison of simulation methods for cluster-scale full-stack simulation.

Method type	System	Sim. performance	End-host full-stack	Cluster-scale
DES	gem5/Simics ns-3/OMNeT++	✗ ✓	✓ ✗	✗ ✓
Modular comp.	SimBricks/ SplitSim	*	✓	✓
Live sim.	Phantora NEX Ossim	✓ ✓ ✓	† ✗ ✓	✓ ✗ ✓

* Depends on component simulators. † Runs unmodified ML apps without an OS.

real hardware under shared simulated time. Live components contribute real execution behavior at native speed, and modeled components are reserved for parts that are intentionally varied or unavailable. Phantora [39] achieves fast and accurate simulation for GPU training clusters. NEX [31] targets broader hardware-accelerated systems, but its demonstrated use cases remain limited to a single server and lack full software-stack compatibility.

Opportunity

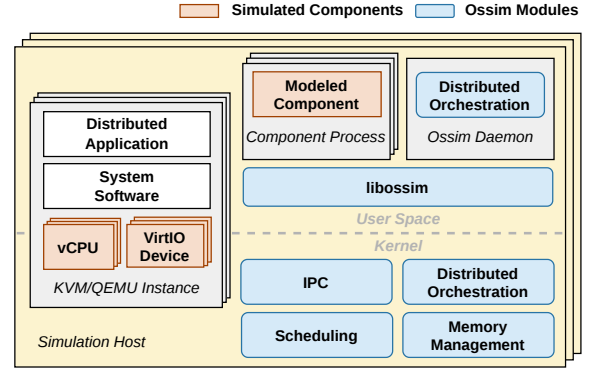
The high performance of live simulation and the full-stack fidelity of modular composition together pave the way for cluster-scale full-stack simulation.

3 Ossim Design

3.1 Ossim Overview

Key idea: Linux virtualization stack as a live execution foundation. The opportunity identified in §2.2 suggests that cluster-scale full-stack simulation can be achieved by combining the high performance of live simulation with the full-stack fidelity of modular composition. The missing piece is *a live execution foundation that can run the unmodified production stack correctly and at native speed*. Our key observation is that the Linux virtualization stack already provides the main building blocks for this foundation.

The Linux virtualization stack matches this need through a division of responsibilities across execution, compatibility, and management. First, KVM [49] enables *native guest execution* through hardware virtualization (e.g., Intel VT-x [19], AMD-V [3]), with each vCPU mapped to a kernel thread. Second, QEMU [37] provides *device compatibility* in user space, including device emulation, VM lifecycle management, and I/O processing through dedicated threads. Together, they allow an unmodified guest OS to run at near-native speed. Third, libvirt [46] provides APIs for VM lifecycle management and orchestration.

**Figure 1: The overall architecture of Ossim.**

Challenges. However, this stack alone does not provide the control needed for live simulation. It offers *functional emulation*, where real software executes correctly, but execution is still governed by the host wall clock and the host’s available performance. Turning this stack into a live execution foundation for cluster-scale simulation requires addressing four challenges:

- **C1. Virtual-time-based coordination:** advance live execution under shared simulated time.
- **C2. Live-component isolation:** bound interference among co-located live components on the shared host.
- **C3. Framework-level coordination:** allow live and modeled components to interact with each other.
- **C4. Scalable orchestration:** coordinate multiple physical hosts to support large-scale simulation.

OS-level simulation control. These challenges require OS-level support because their control points lie below the user-space boundary of existing modular composition frameworks. For **C1**, KVM vCPU execution and guest-visible clocks are kernel-managed. Thus, precise control over live execution cannot be provided by user-space orchestration and must reside inside the kernel. For **C2**, performance isolation depends on privileged host resources, including CPUs, caches, memory bandwidth, NUMA placement, and interrupts. As these resources are exposed through kernel interfaces, isolation requires kernel mechanisms. For **C3**, cross-component events become visible to live guests through virtual-device state updates or injected virtual interrupts. The timing of this visibility is determined by the kernel. Therefore, Ossim realizes these capabilities as OS-level mechanisms rather than as a purely user-space composition layer.

