Achieving a Billion Requests Per Second Throughput on a Single Key-Value Store Server Platform via Full Stack Architecting

Sheng Li[†], Hyeontaek Lim[‡], Victor W. Lee[†], Jung Ho Ahn[§], Anuj Kalia[‡],
Michael Kaminsky[†], David G. Andersen[‡], Seongil O[§], Sukhan Lee[§], Pradeep Dubey[†]
[†]Intel Labs, [‡]Carnegie Mellon University, [§]Seoul National University
[†]{sheng.r.li, victor.w.lee, michael.e.kaminsky, pradeep.dubey}@intel.com,
[‡]{hl, akalia, dga}@cs.cmu.edu, [§]{gajh, swdfish, infy1026}@snu.ac.kr

Abstract

Distributed in-memory key-value stores (KVSs), such as memcached, have become a critical data serving layer in modern Internet-oriented datacenter infrastructure. Their performance and efficiency directly affect the QoS of web services and the efficiency of datacenters. Traditionally, these systems have had significant overheads from inefficient network processing, OS kernel involvement, and concurrency control. Two recent research thrusts have focused upon improving key-value performance. Hardware-centric research has started to explore specialized platforms including FPGAs for KVSs; results demonstrated an order of magnitude increase in throughput and energy efficiency over stock memcached. Software-centric research revisited the KVS application to address fundamental software bottlenecks and to exploit the full potential of modern commodity hardware; these efforts too showed orders of magnitude improvement over stock memcached.

We aim at architecting high performance and efficient KVS platforms, and start with a rigorous architectural characterization across system stacks over a collection of representative KVS implementations. Our detailed full-system characterization not only identifies the critical hardware/software ingredients for high-performance KVS systems, but also suggests new optimizations to achieve record-setting throughput: 120 million requests per second (MRPS) (167 MRPS when with client-side batching) on a single commodity server. Our system delivers the best performance and energy efficiency (RPS/watt) demonstrated to date with existing KVSs—including the best-published FPGA-based and GPU-based claims. We propose a future manycore platform, and via detailed simulations demonstrate the capability of achieving a billion RPS with a single server constructed following our principles.

1. Introduction

Distributed in-memory key-value stores such as memcached [30] have become part of the critical infrastructure for large scale Internet-oriented datacenters. They are deployed at scale across server farms inside companies such as Facebook [32], Twitter [42], Amazon [2], and LinkedIn [28]. Unfortunately, traditional KVS implementations such as the widely used memcached do not achieve the performance that modern hardware is capable of: They use the operating system's network stack, heavyweight locks for concurrency control, inefficient data structures, and expensive memory management. These impose high overheads for network processing, concurrency control, and key-value processing. As a result, memcached shows poor performance and energy efficiency when running on commodity servers [27].

As a critical layer in the datacenter infrastructure, the performance of key-value stores affects the QoS of web services [32], whose efficiency in turn affects datacenter cost. As a result, architects and system designers have spent significant effort improving the performance and efficiency of KVSs. This has led to two different research efforts, one hardware-focused and one software-focused. The hardware-based efforts, especially FPGA-based designs [5, 6, 27, 40], improve energy efficiency by more than 10 times compared to legacy code on commodity servers. The software-based research [11, 12, 20, 26, 29, 31, 34] instead revisits the key-value store application to address fundamental bottlenecks and to leverage new features on commodity CPU and network interface cards (NICs), which have the potential to make KVSs more friendly to commodity hardware. The current best performer in this area is MICA [26], which achieves 77 million requests per second (MRPS) on recent commodity server platforms.

While it is intriguing to see that software optimizations can bring KVS performance to a new level, a number of questions remain unclear: 1) whether the software optimizations can exploit the true potential of modern platforms; 2) what the essential optimization ingredients are and how these ingredients improve performance in isolation and in collaboration; 3) what the implications are for future platform architectures. We believe the answers to these questions will help architects design the next generation of high performance and energy efficient KVS platforms.

