Research Statement

Anuj Kalia

With the seeming end of Moore's law, software does not automatically get faster every year: per-core CPU performance has stagnated, and core count increases slowly. My goal is to develop and disseminate techniques to build faster systems in this challenging new world. As noted by Leiserson et al. [12], two promising methods to improve system performance are optimizing software to best use the available silicon, and using specialized hardware tuned to specific applications.

A key challenge facing computing is to create a cohesive roadmap for future hardware specialization: how far can we go with existing hardware, and when must we turn to the more expensive option of deploying specialized hardware? My work makes the case that today's software has failed to capitalize on the performance and capabilities available within the existing hardware; we must address this failure first before looking to specialized hardware. A directly related question concerns the division of labor between existing and specialized hardware. For example, should "smart" NICs provide functionality such as kernel-bypass networking, Remote Direct Memory Access (RDMA), and congestion control? My results suggest that while supporting kernel-bypass in NICs is crucial for performance, the latter two can be handled well by existing CPUs.

Somewhat surprisingly, my work shows that in many applications for which specialized hardware—RDMA NICs, FPGAs, programmable switches, and GPUs—has been proposed, existing hardware can provide better or comparable performance. This result holds across a wide range of core systems problems: networked key-value storage and sequencing, distributed transaction processing, state machine replication, software packet processing, and regular expression matching. The differentiating aspect of my work has been its focus on building a deeper understanding of the capabilities of hardware—both existing and specialized—through extensive micro-benchmarking and analysis. Two outcomes of these studies led to the result above. First, I created numerous new optimizations for systems running on existing hardware by leveraging hardware capabilities in novel ways. Second, I found that for several applications, the proposed specialized hardware is a bad match from the perspective of the device's architecture, functionality, or placement. For example, a GPU accelerator placed over the slow PCIe bus is ill-suited for low-latency packet processing. Successful use of specialized hardware, such as TPUs for machine learning at Google or FPGAs for network virtualization at Microsoft, requires getting these matches right.

1 Thesis research: Efficient Networked Systems for Datacenters with RPCs

For my PhD thesis, I have studied the optimization-specialization tradeoff in the context of networked systems for modern datacenters. The networking and storage capabilities of datacenters have improved greatly in recent years. Modern networks provide up to 100 Gbps per-port bandwidth and a few microseconds round-trip latency (RTT). Storage in the form of non-volatile memory can persist data in less than a microsecond [1]. The raw hardware capabilities are sufficient for a server to process tens of millions of operations per second, and a client to achieve microsecond-scale latency.

Early efforts to achieve this performance found that software-only systems could not keep up with the network speed, and proposed adding specialized in-network hardware to offload work from the CPU, such as NICs that support Remote Direct Memory Access (RDMA), FPGAs, and programmable switches. This is a costly and complex solution, especially because it involves radical changes to the system's design and implementation. My work shows that by better understanding existing datacenter hardware, we can optimize software systems to use the full hardware capacity, and perform comparably to systems that use in-network devices.

In-network devices enable offloading work from CPUs to create "CPU-bypassing" distributed systems where some end-host CPUs are passive. This is the opposite of the traditional remote procedure call (RPC) architecture of distributed systems, where CPUs do all the heavy lifting. For example, an RDMA-capable NIC at the remote host can handle key-value GET operations by reading directly from its host's memory, bypassing the remote CPU. This is a seemingly attractive option because it avoids the overhead of software networking, which has traditionally been assumed to be fundamentally slow. For example, RPCs in the FaRM system perform up to 4x worse than RDMA [3]; the performance gap is attributed to RDMA's CPU-bypass capability, which FaRM uses extensively.

I found that there is a gap between the requirements of common distributed systems, and the limited compute and memory capabilities of network devices; working around this gap typically requires additional network round trips in CPU-bypassing designs. Consider the RDMA-based key-value store example above. RDMA NICs provide only remote reads, writes, and atomic operations, whereas a key-value store may require accessing a hash table. Hash tables typically consist of an index for fast lookup, and the actual data, requiring two or more RDMA reads to access data. This is unattractive because GETs can instead be handled in one round trip by involving the remote CPU with a simple RPC.

As another example, consider replicating a data item in the memory of a group of programmable switches for fault tolerance. Such systems use simple, serial chain replication because it is difficult to track the state for parallel primary-backup replication with the limited compute and memory resources in programmable switches [5]. An RPC-based design can instead propagate data to replicas in parallel, with fewer round trips on the critical path.