Architecture overview. Figure 1 shows the overall architecture of Ossim. Ossim follows one principle: “keep execution live and make the OS simulation-aware”. Live components, i.e., KVM/QEMU instances, execute directly on physical hardware. Modeled components, i.e., simulated devices,

advance according to their performance models. Ossim coordinates both types of components through four subsystems: *simulation-oriented scheduling* for virtual-time coordination, *live memory hierarchy management* for performance isolation, *simulation-aware IPC* (interprocess communication) for visibility-controlled communication, and *distributed orchestration* for multi-host scale-out. This allows Ossim to preserve full-stack fidelity while avoiding the fine-grained event processing overhead of pure DES-based simulation. Table 1 compares Ossim with existing methods.

3.2 Simulation-Oriented Scheduling

Ossim uses simulation-oriented scheduling to coordinate live and modeled components under bounded virtual-time skew (**C1 addressed**). The scheduler is built around the *vtask* abstraction, which represents any user-space thread whose execution must be coordinated by the simulation. Using this unified abstraction, the scheduler *accounts* for vtime and *dispatches* vtasks only when their progress remains within a configured skew bound.

Virtual-time accounting. For each vtask, Ossim maintains a *virtual time (vtime)* representing its cumulative simulated-time progress. For live vtasks, vtime is derived from hardware execution, while for modeled vtasks it is derived from simulated latency reported by the component model.

Clock-derived vtime. For live vtasks, Ossim derives vtime by adapting KVM’s *paravirtual clock (pvclock)* [50]. In standard KVM, preemption gaps are exposed to the guest through steal time. Ossim instead absorbs these gaps into the hardware TSC (Time Stamp Counter) [19] offset so that pvclock advances only during actual vCPU execution. The scheduler reads the same adapted pvclock as the guest, providing a single source of truth for both scheduler-maintained vtime and guest-visible time.

Model-driven vtime. For modeled vtasks, Ossim lets components report vtime advances computed by their performance models through either a synchronous *ioctl* or a shared per-*vtask run page* for the common asynchronous case. Modeled components report accumulated simulated latency at configurable granularity, and the scheduler advances vtime accordingly. Ossim also preempts modeled vtasks that run too long without reporting progress, preventing faulty components from stalling the simulation.

Dispatch. Ossim coordinates vtasks through configurable *synchronization domains* (hereafter, domains), each of which groups vtasks that should progress together within a bounded virtual-time skew. A vtask may participate in multiple domains.

Common-case dispatch rule. Each domain maintains a cached vtime, defined as the minimum vtime among its runnable members. During dispatch, a runnable vtask may execute

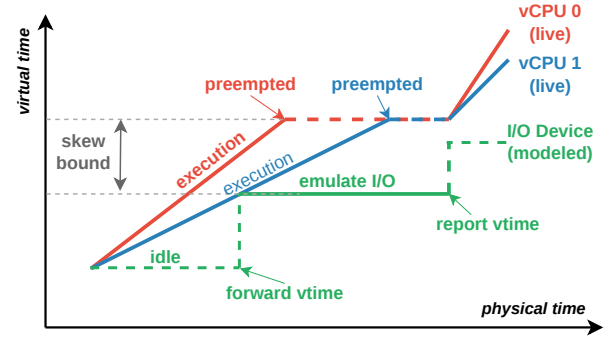


Figure 2: An example Ossim scheduling timeline.

only if $vtask.vtime \leq domain.vtime + skew_bound$. For vtasks belonging to multiple domains, the condition must hold for every domain. Vtasks that exceed the skew bound are rescheduled until slower components catch up. If no vtask on a CPU is eligible, the scheduler yields until other vtasks advance the relevant domain vtime. The skew bound controls the trade-off between synchronization precision and parallelism, and different domains may use different skew bounds.

Handling blocked vtasks. A blocked vtask is unable to make progress from the scheduler’s perspective, e.g., while waiting for I/O or halted, and is therefore excluded from the domain minimum. Including blocked vtasks would artificially pin $domain.vtime$ and may deadlock the simulation, e.g., when halted vCPUs lag behind a running bootstrap vCPU during VM boot. When a blocked vtask becomes runnable again, Ossim forwards its vtime to the current domain vtime before allowing it to rejoin bounded-skew scheduling. This preserves time causality: a vtask that blocks and later wakes observes that simulated time has advanced, just as a halted physical CPU observes elapsed time after resuming.