Unlike prior research focusing on hardware or software in isolation, this work uses a full-stack methodology (software through hardware) to understand the essence of KVSs and to architect high performance and energy efficient KVS platforms. On the software side, we focus on the major KVS software components: the network stack, KV processing, memory management, and concurrency control. On the hard-

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

ware side, we architect a platform with balanced compute, memory, and network resources. We begin with a rigorous and detailed characterization across system stacks, from application to OS and to bare-metal hardware. We evaluate four KVS systems, ranging from the most recent (MICA) to the most widely used (memcached). Our holistic system characterization provides important *full-stack* insights on how these KVSs use modern platforms, from *compute* to *memory* and to *network* subsystems. Guided by these insights, we optimize MICA and achieve record-setting performance of 120 Million RPS (167 MRPS when with client-side batching) and energy efficiency of 302 kilo RPS/watt (401 kilo RPS/watt when with client-side batching) on our commodity CPU-based system. Our system delivers best performance and energy efficiency (RPS/watt) demonstrated to date, over existing KVSs including the best-published FPGA-based [40] and GPU-based [16, 43] claims. Finally, based on these full-stack insights, we propose a future manycore-based and whole-system-optimized platform architecture with the right *system balance* among compute, memory, and network, to illuminate the path to future high performance and energy efficient KVS platforms. Our detailed simulations demonstrate that the resulting design is capable of exceeding a billion requests per second on a quad-socket server platform.

2. Modern Platforms and the KVS Design Space

This section describes recent improvements in hardware and software, efficient KVS implementations, and the synergies between them.

2.1. Modern Platforms

The core count and last level cache (LLC) size of modern platforms continues to increase. For example, Intel Xeon processors today have as many as 18 powerful cores with 45MBs of LLC. These multi-/manycore CPUs provide high aggregate processing power.

Modern NICs, aside from rapid improvements in bandwidth and latency, offer several new features to better work with high-core-count systems: multiple queues, receiver-side scaling (RSS), and flow-steering to reduce the CPU overhead of NIC access [9, 37]. Multiple queues allow different CPU cores to access the NIC without contending with each other, and RSS and flow-steering enable the NIC to distribute a subset of incoming packets to different CPU cores. Processors supporting write-allocate-write-update-capable Direct Cache Access (wauDCA) [17],[1] implemented as Intel Data Direct I/O Technology (Intel DDIO) [8] in Intel processors, allow both traditional and RDMA-capable NICs to inject packets directly into the processor's LLC. The CPU can then access the packet data without going to main memory, with better control over cache contention should the I/O data and CPU working sets conflict.

Figure 1 briefly illustrates how these new technologies work together to make modern platforms friendly to network-intensive applications. Before network processing starts, a processor creates descriptor queues inside its LLC and exchanges queue information (mostly the head and tail pointers) with the NIC. When transmitting data, the processor prepares data packets in packet buffers, updates some transmit descriptors in a queue, and notifies the NIC through memory-mapped IO (MMIO) writes. The NIC will fetch the descriptors from the descriptor queue and packets from the packet buffers directly from LLC via wauDCA (e.g., Intel DDIO), and start transmission. While this process is the same as with single-queue NICs, multi-queue NICs enable efficient parallel transmission from multiple cores by eliminating queue-contention, and parallel reception by providing flow-steering, implemented as Intel Ethernet Flow Director (Intel Ethernet FD) [13] in Intel NICs. With flow-steering enabled NICs, each core is assigned a specific receive queue (RX Q), and the OS or an application requests the NIC to configure its on-chip hash table for flow-steering. When a packet arrives, the NIC first applies a hash function to a portion of the packet header, and uses the result to identify the associated RX Q (and thus the associated core) by looking up the on-chip hash table. After that, the NIC will inject the packet and then the corresponding RX Q descriptor directly into the processor LLC via wauDCA. The core can discover the new packets either by polling or by an interrupt from the NIC. The NIC continues processing new packets. Using wauDCA (e.g., Intel DDIO), network I/O does not always lead to LLC misses: an appropriately structured network application thus has the possibility to be as cache-friendly as non-networked programs do.

With fast network I/O (e.g., 100+ Gbps/node), the OS network stack becomes a major bottleneck, especially for small packets. Userspace network I/O, such as PacketShader I/O [14] and Intel Data Plane Development Kit (DPDK) [10], can utilize the full capacity of high speed networks. By eliminating the overheads of heavy-weight OS network stacks, these packet I/O engines can provide line-rate network I/O for very high speed links (up to a few hundred Gbps), *even for minimum-sized packets* [14, 39]. Furthermore, userspace networking can also be kernel-managed [4, 38] to maximize its benefits.