Next, I found that RPCs—and software networking in general—are not fundamentally slow. The key to fast RPCs is an improved understanding of datacenter network hardware capabilities such as NIC architecture, CPU-NIC interaction over the PCIe bus, and switch buffers. My study of these factors led to numerous optimizations, resulting in RPC implementations that perform close to wire speed. While RPC-based systems benefit from these optimizations, the laws of physics prevent optimizing away the slowdown from additional round trips in offloaded designs. Datacenter network links can be 300 meters long [4], so one round trip takes 2 microseconds at the speed of light. With fast RPC software that I built, CPU bypass saves only nanoseconds, which cannot make up for additional, microsecond-scale round-trips. Additional round trips implies more packets, and therefore reduced throughput.

My thesis contributes the principle that *the physical limitations of networks must inform the design of distributed systems*, i.e., designs with fewer round trips across high-latency interconnects are likely to perform better. I present practical systems, observations, and optimizations aimed at realizing the benefits of minimizing round trips. Below, I discuss my contributions in more detail.

1.1 Optimizations for CPU-based networking

My principle requires CPUs to handle all networking, which has traditionally been considered slow. I showed that this is because while modern NIC hardware offers the potential for exceptional latency and throughput, design choices including *which* NIC primitives to use and *how* to use them significantly affect observed performance. Navigating this large design space requires understanding complex low-level factors that were not well-understood in the systems and networking research community, such as the NIC's hardware architecture, and the CPU-NIC communication over the PCIe bus.

Throughout my PhD, I have studied these factors in detail using micro-benchmarking and analysis, and used the insights to create several new optimizations for RPCs in particular, and for networking in general. My work demonstrates the effectiveness of these guidelines by applying them to several systems. To make these insights easily accessible, I devised guidelines to help system designers navigate the design space [9]. First, I emphasize reducing CPU-NIC communication over PCIe: I showed how the performance counters on modern CPUs can model PCIe traffic without expensive analyzers, and created new techniques to reduce PCIe use based on these models. Second, I emphasize the role of the NIC's internal parallel architecture, and present techniques for exploiting this parallelism, and avoiding contention among the NIC's multiple processors. This work appeared in USENIX ATC 2016 and received a Best Student Paper award.

1.2 Case studies with real systems

I now describe two high-performance RPC-based systems that I have built, and how they compare with designs that use CPU offloads. A third system—state machine replication—is discussed in the next section. These systems store data in DRAM, and depend on small batteries to flush data to stable storage on failure [3]. I later describe my work on using recent non-volatile memory technologies in these systems.

Key-value stores. Distributed in-memory key-value stores are an important building block for large-scale web services. For example, Facebook's Memcache deployment consists of thousands of machines and acts as an object cache for trillions of data items [13]. HERD, published in SIGCOMM 2014 [6], is an RDMA-based key-value store that focuses its design on reducing network round trips. Unlike prior RDMA-based key-value stores, HERD does not bypass the server's CPU: A HERD client writes its request into the server's memory using an RDMA write; the server's CPU computes and sends the reply. In addition to lower, single-RTT latency, HERD's throughput is similar to native RDMA throughput for small key-value items, and is therefore around 2x higher than prior RDMA-based key-value stores.

Although HERD eschewed RDMA's main feature—remote CPU bypass—it still depended on an RDMA network for other features. Importantly, RDMA networks employ a lossless link layer that allowed HERD to ignore packet drops. Whether lossy networks allow high-performance networking remained an open problem until my recent work on eRPC, discussed in the next section.

Distributed transaction processing. FaSST, published in OSDI 2016 [8], extends the insight of minimizing RTTs to a more challenging application: distributed transactions. Transactions are useful for designing strongly-consistent distributed systems such as object stores and online transaction processing (OLTP) systems.

Similar to HERD, FaSST eschews remote bypass for fast RPCs. Unlike HERD, FaSST does not use any RDMA operations. This is because I found that current NIC architecture is mismatched to large-scale RDMA communication: the state for RDMA connections is held in small, on-NIC caches; cache misses are served over the slow PCIe bus, leading to low performance on large clusters that require thousands of connections. I designed a new scalable RPC system for FaSST that uses only datagram packets with no on-NIC connection state, achieving HERD-like RPC performance by leveraging optimizations for datagram packet I/O from our prior work [9]. FaSST outperforms prior remote-bypassing transaction systems by around 2x on standard OLTP benchmarks, while also being more scalable and simpler.