Figure 2 traces these mechanisms within a domain containing two live vCPUs and a modeled I/O device. The device starts idle and is excluded from the domain minimum. On wake-up, its vtime is forwarded to the domain minimum, which then preempts the two vCPUs at the skew bound.

3.3 Live Memory Hierarchy Management

Ossim prevents performance interference among live components through memory hierarchy management (**C2 addressed**). Live KVM hosts execute on real CPUs, so their performance is sensitive to shared cache capacity, memory bandwidth, and NUMA locality. By controlling these memory-hierarchy resources, Ossim preserves the fidelity of live execution.

Challenges. The subsystem must reduce two kinds of distortion. The first is *spatial interference*. Multiple live hosts running concurrently on the same physical machine compete for shared caches, memory bandwidth, and NUMA links, which slows their execution and therefore corrupts the virtual-time progress derived from live execution. The second is *temporal residue*. When live hosts time-share the same hardware, an incoming host may inherit cache and memory state left by a previously resident host.

Solution. Ossim organizes live-component isolation around the *cell* abstraction. A cell is a simulation-oriented generalization of Linux cpusets [48] that binds one live host to a controlled resource domain. This domain includes CPU and NUMA placement, cache capacity, memory-bandwidth limits, interrupt placement, and cell-switch state management. For spatial isolation, concurrently active live hosts are assigned to distinct cells. Cache capacity is controlled via Intel CAT [19] and AMD QoS [3], memory bandwidth is regulated using MBA [20], and NUMA locality is enforced through NUMA pinning. For temporal reconditioning, Ossim reconditions cache and memory state at cell-switch boundaries by invalidating the outgoing cell’s cache footprint and prefetching the incoming cell’s working set. Residual deviations are estimated through PMU sampling and reflected by adjusting the live CPU’s virtual-time advance. In this way, imperfect isolation is not hidden; it is explicitly incorporated into simulated time.

3.4 Simulation-Aware IPC

The simulation-aware IPC subsystem controls when cross-component messages become visible, so live and modeled components communicate in simulated-time order rather than host wall-clock order (**C3 addressed**).

Challenges. The first challenge is preserving communication semantics across heterogeneous components. A sender may produce an event at one vtime while the receiver is ahead, behind, or blocked. Immediate delivery can therefore expose state at the wrong simulated time and break causality. The second challenge is balancing generality with performance. The IPC abstraction should support simple links, lightweight latency models, and detailed modeled components, while keeping the common path fast.

Solution. Ossim organizes simulation-aware IPC around *messages*, *endpoints*, and *hubs*. Each message separates timing control from data movement. Its metadata records addressing and virtual-time information, while its payload is carried directly or through shared memory for high-bandwidth transfer. Endpoints act as proxies for existing component communication interfaces, including QEMU device interfaces for live guests. Hubs reside in the kernel, connecting endpoints and handling lightweight routing and latency control on the

common path. Their behavior is extensible through eBPF hook points, allowing simple connection logic to execute inside the hub without expensive context switches. More detailed connection behavior can instead be modeled as a separate component through the same endpoint–hub interface with higher overhead. For each vtask, Ossim maintains an incoming queue ordered by visibility time, so that messages become visible in virtual-time order. The scheduler can use the incoming queue head as a dispatch hint.

3.5 Distributed Simulation Orchestration

The distributed orchestration subsystem scales Ossim by composing per-host subsystems (virtual-time coordination, IPC, isolation) into a single cluster-scale simulation substrate while preserving a unified simulation view (**C4 addressed**).

Challenges. This subsystem must address two challenges. First, it must preserve local subsystem semantics across hosts, e.g., synchronization domains may contain remote vtasks, and IPC hubs may connect endpoints on different machines. Second, it must limit coordination overhead, since cross-host vtime synchronization and distributed IPC can place network communication on the simulation hot path.