Although modern platforms provide features to enable fast in-memory KVSs, using them effectively is nontrivial. Unfortunately, most stock KVSs still use older, unoptimized software techniques. For example, memcached still uses the traditional POSIX interface, reading one packet per system call. This renders it incapable of saturating multi-gigabit links. Thus, we navigate through the KVS design space to shed light on how KVSs should exploit modern platforms.

[1] This paper always refers to DCA as the wauDCA design [17] (e.g., Intel Data Direct I/O Technology [8]) instead of the simplified Prefetch Hint [17] based implementation (e.g., Intel I/OAT [18]).

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

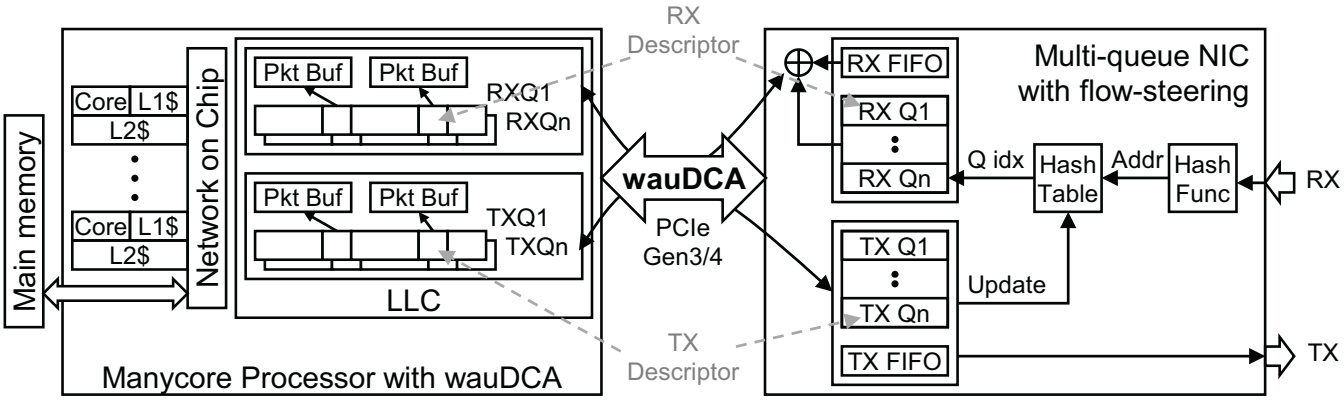

Figure 1: A modern system with write-allocate-write-update-capable Direct Cache Access (wauDCA), e.g., Intel DDIO [8], and a multi-queue NIC with flow-steering, e.g., Intel Ethernet Flow Director [13], to support high performance network intensive applications.

Network stack	Example systems
Kernel	memcached [30], MemC3 [12]
Userspace	Chronos [21], MICA [26]

Concurrency control	Example systems
Mutex	memcached, MemC3
Versioning	Masstree [29], MemC3, MICA
Partitioning	Chronos, MICA

Indexing	Replacement policy	Example systems
Chained hash table	Strict LRU	memcached
Cuckoo hash table	CLOCK	MemC3
Lossy hash index	FIFO/LRU/Approx.LRU	MICA

Memory management	Example systems
SLAB	memcached, MemC3
Log structure	RAMCloud [34]
Circular log	MICA

Table 1: Taxonomy of design space of key-value store (KVS) systems.

2.2. Design Space of KVSs

Despite their simple semantics and interface, KVSs have a huge design and implementation space. While the original memcached uses a conservative design that sacrifices performance and efficiency, newer memcached-like KVSs, such as MemC3 [12], Pilaf [31], MICA [26], FaRM-KV [11], and HERD [20], optimize different parts of the KVS system to improve performance. As a complex system demanding hardware and software co-design, it is hard to find a "silver bullet" for KVSs, as the best design always depends on several factors including the underlying hardware. For example, a datacenter with flow-steering-capable networking (e.g., Intel Ethernet FD) has a different subset of essential ingredients of an appropriate KVS design from a datacenter without it. Table 1 shows a taxonomy of the KVS design space in four dimensions: 1) the networking stack; 2) concurrency control; 3) key-value processing; and 4) memory management.

The networking stack refers to the software framework and protocol used to transmit key-value requests and responses between servers and clients over the network. memcached uses OS-provided POSIX socket I/O, but many newer, high-performance systems often use a userspace network stack to avoid kernel overheads and to access advanced NIC features. For example, Intel DPDK and Mellanox RDMA driversexpose network devices and features to user applications, bypassing the kernel.