1.3 eRPC: Fast and general RPCs for datacenter networks

Although FaSST showed that RPCs are a good fit for building fast and scalable distributed systems, FaSST RPCs lacked several important features. The biggest deficiency was the inability to handle packet loss or congestion, preventing use in most existing datacenters, which use lossy Ethernet. eRPC, appearing in NSDI 2019 [10], is an RPC library that adds support for these and several other features for a small CPU cost.

eRPC was the first system to demonstrate near-hardware performance on lossy Ethernet datacenters. It showed that a networking library can achieve high performance without sacrificing functionality, or relying on in-network support. This breakthrough was made possible by new insights about datacenter hardware. For example, I showed an intriguing system-wide consequence of interacting NIC and switch hardware design decisions: the dynamic buffer pool in datacenter switches is sized in the tens of megabytes, which is much larger than the network's bandwidth-delay product (tens of kilobytes). Restricting each connection to one BDP of outstanding data therefore limits buffer buildup during incast, preventing most packet drops in real-world workloads.

eRPC allows building fast distributed systems that follow the *end-to-end principle*: all intelligence is contained at end host CPUs, and the network is a simple lossy pipe. Specialized offload devices and lossless link layers were deployed with the belief that placing their functionality in the network will yield a large performance gain, but I showed that eRPC can provide state-of-the-art performance without additional network support. For example, the latency of a production-grade state machine replication (SMR) system ported to use eRPC is better than or comparable to prior specialized replication systems that use programmable switches, FPGAs, or RDMA.

1.4 Adding persistence support with non-volatile memory

The systems described above assumed that batteries are available to flush volatile contents to stable storage on failure if persistence is needed. Recent non-volatile memory technologies like Intel's Optane DC Persistent Memory DIMMs [1] offer a cheaper, more reliable, and easier-to-manage alternative for high-performance persistence.

Intel has given me early access to these non-volatile DIMMs under a non-disclosure agreement. An advantage of my decision to involve the CPU in all operations in HERD, FaSST, and SMR-over-eRPC, is that these systems are easily ported to store data in persistent memory. For example, I showed that the latency of SMR-over-eRPC with persistent memory is close to DRAM-only replication, and an order of magnitude lower than replication to SSDs. I also found that adding persistence support to specialized systems that use in-network devices is not straightforward, since they lack ordering and persistence primitives. This work is currently under preparation [11].

2 Other research

The other major line of my research aims to provide stronger CPU baselines for accelerator research, which sometimes uses legacy, unoptimized CPU code for performance comparison. In G-Opt, published in NSDI 2015 [7], I examined the popular use of GPUs as software packet processing accelerators for applications such as routing table lookup and intrusion detection. I developed an automatic memory latency hiding technique that produces CPU baselines that are more resource efficient than existing GPU versions. In a masters project that I supervised, Xin Zhang examined the use of FPGAs for pattern matching in main-memory databases [14]. We showed that off-the-shelf CPU pattern matchers that are optimized for performance can compete with FPGAs for database pattern matching. This work is under submission [15].

3 Future research

It is an exciting time for impactful high-performance systems research: fast networks are now commonplace in datacenters, and the storage latency barrier has finally been broken by non-volatile memory. The software bloat and inefficiency in current implementations of infrastructure abstractions—networking, file systems, databases, etc—can no longer hide behind slow networks and storage. One of my long-term goals is to investigate how far existing hardware architectures can go towards achieving the speed of the underlying network or storage in these systems. After understanding the limits that might exist, we can turn towards specialized hardware. For example, eRPC shows that line rate on current 100 Gbps networks (with one core) is achievable without specialization. However, doing so for future terabit-per-second speeds without in-network support will require augmenting CPU architectures with specialized memory copy accelerators.

I am also broadly interested in systems and networks research. In the near future, I plan to investigate issues in datacenter networking. Three concrete paths are outlined next.

Protocol and API design for datacenter networks. One of eRPC's biggest contributions was showing that reliability and congestion control can be supported with low CPU overhead. It is therefore unclear why existing datacenter transport protocols such as TCP require much higher CPU cycles per packet: One CPU core can handle 15 million RPCs per second with eRPC, but only around 1.5 million packets per second with optimized userspace TCP implementations. Early pioneers of TCP have shown that TCP's original datapath is simple and can be implemented with low overhead [2]; I would like to study if this is still the case. If so, a likely cause for low performance could be the POSIX send/recv API that handles only one connection per call. In contrast, eRPC's API is explicitly designed to permit batched processing of multiple RPCs, on possibly different connections. Such batching allows several optimizations, such as faster network I/O and prefetching memory locations. These results, together with the increasing need for the underlying hardware to be parallel, suggest that a batching-centric redesign of operating system APIs may yield substantial benefits.