Solution. Ossim combines a per-host user-space daemon with kernel-level support to extend local coordination mechanisms across physical hosts. To preserve cross-host semantics, remote vtasks are represented as local proxy vtasks that participate in bounded-skew enforcement, while logical hubs are implemented as distributed hub instances that exchange messages carrying addressing and visibility-time metadata. To reduce coordination overhead, the daemon handles only control-plane tasks, such as component placement and channel setup, while kernel mechanisms keep virtual-time updates and IPC delivery on the hot path. Ossim further reduces cross-host traffic by co-locating frequently interacting components when resources permit.

4 Preliminary Results

We implement a prototype of Ossim and conduct preliminary experiments to evaluate its accuracy and efficiency when simulating unmodified software stacks.

Implementation. Our prototype implements the core synchronization mechanism and adds an Ethernet switch hub. For synchronization, we integrate a customized scheduler with QEMU/KVM so that vCPU threads across instances advance under shared virtual time. For inter-component communication, we build the Ethernet switch hub on a Linux bridge with an interface to configure per-link performance characteristics, including bandwidth and latency. We used coding agents to assist with the implementation and debugging of Ossim.

Table 2: Accuracy and wall time on example workloads.

Workload	#Inst.	Metric	Acc.	Wall time (s)	
				Real	Ossim
CoreMark	1	Arith. Speed	97.8%	12.5	14.2 (1.1×)
TPC-C (MySQL)	2	Avg. Latency	82.2%	60.3	90.4 (1.5×)
		Throughput	92.3%		
YCSB (HBase)	3	Runtime	81.5%	28.7	81.5 (2.8×)
TPC-DS 99 (Spark)	3	Runtime	72.7%	9.3	36.8 (4.0×)
TPC-DS 99 (Hive)	3	Runtime	92.0%	31.8	98.8 (3.1×)

Experiments. We connect up to three QEMU/KVM VMs on one host in a star topology via the prototype hub. As ground truth, we deploy a matching physical testbed connected through a hardware Ethernet switch, with prototype hub parameters set to match. We evaluate Ossim against the physical testbed across three workload categories: a single-host CPU benchmark (CoreMark), database workloads (TPC-C on MySQL, YCSB on HBase), and big-data frameworks (TPC-DS 99 on Spark and on Hive).

Results. For each workload, Table 2 reports Ossim’s accuracy relative to the physical testbed together with wall-clock execution time. Ossim completes all workloads with a modest slowdown relative to the physical run, ranging from 1.1× (CoreMark) to 4.0× (TPC-DS on Spark). The slowdown increases with the number of synchronized KVM instances due to additional cross-instance coordination overhead. For comparison, we also attempted the TPC-C (MySQL) workload with modular simulation using gem5 [16] hosts, but the experiment did not finish within a week, whereas Ossim completed the same workload in 90.4 seconds. The prototype achieves up to 97.8% accuracy relative to the physical metric. We read this as an encouraging preliminary signal, given that the prototype implements only part of the full design. Several sources of inaccuracy remain uncontrolled, such as memory-hierarchy interference and IPC visibility timing. Overall, these preliminary results suggest that Ossim can simulate unmodified distributed software stacks with promising accuracy and substantially better performance than existing modular simulation approaches.

5 Related Work and Discussion

Simulation acceleration. Apart from live simulation, prior work accelerates simulation through system and runtime optimizations [6, 11, 15, 28, 30, 32], learned performance models [13, 26, 42, 54, 56], and specialized hardware such as GPUs and FPGAs [7, 18, 23, 25, 29, 38].

Simulation for distributed stacks. Other work targets specific cluster workloads, particularly GPU training [17, 39, 52], simulating both the distributed training strategies and the underlying hardware behavior. In contrast, Ossim

focuses on unmodified full-stack distributed software and its interaction with underlying hardware at cluster scale.

Resource isolation for guests. Partitioning hypervisors [8, 24, 33, 40] statically bundle fixed host resources for each guest and provide isolation through strict non-sharing of cache and memory hardware. In contrast, Ossim allocates host resources dynamically as simulated guests come and go, and its goal is simulation accuracy under controlled sharing.

Discussion. Our current prototype remains preliminary, with several subsystems still under active development. Beyond KVM-based VMs, integrating other virtualization technologies [10, 12, 35, 43, 53, 55, 57] could extend Ossim to additional live hardware such as GPUs.