Concurrency control is how the KVS exploits parallel data access while maintaining data consistency. memcached relies on a set of mutexes (fine-grained locking) for concurrency control, while many newer systems use optimistic locking mechanisms including versioned data structures. Versioning-based optimistic locking reduces lock contention by optimizing the common case of reads that incur no memory writes. It keeps metadata to indicate the consistency of the stored

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

key-value data (and associated index information); this metadata is updated only for write operations, and read operations simply retry the read if the metadata and read data versions differ. Some designs partition the data for each server core, eliminating the need for consistency control.

Key-value processing comprises key-value request processing and housekeeping in the local system. Hash tables are commonly used to index key-value items in memcached-like KVSs. In particular, memcached uses a chained hash table, with linked lists of key-value items to handle collisions. This design is less common in newer KVSs because simple chaining is inefficient in both speed and space due to the pointer chasing involved. Recent systems use more space- and memory access-friendly schemes such as lossy indexes (similar to a CPU cache's associative table) or recursive eviction schemes such as cuckoo hashing [35] and hopscotch hashing [15]. Replacement policies specify how to manage the limited memory in the server. For example, memcached maintains a full LRU list for each class of similar-sized items, which causes contention under concurrent access [12]; it is often replaced by CLOCK or other LRU-like policies for high performance.

Memory management refers to how the system allocates and deallocates memory for key-value items. Most systems use custom memory management to reduce the overhead of `malloc()` [30], to reduce TLB misses via huge pages [12, 30], and to help enforce the replacement policy [26, 30]. One common scheme is SLAB that defines a set of size classes and maintains a memory pool for each size class to reduce the memory fragmentation. There are also log-structured schemes [34], including a circular log that optimizes memory access for KV insertions and simplifies garbage collection and item eviction [26].

3. Experimental Methods

Our ultimate goal is to achieve a billion RPS on a single KVS server platform. However, software and hardware co-design/optimization for KVS is challenging. Not only does a KVS exercise all main system components (compute, memory, and network), the design space of both the system architecture and KVS algorithms and implementation are huge, as described in Section 2. We therefore use a multi-stage approach. We first optimize the software to exploit the full potential of modern architecture with efficient KVS designs and gain full-stack insights on the essential hardware and software ingredients. We then use these full-stack insights to architect future platforms that can deliver over a billion RPS per KVS server.

To pick the best KVS software design to start with, we have to navigate through the large design space of KVS and ideally try all the combinations of the design taxonomy as in Table 1, which is a nearly impossible task. Thus, we choose MICA [26], a state-of-the-art KVS software design as the starting point for optimization to fully exploit the potential of modern platforms. MICA is a KVS that uses a partitioned/sharded design

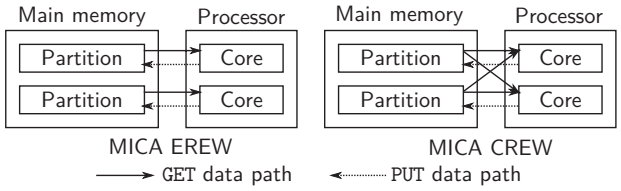

Figure 2: Partitioning/sharding in MICA.

Dataset	Count	Key size	Value size	Max pkt size
Tiny	192 Mi	8 B	8 B	88 B
Small	128 Mi	16 B	64 B	152 B
Large	8 Mi	128 B	1,024 B	1,224 B
X-large	8 Mi	250 B	1,152 B	1,480 B

Table 2: Workloads used for experiments. Max packet size is largest packet size including the overhead of protocol headers (excluding the 24-byte Ethernet PHY overhead). All data sets have a minimum 10GB memory requirement. Both Uniform key popularity and skewed key popularity follow a Zipf distribution (skewness of 0.99) are used. Workloads with 95% and 50% GET ratios are used to highlight how KVSs operate for read-intensive and write-intensive applications, respectively.

and high-speed key-value structures. It achieved 77 MRPS on a single KVS server, orders of magnitude faster than other KVSs. It partitions the key-value data and allocates a single core to each partition, which is accessed by cores depending on the data access mode as shown in Figure 2. In *exclusive read exclusive write* (EREW) mode, MICA allows only the "owner core" of a partition to read and write the key-value data in the partition, eliminating the need for any concurrency control. In *concurrent read exclusive write* (CREW) mode, MICA relaxes this access constraint by allowing cores to access any partition for `GET` requests (whereas keeping the same constraint for `PUT` requests), which requires MICA to use a versioning-based optimistic locking scheme. In MICA, remote key-value requests for a key must arrive at an appropriate core that is permitted to access the key's partition. This request direction is achieved by using flow-steering as described in Section 2.1 and making clients to specify the partition of the key explicitly in the packet header.

