Research Statement

Cross-Layer Co-Design for Next-Generation Datacenter Systems

Inho Choi

1 Introduction

Modern datacenters face a fundamental challenge: serving an ever-expanding spectrum of latency-critical and data-intensive workloads, ranging from microservices and real-time analytics to large-scale AI training and inference. These workloads demand both microsecond-scale tail latency and massive throughput at unprecedented scale. In response, datacenter network fabrics have scaled exponentially, with hyperscalers now entering the era of terabit networking. However, host-side compute has not kept pace. With the end of Moore's law and Dennard scaling, single-thread CPU performance has plateaued, while the volume and complexity of data each host must process continue to grow. As a result, the host, running applications and systems software, is emerging as a new performance bottleneck in modern datacenters.

Meanwhile, two new opportunities are reshaping the datacenter landscape. First, **specialized hardware accelerators** (e.g., SmartNICs, programmable switches, FPGAs, GPUs) are becoming widely available across the datacenter. Each is tailored to a specific class of computation, offering much higher performance and energy efficiency than general-purpose processors. Second, **machine learning** is increasingly being integrated into core system design. Recent advances have made ML models fast and accurate enough to make complex decisions, enabling their use for real-time decisions such as OS parameter tuning, resource allocation, and scheduling, where hand-crafted heuristics fall short. In parallel, advances in generative AI now allow synthesis of realistic system data at scale, addressing the long-standing scarcity of representative traces in tasks such as system analysis, anomaly detection, and model training.

Together, these trends unlock a new opportunity for **cross-layer co-design** of modern datacenter systems: addressing bottlenecks by deliberately distributing functionality across hardware, software, and machine learning, rather than optimizing within any single layer in isolation. My research applies this methodology along three connected directions. The first, **co-designing distributed systems with programmable network hardware**, has been the main focus of my doctoral research, where I have introduced programmable network hardware into distributed systems to deliver significantly higher performance and efficiency. Building on this foundation, my ongoing and future research extends the methodology in two further directions: **co-designing operating systems with machine learning**, where I am bringing ML into the OS as a first-class system component for real-time performance optimization, and **co-designing AI infrastructure with hardware accelerators**, where I am working to build systems that coordinate heterogeneous devices to accelerate AI workloads.

My research is grounded in both academic rigor and the realities of industry datacenters at scale. Through three research internships at Microsoft Research and AMD, and ongoing research collaboration with Microsoft Research, I have built systems that target the bottlenecks faced by large-scale industry environments, with a consistent emphasis on designs that are both technically novel and practically deployable. The resulting work has appeared at top-tier venues including NSDI and SIGCOMM.

2 Co-Designing Distributed Systems with Programmable Network Hardware

Traditionally, distributed systems have treated the network as a best-effort fabric: applications send and receive packets, while the network simply forwards and routes them. The emergence of programmable network hardware overturns this assumption: the network can now execute custom logic at line rate, exposing a rich design space in which distributed system semantics can be co-designed with the network. In this view, programmable network hardware is not merely a faster substrate for existing protocols, but a primitive that enables fundamentally new distributed protocols. My research has leveraged this co-design approach to address three critical datacenter challenges: distributed consensus (Hydra [1]), network load balancing (ConWeave [2]), and server load balancing (Capybara [3, 4]).

2.1 Hydra: Replacing Consensus with Scalable Network Ordering (NSDI '23) [1]

Maintaining strongly consistent state across distributed servers is foundational to mission-critical datacenter services. Traditional consensus algorithms (e.g., Paxos) achieve this through expensive coordination on every request, imposing substantial latency penalties and limiting system scalability. Network ordering systems (e.g., NOPaxos) eliminate this coordination overhead by ordering request messages at a sequencer within the network, but their fundamental reliance on a single sequencer (i.e., network serialization) leaves three critical problems: (1) sequencer scalability bottleneck, (2) network load imbalance, and (3) prolonged sequencer failure recovery time.

Hydra addresses these three problems by distributing ordering across *multiple* programmable switches that operate in parallel, with a novel cross-switch ordering scheme that enables receivers to merge per-switch sequences without any coordination among the switches themselves. Using this serialization-free network ordering protocol for distributed transactional systems, Hydra delivers 13× lower median latency, 47% higher throughput, and 5× faster sequencer failover than prior in-network ordering systems.

2.2 ConWeave: Network Load Balancing with In-network Reordering Support for RDMA (SIGCOMM '23) [2]

Datacenters typically have multiple paths between any two servers, and load balancing is used to spread traffic across these paths to maximize network capacity. RDMA, a high-performance networking technology widely adopted in modern datacenters, complicates this picture: RDMA hardware assumes packets always arrive in the order they were sent, and treats any out-of-order arrival as a sign of network congestion, immediately slowing down the sender. This assumption clashes with existing load balancing schemes: coarse-grained approaches (e.g., ECMP) cannot evenly distribute RDMA traffic, while finer-grained approaches (e.g., per-packet or per-flowlet distribution) inevitably cause out-of-order arrivals that severely degrade RDMA performance. The result is a fundamental trade-off between how finely traffic can be rerouted and how well packet order can be preserved.