6 Conclusion

Ossim takes an OS-level approach to cluster-scale full-stack simulation of unmodified distributed software stacks, integrating simulation-oriented mechanisms into the Linux virtualization stack. Our preliminary prototype reproduces physical behavior with promising accuracy and substantially better performance than existing approaches. We see Ossim as an initial step toward *simulation-native OS support*, a broader direction beyond cluster-scale full-stack simulation.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work was supported by the Singapore Ministry of Education, under the Academic Research Fund Tier 3 grant MOE-MOET32024-0004.

References

- [1] 2025. Ultra Ethernet Specification v1.0.1. <https://ultraethernet.org/uec-1-0-spec> Accessed: 2026-05-18.
- [2] 2026. Ultra Accelerator Link Specifications. <https://ualinkconsortium.org/specification/>. Accessed: 2026-05-18.
- [3] Advanced Micro Devices, Inc. 2024. *AMD64 Architecture Programmer’s Manual*. Accessed: 2026-05-19.
- [4] Apache Software Foundation. 2026. Apache Hadoop: Open-Source Framework for Distributed Storage and Processing. <https://hadoop.apache.org/> Accessed: 2026-05-18.
- [5] Apache Software Foundation. 2026. Apache Spark: Unified Analytics Engine for Large-Scale Data Processing. <https://spark.apache.org/> Accessed: 2026-05-18.
- [6] Songyuan Bai, Hao Zheng, Chen Tian, Xiaoliang Wang, Chang Liu, Xin Jin, Fu Xiao, Qiao Xiang, Wanchun Dou, and Guihai Chen. 2024. Unison: A Parallel-Efficient and User-Transparent Network Simulation Kernel. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys ’24)*.
- [7] Ben Romdhanne Bilel and Nikaein Navid. 2012. Cunesim: A gpu based simulation testbed for large scale mobile networks. In *2012 International Conference on Communications and Information Technology (ICCIT)*. IEEE, 374–378.
- [8] T. L. Borden, J. P. Hennessy, and J. W. Rymarczyk. 1989. Multiple Operating Systems on One Processor Complex. *IBM Systems Journal* 28, 1 (1989), 104–123. doi:10.1147/sj.281.0104