We use YCSB for generating key-value items for our workload [7] as shown in Table 2. The `STANDARD` workload used in several of our experiments later is defined as a uniform workload with tiny items and a 95% GET ratio.

Our experiment system contains two dual-socket systems with Intel® Xeon™E5-2697 v2 processors (12 core, 30MB LLC, 2.7GHz) supporting Intel DDIO (an implementation of wauDCA on Intel processors). Each system is equipped with 128GB of DDR3-1600 memory and four Intel®flow-steering-capable X520-QDA1 NICs, with total network bandwidth of 160Gbps per system.

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

This article has been accepted for publication in IEEE Micro but has not yet been fully edited.
Some content may change prior to final publication.

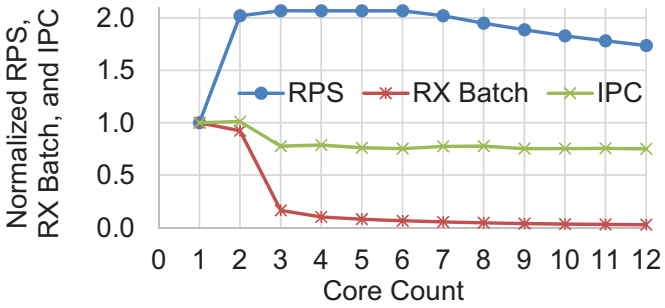

Figure 3: MICA's number-of-cores scalability with 1 10GbE port and one socket. We use `STANDARD` workload and EREW mode. RX batch size is the average number of packets MICA fetches per I/O operation from the NIC via DPDK because RX packets may queued up before CPUs processing them, which represents the amount of KV processing work per I/O operation. All numbers are normalized to their values at one core, where the actual RPS, RX batch size, and IPC are 5.78 MRPS, 32 packets per I/O operation, and 1.91 (IPC per core), respectively.

4. High Performance and Energy Efficient KVS on Commodity Systems

We first describe our optimizations guided by detailed full-system characterization, achieving 120 MRPS and beyond on our experiment platform. Then, we present insights gained from cross-layer performance analysis on system implications of KVS software design choices, as well as the essential ingredients for high performance KVS systems.

4.1. Architecture Balancing and System Optimization

Because KVSs exercise the entire software stack and all major hardware components, a balance between compute, memory, and network resources is critical. An unbalanced system will either limit the software performance or waste expensive hardware resources. An important optimization step is to find the compute resources required to saturate a given network bandwidth, for example, a 10GbE link. Figure 3 shows MICA's throughput when using one 10GbE link with an increasing number of CPU cores. While one core of the Intel Xeon processor is not enough to keep up with a 10GbE link, two cores provide close to optimal compute resources, serving 9.76 Gbps out of the 10 Gbps link. Using more cores can squeeze out the remaining 2.4% of the link bandwidth, at the expense of spending more time on network I/O compared to actual key-value (KV) processing. For example, using three cores instead of two reduces the average RX batch size by a factor of 6 (from 32 to 5.29), meaning that cores do less KV processing per I/O operation. Although the IPC does not drop significantly with adding more cores, the newly added cores simply busy-wait on network I/O without doing useful KV processing.

Holding the core to network port ratio as 2:1, we increase the cores and 10GbE ports in lockstep to test the full-system scalability. The maximum throughput achieved in this way is

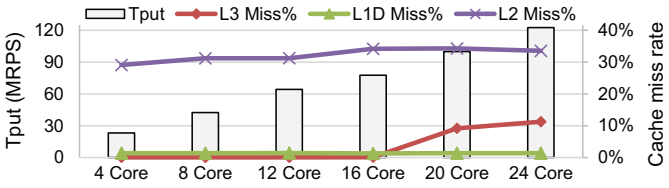

Figure 4: Throughput scalability and cache[2] miss rates of MICA with our optimizations. The port count used is half the core count. We use `STANDARD` workload and EREW mode.