ConWeave breaks this trade-off by performing fine-grained rerouting while restoring packet order *inside the network itself*, before packets reach the destination server. When congestion arises, the switch near the sender reroutes traffic onto a less congested path; the switch near the receiver then uses hardware features of modern programmable switches to temporarily hold and reorder the diverted packets, delivering them in the original order. Because all reordering happens within the network, no changes are required at end hosts or applications. Evaluations on a hardware testbed

show that ConWeave reduces average and 99-percentile flow completion times by up to 42.3% and 66.8%, respectively, compared to existing load balancing schemes.

2.3 Capybara: Dynamic Server Load Balancing with TCP Migration (APSys '23, SIGCOMM '26) [3, 4]

Datacenters rely on *load balancers* to distribute incoming client connections across servers. The most widely deployed type, the Layer-4 load balancer, statically assigns each connection to a server at the time it is first established and pins every subsequent packet of that connection to the same server. Such static assignment is required because TCP is a stateful protocol whose per-connection state must remain on the assigned server. As a result, the load balancer cannot rebalance workloads in response to the unpredictable traffic bursts and workload skews that frequently arise in real datacenters, leading to poor tail latency.

Capybara resolves this by enabling *live migration* of established TCP connections at microsecond timescales. When the load balancer detects an overloaded server, it can move some of its connections to underutilized servers in tens of microseconds. We achieve this through a tight co-design between a programmable network switch that redirects connection traffic according to migrations and a host TCP stack that can transfer per-connection state at microsecond speed. Under realistic workloads, Capybara achieves up to 149× lower 99-percentile latency and more than 2× higher throughput compared to existing L4 load balancers.

3 Ongoing and Future Research Directions

I am extending cross-layer co-design in two new directions: into the operating system itself, where the host stack remains a major source of performance variability, and into the AI infrastructure layer, where the coordination of heterogeneous accelerators will define the next decade of systems research.

3.1 Co-Designing Operating Systems with Machine Learning (Ongoing)

The host (and particularly the OS that mediates every I/O) has emerged as a major performance bottleneck in modern datacenters. Sustaining low latency and high throughput under increasingly demanding workloads requires the OS to adapt dynamically to workload characteristics, hardware behavior, and runtime system conditions. Yet the tuning surface is vast (dozens of parameters spanning networking, scheduling, and memory subsystems), parameter interactions are non-obvious, and optimal settings shift at microsecond timescales, far beyond what fixed rules or human-tuned heuristics can navigate. This motivates ML-native OS designs that integrate machine learning into the OS as a first-class system component, enabling continuous, workload-adaptive optimization.

Autokernel: an ML-native dataplane OS architecture. Autokernel [5] treats machine learning as a first-class component of the dataplane OS itself: it exposes a uniform tuning interface across networking, scheduling, and memory subsystems; embeds observation points throughout the system to generate the high-quality training data ML models require; and uses a hybrid inference strategy combining offline lookup tables for well-understood scenarios with lightweight ML models for unseen workloads, keeping the entire tuning loop within the tight latency budgets of dataplane I/O processing. In preliminary evaluation, Autokernel sustains around 100μs tail latency under dynamic workloads where a statically-tuned baseline incurs millisecond-scale queuing delays. Initial findings appeared at APSys '25, with full-system development ongoing in close collaboration with Microsoft Research.

Intra-host network bottleneck analysis. Effective ML-driven OS tuning requires precise understanding of where bottlenecks actually arise inside the host and which parameters meaningfully affect performance. In a complementary ongoing effort, I am conducting a systematic analysis of the intra-host network: the hardware data path that carries packets between the NIC and CPU/memory through the PCIe complex, IOMMU/IOTLB, and DDIO cache. Using detailed hardware telemetry, kernel-bypass profiling, and parameter sweeps, the analysis reveals that key dataplane parameters are strongly regime-dependent, with optimal settings differing by orders of magnitude across workload conditions (packet size, offered rate, core count). No static default is best across the operating range, directly motivating learned, runtime-adaptive policies that fuse live telemetry with workload characteristics.

Building on these foundations, I plan to advance ML-native systems along three forward-looking research thrusts:

- **Safe and verifiable learned OS policies.** Operators need confidence that learned decisions do not violate system-level safety, performance, or liveness guarantees. Building on Autokernel's runtime safety mechanisms, I plan to develop runtime safety frameworks for the OS: developers declaratively specify safety properties (input distribution, output bounds, decision quality, fairness) and corrective actions (fallback, retraining, resource reallocation) that compile into lightweight OS monitors. This moves ML-native systems from ad-hoc per-model safety toward a composable foundation that operators can trust and incrementally deploy in production.