- [9] CXL Consortium. 2025. Compute Express Link (CXL) Specification Revision 4.0. <https://www.computeexpresslink.org/download-the-specification>. Accessed: 2026-05-19.
- [10] Niklas Eiling, Jonas Baude, Stefan Lankes, and Antonello Monti. 2022. Cricket: A virtualization layer for distributed execution of CUDA applications with checkpoint/restart support. *Concurrency and Computation: Practice and Experience* 34, 14 (2022), e6474. arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6474> doi:10.1002/cpe.6474
- [11] Fares Elsabbagh, Shabnam Sheikhha, Victor A Ying, Quan M Nguyen, Joel S Emer, and Daniel Sanchez. 2023. Accelerating rtl simulation with hardware-software co-design. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 153–166.
- [12] Ruwen Fan, Tingxu Ren, Minhui Xie, Shiwei Gao, Jiwu Shu, and Youyou Lu. 2025. GPREEPT: GPU Preemptive Scheduling Made General and Efficient. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. USENIX Association, Boston, MA, 263–272.
- [13] Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangli Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2023. RouteNet-Fermi: Network modeling with graph neural networks. *IEEE/ACM transactions on networking* 31, 6 (2023), 3080–3095.
- [14] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking: {SmartNICs} in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 51–66.
- [15] Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, and Lu Lu. 2023. Dons: Fast and affordable discrete event network simulation with automatic parallelization. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 167–181.
- [16] gem5 Project. 2026. The gem5 Simulator Project. <https://www.gem5.org/>. Accessed: 2026-05-19.
- [17] Fei Gui, Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Ran Zhang, Hongbing Yang, and Dian Xiong. 2025. Accelerating Design Space Exploration for {LLM} Training Systems with Multi-experiment Parallel Simulation. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 473–488.
- [18] Zizheng Guo, Yanqing Zhang, Runsheng Wang, Yibo Lin, and Haoxing Ren. 2025. GEM: GPU-accelerated emulator-inspired RTL simulation. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.
- [19] Intel Corporation. 2026. *Intel(R) 64 and IA-32 Architectures Software Developer's Manual*. Accessed: 2026-05-19.
- [20] Intel Corporation. 2026. Introduction to Memory Bandwidth Allocation. <https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-memory-bandwidth-allocation.html>. Accessed: 2026-05-20.
- [21] Intel Simics. 2026. Simics Full-System Simulator. <https://www.intel.com/content/www/us/en/developer/articles/tool/simics-simulator.html>. Accessed: 2026-05-19.
- [22] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.
- [23] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. 2018. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 29–42.
- [24] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. 2010. NoHype: Virtualized Cloud Infrastructure without the Virtualization. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. 350–361. doi:10.1145/1815961.1816010
- [25] Sajj Khashab, Hariharan Sezhiyan, Rani Abboud, Alex Normatov, Stefan Kaestle, Eliav Bar-Ilan, Mohammad Nassar, Omer Shabtai, Wei Bai, Matty Kadosh, et al. 2025. NSX: Large-Scale Network Simulation on an AI Server. In *Proceedings of the 2nd Workshop on Networks for AI Computing*. 19–25.
- [26] Chenning Li, Arash Nasr-Esfahany, Kevin Zhao, Kimia Noorbakhsh, Prateesh Goyal, Mohammad Alizadeh, and Thomas E Anderson. 2024. m3: Accurate flow-level performance estimation using machine learning. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 813–827.
- [27] Hejing Li, Jialin Li, and Antoine Kaufmann. 2022. SimBricks: end-to-end network system evaluation with modular simulation. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 380–396.
- [28] Hejing Li, Marvin Meiers, Jialin Li, and Antoine Kaufmann. 2025. SplitSim: Towards Practical Large-Scale Full-System Simulation for Systems Research. *Proceedings of the ACM on Networking* 3, CoNEXT4 (2025), 1–19. doi:10.1145/3768999
- [29] Qinyong Li, Zhiwei Zhao, Geyong Min, Zi Wang, and Luwei Fu. 2026. GeDES: GPU-Driven Discrete Event Network Simulator. In *Proceedings of the 21st European Conference on Computer Systems*. 468–483.
- [30] Tiantian Lin, Cheng Qiu, Xiaohang Wang, Ling Wang, Zhulin Zheng, Yingtao Jiang, Amit Kumar Singh, Jieming Yin, Sihai Qiu, Xiaodong Li, et al. 2025. LEGOSim: A Unified Parallel Simulation Framework for Multi-chiplet Heterogeneous Integration. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. 1347–1362.
- [31] Jiacheng Ma, Jonas Kaufmann, Emilien Guandalino, Rishabh Iyer, Bourgeat Thomas, and George Candea. 2025. Fast End-to-End Performance Simulation of Accelerated Hardware–Software Stacks. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles (SOSP '25)*.
- [32] Tie Ma, Long Luo, Hongfang Yu, Xi Chen, Jingzhao Xie, Chongxi Ma, Yunhan Xie, Gang Sun, Tianxi Wei, Li Chen, et al. 2024. Klonet: an {Easy-to-Use} and scalable platform for computer networks education. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2025–2046.
- [33] José Martins, Adriano Tavares, Marco Solieri, Marko Bertogna, and Sandro Pinto. 2020. Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems. In *Workshop on Next Generation Real-Time Embedded Systems (NG-RES), co-located with HiPEAC*. 3:1–3:14. doi:10.4230/OASiCS.NG-RES.2020.3
- [34] NVIDIA Corporation. 2016. *NVIDIA NVLink: High-Speed GPU Interconnect*. Technical Report. Accessed: 2026-05-18.
- [35] NVIDIA Corporation. 2026. NVIDIA Virtual GPU (vGPU) Software. <https://docs.nvidia.com/vgpu/>. Accessed: 2026-05-19.
- [36] OpenInfra Foundation. 2026. The Most Widely Deployed Open Source Cloud Software in the World. <https://www.openstack.org> Accessed: 2026-05-19.
- [37] QEMU Project. 2026. QEMU: The Fast Processor Emulator. <https://www.qemu.org/>. Accessed: 2026-05-18.
- [38] Yicheng Qian, Ran Shu, Rui Ma, Yang Wang, Derek Chiou, Nadeen Gebara, Luca Piccolboni, Miriam Leeser, and Yongqiang Xiong. 2025. Miniature: Fast AI Supercomputer Networks Simulation on FPGAs. In *Proceedings of the 9th Asia-Pacific Workshop on Networking*. 114–120.
- [39] Jianxing Qin, Jingrong Chen, Xinhao Kong, Yongji Wu, Tianjun Yuan, Liang Luo, Zhaodong Wang, Ying Zhang, Tingjun Chen, Alvin R Lebeck, et al. 2026. Phantora: Maximizing Code Reuse in Simulation-based Machine Learning System Performance Estimation. In *23rd*

- USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. 1809–1825.
- [40] Ralf Ramsauer, Jan Kiszka, Daniel Lohmann, and Wolfgang Mauerer. 2017. Look Mum, no VM Exits! (Almost). In *Proceedings of the 13th Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPRT)*. <https://arxiv.org/abs/1705.06932>.
 - [41] George F. Riley and Thomas R. Henderson. 2010. The ns-3 Network Simulator. In *Modeling and Tools for Network Simulation*. Springer, 15–34. doi:10.1007/978-3-642-12331-3_2
 - [42] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2020. RouteNet: Leveraging graph neural networks for network modeling and optimization in SDN. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2260–2270.
 - [43] Weihang Shen, Mingcong Han, Jialong Liu, Rong Chen, and Haibo Chen. 2025. {XSched}: Preemptive Scheduling for Diverse {XPU}s. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. 671–692.
 - [44] Jovan Stojkovic, Abraham Farrell, Zhangxiaowen Gong, Christopher J Hughes, and Josep Torrellas. 2026. AccelFlow: Orchestrating an On-Package Ensemble of Fine-Grained Accelerators for Microservices. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1–17.
 - [45] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. 2023. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 105–121.
 - [46] The libvirt Project. 2026. libvirt virtualization API. <https://libvirt.org/>. Accessed: 2026-05-19.
 - [47] The Linux Foundation. 2026. Kubernetes: Production-Grade Container Orchestration. <https://kubernetes.io/>. Accessed: 2026-05-19.
 - [48] The Linux Kernel Foundation. 2026. Cpusets. <https://docs.kernel.org/admin-guide/cgroup-v1/cpusets.html>. Accessed: 2026-05-20.
 - [49] The Linux Kernel Foundation. 2026. KVM: Kernel-based Virtual Machine. <https://www.linux-kvm.org/>. Accessed: 2026-05-18.
 - [50] The Linux Kernel Foundation. 2026. KVM PVclock. <https://www.linux-kvm.org/page/KVMClock>. Accessed: 2026-05-20.
 - [51] András Varga and Rudolf Hornig. 2008. An Overview of the OM-NeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools '08)*. ICST, Marseille, France, 1–10. doi:10.4108/ICST.SIMUTOOLS2008.3027
 - [52] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, et al. 2025. {SimAI}: unifying architecture design and performance tuning for {Large-Scale} large language model training with scalability and precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 541–558.
 - [53] Yechen Xu, Yifei Wang, Nathanael Ren, Yiran Chen, and Danyang Zhuo. 2026. Nixie: Efficient, Transparent Temporal Multiplexing for Consumer GPUs. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*. USENIX Association, Seattle, WA, 2085–2101.
 - [54] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. 2022. DeepQueueNet: Towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 441–457.
 - [55] Yanan Yang, Zhengxiong Jiang, Meiqi Zhu, Hongqiang Xu, Yujun Wang, Liang Li, Jiansong zhang, and Jie Wu. 2026. gShare: Efficient GPU Sharing with Aggressive Scheduling in Multi-tenant FaaS platform. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (USA) (ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 838–859. doi:10.1145/3779212.3790168
 - [56] Qizhen Zhang, Kelvin KW Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. 2021. MimicNet: Fast performance estimates for data center networks with machine learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 287–304.
 - [57] Shulai Zhang, Ao Xu, Quan Chen, Han Zhao, Weihao Cui, Zhen Wang, Yan Li, Limin Xiao, and Minyi Guo. 2025. Efficient Performance-Aware GPU Sharing with Compatibility and Isolation through Kernel Space Interception. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. USENIX Association, Boston, MA, 1003–1019.