80 MRPS with 16 cores and 8 10 GbE ports. Going beyond these values leads to a performance drop because of certain inefficiencies that we identified in the original MICA system. First, originally, each server core performed network I/O on all NIC ports in its NUMA domain. Thus, the total number of NIC queues in the system is $NumCores \times NumPorts$, leading to a rapid increase in the total network queues the processor must maintain. Having more total queues forces the NICs to inject more data into the LLC via Intel DDIO, but only up to 10% of the LLC capacity is allocated to Intel DDIO [8]. In addition, with more cores and higher throughput, the cores must fetch more data into the LLC for key-value processing. The combination of these two effects causes LLC thrashing and increases the L3 miss rate from less than 1% to more than 28%. To reduce the number of queues in the system, we changed the core-to-port mapping so that each core talks to only one port. We also optimized MICA to reduce MMIO induced overhead when collecting network statistics for flow control. These optimizations make MICA scale linearly with number-of-cores and number-of-ports in the system.

Figure 4 shows MICA's throughput and scalability with our optimizations. MICA achieves 120 MRPS when all 24 cores in both sockets are used. With increasing numbers of cores and ports, L1D and L2 cache misses remain stable, at $\sim$ 1.5% and $\sim$ 32%, respectively. The L1D miss rate stays low because of 1) MICA's intensive software prefetching, which ensures that data is ready when needed, and 2) MICA's careful buffer reuse such as zero-copy RX-TX packet processing. The high L2 cache miss rate is due to packet buffers that do not fit in L1D. The LLC cache miss rate is also low because network packets are placed in LLC directly by the NICs via Intel DDIO and because MICA uses intensive software prefetching. While the performance increases linearly, the LLC cache miss rate increases when there are more than 16 active cores (8 per socket), which indicates the importance of sufficient LLC capacity to accommodate both CPU data and network injected data for future manycore processors for high KVS performance even with the mapping optimization. Hereafter, we refer to MICA with our optimizations as MICA for simplicity.

[2]Unlike memcached with 20+% L1I$ miss rate due to the complex code path in the Linux kernel and networking stack [27], MICA's L1I$ miss rate is below 0.02% due to the use of userspace networking and kernel bypass.

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

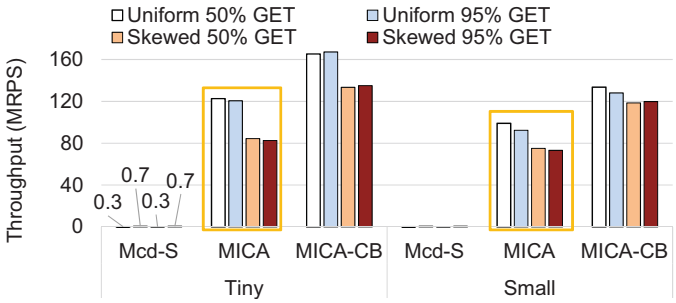

Figure 5: Throughput of optimized MICA and stock memcached (for comparison) with different datasets. MICA and MICA-CB are for optimize MICA without and with client-side batching, respectively. Key-value hit rates are within 98%∼99.6%, and the 95^{th} percentile of latency is less than 100µs. MICA runs EREW mode.

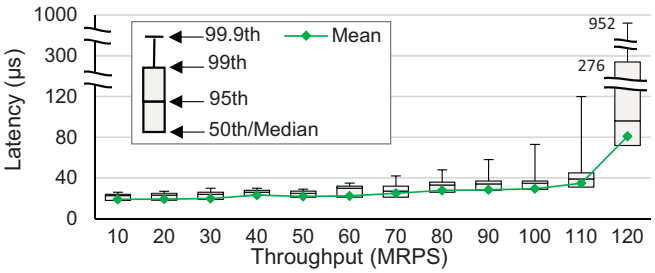

Figure 6: Round trip latency (RTT) (including mean, 50^{th}, 95^{th}, 99^{th}, and 99.9^{th} percentile) for different throughputs. `STANDARD` workload and MICA's EREW mode are used. Mean is always larger than median, because of the tailing effect. Experiments repeat multiple times to eliminate run-to-run variations.