- **Generative AI for ML-native systems.** Training-data scarcity is a persistent obstacle in systems ML, because production traces are difficult to obtain and rarely shared. Recent advances in generative AI enable a new approach: synthesizing realistic system traces such as packet flows, syscall sequences, scheduling decisions, and request patterns at scale, making model training, anomaly detection, and what-if analysis feasible even when real traces are unavailable. I plan to investigate how generative models can be co-trained with system telemetry to produce both faithful synthetic workloads and interpretable representations of system behavior.

- **Extending ML-native design to other OS domains.** ML-native design is not specific to datacenter dataplanes. I plan to apply the same principles to other systems with similarly large, dynamic tuning surfaces: GPU host OSes (where memory-allocator, scheduler, and interconnect tuning shape AI workload performance) and embedded/robotics OSes (where adaptive resource management is critical under tight real-time constraints). Each domain introduces distinct safety, latency, and observability requirements that will help shape ML-native OS principles more broadly.

3.2 Co-Designing AI Infrastructure with Hardware Accelerators (Future)

AI workloads such as LLM training and inference, and emerging agentic systems are fundamentally different from the workloads that shaped previous decades of datacenter design. They are often limited by data movement rather than raw compute; they span heterogeneous memory tiers (HBM, DRAM, CXL); they execute across thousands of GPUs coordinated by specialized fabrics (NVLink, InfiniBand); and they expose a level of asymmetry between accelerator compute and host I/O that existing system stacks were not designed to handle. My exposure to this design space began with RecoNIC, an industrial collaboration with AMD that gave me direct experience with SmartNIC platforms and with the host–device interaction patterns that dominate AI infrastructure performance.

RecoNIC is a hardware–software co-designed platform for RDMA-enabled compute offloading on FPGA-based SmartNICs. The motivation arose from a fundamental inefficiency in conventional

FPGA deployments: data arriving from the network is first DMA'd into host memory, then copied over PCIe to the FPGA for processing, and finally copied back to host memory before transmission. These extra hops cause significant performance overhead. RecoNIC address this by integrating an RDMA engine and programmable compute-logic modules directly into the SmartNIC's hardware shell, allowing network data to be placed directly into device memory and processed in place by user-defined accelerators.

Taking RecoNIC as a starting point, my research extends cross-layer co-design to the full landscape of AI infrastructure, where multiple specialized accelerators (GPUs, SmartNICs, programmable switches) and heterogeneous memory tiers (HBM, DRAM, CXL, SSD) interact. I aim to treat AI infrastructure as a single co-designed system rather than a collection of independently tuned components: cross-layer dataplanes that orchestrate data movement across accelerators and memory tiers, ML-driven policies that adapt placement and scheduling to workload dynamics, and programmable-network primitives that offload collective operations from hosts. This direction is a natural synthesis of my prior work, including programmable network coordination (Hydra, Capybara), runtime-adaptive OS (Autokernel), and SmartNIC compute offloading (RecoNIC), applied to the AI workloads that define the next generation datacenter design.

4 Conclusion

Datacenter systems are evolving rapidly: workloads driven by AI, microsecond-scale services, and agentic systems demand more than incremental tuning, while specialized hardware and machine learning expand the design space across the stack. My research pursues **cross-layer co-design**: deliberately redistributing system functionality across hardware, software, and machine learning. My research carries this methodology across the datacenter stack: from the distributed systems layer, into the host system itself, and toward the broader AI infrastructure layer. Together, these directions lie under the umbrella of redrawing the boundaries between hardware, software, and AI to meet the demands of next-generation datacenter systems.

References

[1] **Inho Choi**, Ellis Michael, Yunfan Li, Dan Ports, and Jialin Li. *Hydra: Serialization-Free Network Ordering for Strongly Consistent Distributed Applications.* In Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23), 2023.

[2] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, **Inho Choi**, Jialin Li, and Mun Choon Chan. *Network Load Balancing with In-network Reordering Support for RDMA.* In Proceedings of the 2023 ACM SIGCOMM Conference (SIGCOMM '23), 2023.

[3] **Inho Choi**, Nimish Wadekar, Raj Joshi, Joshua Fried, Dan R. K. Ports, Irene Zhang, and Jialin Li. *Capybara: µSecond-scale Live TCP Migration.* In Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '23), 2023.

[4] **Inho Choi**, Nimish Wadekar, Guangda Sun, Raj Joshi, Joshua Fried, Omar S. Navarro Leija, Dan Ports, Irene Zhang, and Jialin Li. *Capybara: Dynamic Load Balancing with Microsecond-Scale TCP Migration.* In Proceedings of the 2026 ACM SIGCOMM Conference (SIGCOMM '26), 2026.

[5] **Inho Choi**, Anand Bonde, Jing Liu, Joshua Fried, Irene Zhang, and Jialin Li. *ML-native Dataplane Operating Systems.* In Proceedings of the 16th ACM SIGOPS Asia-Pacific Workshop

on Systems (APSys '25), 2025.

[6] Guanwen Zhong, Aditya Kolekar, Burin Amornpaisannon, **Inho Choi**, Haris Javaid, and Mario Baldi. *A Primer on RecoNIC: RDMA-enabled Compute Offloading on SmartNIC.* arXiv:2312.06207, December 2023.