Towards a full-fledged networking library. Developers often desire features that are currently not provided by eRPC, such as encryption, and marshalling and unmarshalling of RPC arguments. These features are considered expensive, but I hypothesize that much of that perceived cost arises due to implementation inefficiencies and mismatches between hardware requirements and current API designs. For example, I believe that efficient secure communication can be provided end-to-end without custom hardware such as FPGAs and on-NIC accelerators. Modern CPUs provide acceleration for secure communication, such as instructions for AES encryption and SHA hashing, and vector instructions. Another example of the usefulness of batched APIs is that, due to intra-core hardware parallelism, these instructions are even more effective if multiple messages are available for simultaneous processing.

The database community, in the heyday of in-memory and NoSQL databases, tried to understand what makes a full-featured database slow. They analyzed which components are slow fundamentally, and which due to implementation and design. A similar analysis for datacenter networking is needed to establish a grounded basis for the development of future systems that successfully balance utility and efficiency.

Rearchitecting NICs and the application-NIC interface for efficient networking. While our research community is focused primarily on adding more features and intelligence to NICs, it is also worthwhile to revisit NIC architecture at a more fundamental level. The primary function of a NIC is to allow many CPU cores to send and receive packets with minimal overhead. Current NICs, however, run into scalability issues with only a few tens of CPU cores, and they use their internal SRAM and the PCIe bus inefficiently. In addition, vendors keep NIC architecture as a closely-guarded secret, which hinders performance understandability. A first step towards NICs that are efficient, scalable, and understandable, is specifying the interface and design of a minimal NIC, and developing a reference FPGA implementation. For example, it may be possible to simplify NIC design by replacing the send, receive, and completion descriptor queues used in current NICs with registers, which have constant NIC SRAM footprint.

More generally, I believe that by better leveraging existing hardware, redesigning APIs with a grounded understanding of what limits efficiency, and carefully choosing hardware specializations, we can lay the groundwork for creating future systems that match the performance of ever-faster networks and storage technologies.

References

[1] Available first on Google Cloud: Intel Optane DC Persistent Memory, 2018. https://cloud.google.com/blog/topics/partners/available-first-on-google-cloud-intel-optane-dc-persistent-memory.

[2] D. C. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Communications Magazine*, June 1989.

[3] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast remote memory. In *Proc. 11th USENIX NSDI*, Seattle, WA, Apr. 2014.

[4] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn. RDMA over commodity Ethernet at scale. In *Proc. ACM SIGCOMM*, Florianopolis, Brazil, Aug. 2016.

[5] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica. NetChain: Scale-free sub-RTT coordination. In *Proc. 15th USENIX NSDI*, Renton, WA, Apr. 2018.

[6] A. Kalia, M. Kaminsky, and D. G. Andersen. Using RDMA efficiently for key-value services. In *Proc. ACM SIGCOMM*, Chicago, IL, Aug. 2014.

[7] A. Kalia, D. Zhou, M. Kaminsky, and D. G. Andersen. Raising the bar for using GPUs in software packet processing. In *Proc. 12th USENIX NSDI*, Oakland, CA, May 2015.

[8] A. Kalia, M. Kaminsky, and D. G. Andersen. FaSST: Fast, scalable and simple distributed transactions with two-sided RDMA datagram RPCs. In *Proc. 12th USENIX OSDI*, Savannah, GA, Nov. 2016.

[9] A. Kalia, M. Kaminsky, and D. G. Andersen. Design guidelines for high-performance RDMA systems. In *Proc. USENIX Annual Technical Conference*, Denver, CO, June 2016.

[10] A. Kalia, M. Kaminsky, and D. G. Andersen. Datacenter RPCs can be general and fast. In *Proc. 16th USENIX NSDI*, Boston, MA, Feb. 2019.

[11] A. Kalia, M. Kaminsky, and D. G. Andersen. Microsecond-scale Persistent Distributed Systems with Non-volatile Memory. 2019. In preparation.

[12] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl. There's plenty of room at the top: What will drive growth in computer performance after Moore's law ends? 2018. To appear.

[13] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In *Proc. 10th USENIX NSDI*, Lombard, IL, Apr. 2013.

[14] D. Sidler, Z. István, M. Owaida, and G. Alonso. Accelerating pattern matching queries in hybrid CPU-FPGA architectures. In *Proc. ACM SIGMOD*, Chicago, USA, May 2017.

[15] X. Zhang, A. Kalia, M. Kaminsky, and D. G. Andersen. A Comparison of CPUs and FPGAs for Database Pattern Matching. Under submission to USENIX FAST, 2019.