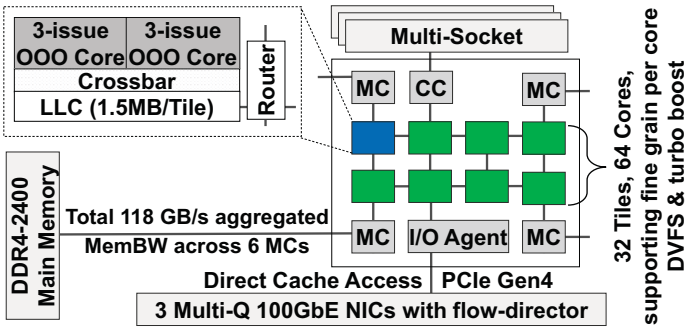

Figure 7: Proposed platform architecture for high performance KVS systems.

We also performed an architectural characterization of the system implications of simultaneous multithreading (SMT) [41] on our KVS performance, using Intel Hyper-threading Technology, an implementation of 2-way SMT on Intel processors. Our characterization shows that 2-way SMT causes a 24% throughput degradation with the full system setup (24 cores and 12 10GbE ports). This degradation is because the two hardware threads on the same physical core compete on cache hierarchy from L1 to LLC and cause cache thrashing, resulting in a 14%, 27%, and 3.6X increase on L1, L2, and LLC MPKI, respectively. While SMT can improve resource utilization for a wide variety of applications, MICA's relatively simple control structure means that it can incorporate application-specific prefetching and pipelining to achieve the same goal, making single-threaded cores sufficient.

4.2. Throughput, Latency, and Energy Efficiency

Figure 5 shows the measured full-system performance of optimized MICA with tiny and small datasets (Table 2) and different GET/PUT ratios. For tiny key-value pairs, MICA's throughput reaches 120.5∼116.3 MRPS with the uniform workload and 84.6∼82.5 MRPS for the skewed workload, with more than two orders of magnitude better performance than that of stock memcached on the same hardware. Client-side batching (MICA-CB in Figure 5) further improves performance of MICA by 35% to 68% for different workloads[3]. At 120.5 MRPS without client-side batching, MICA uses 110∼118 Gbps of network bandwidth under the uniform workload, almost saturating the network stack's sustainable 118 Gbps bandwidth on the server (when processing packet I/O only).

High throughput is only beneficial if latency SLAs (service level agreement) are satisfied, and Figure 6 reveals more latency-vs-throughput details. As throughput changes from 10M∼120M RPS, latency changes gracefully (e.g., mean: 19∼81µs; 95^{th}: 22∼96µs). Our optimized MICA achieves high throughput with robust SLA guarantees. Figure 5 shows that with the same 95^{th} percentile latency (less than 100µs), MICA (120 MRPS) achieves over *two orders of magnitude higher* performance than stock memcached (0.3 MRPS). Moreover, even at the highest throughput (120 MRPS), the 95^{th} percentile latency of MICA is only 96µs, ∼11X better than the 95^{th} percentile latency of 1135µs reported by Facebook [32].

The power consumption of our KVS platform is 399.2 W and 416.7 W for without and with client-side batching, respectively.At 120.5 MRPS, our KVS platform achieves 302 kilo RPS/watt (KRPS/W) energy efficiency. At 167.2 MRPS with client-side batching is enabled, our KVS platform achieves energy efficiency of 401 KRPS/W. These results show that traditional processors with full potential exploited can provide much higher performance *as well as much higher energy efficiency* than both FPGA- [5, 40] and GPU-based [16, 43] KVS platforms. In particular, our system achieves at least 6X better performance and 1.7X better energy efficiency than FPGA based systems and at least 1.39X better performance and energy efficiency than GPU based systems[4].

[3]This paper focus on use cases without client-side batching, and all results are without client-side batching unless otherwise specified. For more detailed analysis of client-side batching, please see our journal paper [24].

[4]Please see our journal paper [24] for detailed comparisons among different KVS systems.

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

CPU (w/ wauDCA similar to Intel DDIO [8])	
Technology (nm)	14
Core	Single-thread, 3-issue, OOO, 64 ROB
Clock rate (GHz)	2.5(N,0.7v)/1.5(LP, 0.6v)/ 3.0(TB, 0.85v)
L1 Cache	32 KB, 8-way, 64 B
L2 (LLC) Cache / tile	1.5 MB (768KB/core), 16-way, 64 B
# Cores(Tiles)/socket	60 (30), w/ 2 cores per tile
Integrated IO agent	PCIe 4.0 (tot. 32 lanes);
Memory Subsystem	
Memory Controllers	6, single-channel
Memory type	DDR4-2400
Network (w/ flow-steering similar to Intel Ethernet FD [13])	
Multi-queue NICs	Three 100GbE, PCIe4.0 x8 per NIC

Table 3: Parameters of the target platform. All numbers are for one socket, the target platform has two or four sockets. The different frequency-voltage pairs (obtained from McPAT [22]) are for normal (N), low power (LP), and turbo boost (TB). wauDCA can use up to 10% of LLC [8].

5. Achieving a Billion Request-per-Second on a Single KVS Server

Our optimized MICA design achieves record-setting performance and energy efficiency, offering valuable insights about how to design KVS software and its main architectural implications. This section focuses on our final grand challenge: designing future KVS platforms to deliver a billion RPS (BRPS) throughput using a single multi-socket server.

5.1. Architecting a Balanced Platform Holistically

Designing a BRPS-level (billion requests per second) KVS platform requires the right system balance among compute, memory, and network. As shown in Figure 7 and Table 3, the proposed platform consists of multiple manycore processors. Each processor is organized as multiple clusters of cores connected by an on-chip 2D mesh network. Each cluster has two out-of-order (OOO) cores, connected to a distributed shared LLC (L2 cache in our case) via a crossbar. A two-level hierarchical directory-based MOESI protocol is used for cache coherence for L1 and L2 as well as for wauDCA for the NICs. Multiple memory controllers provide sufficient memory bandwidth. The target processor was estimated to have a 440mm^2 die size and 125W TDP by using McPAT [22]. Each processor is paired with three multi-queue 100GbE NICs with flow-steering. The NICs communicate with the processor through PCIe 4.0 and inject packets directly to the LLC via wauDCA. The target server contains two or four such manycore processors.

Compute: Figure 3 shows that MICA's IPC is up to 1.9 on the CPU, which indicates that 4-issue OOO cores could be an overkill. Thus, we perform a sensitivity study for core weight through simulations. Figure 8 shows the platform's performance in normalized RPS as the reorder buffer (ROB) size (number of entries) and issue width are varied for datasets

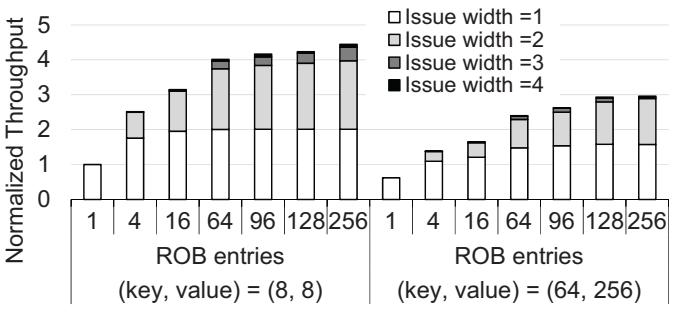

Figure 8: Relative performance for different item size when varying the ROB size and issue width of cores.

with different key and value sizes. Supporting multiple issues and out-of-order execution with a reasonably sized instruction window substantially improves the performance, but further increasing issue width or ROB size brings diminishing returns. In particular, as shown in Figure 8, increasing the ROB size from 1 (in-order issue) to 64 in the single-issue core doubles performance, but increasing it further to 256 only provides an additional 1% boost. With a ROB size of 64 entries, increasing issue width from 1 to 3 almost doubles system performance. Further increasing the issue width to 4, however, improves performance by only 5%. Considering the super-linear increase in complexity with larger window sizes and issue width, using a core more powerful than 3-issue with 64 ROB entries is not cost effective. Thus, we choose 3-issue OOO cores with 64 ROB entries in the target system.

Network and I/O subsystem: MICA (or any KVS) is a network application. Because our optimized MICA achieves near-perfect scaling (Section 4.1), we expect that the number of cores required per 10Gbps network capacity will remain unchanged, with appropriately sized (issue width, ROB size) cores and other balanced components. Thus, each 60 core processor can provide enough processing power for 300Gbps bandwidth. We assume that our platform will use emerging 100Gbps Ethernet NICs. Each 100GbE NIC requires at least 100Gbps of I/O bandwidth—an 8 lane (upcoming) PCIe 4.0 slot will be enough with its 128Gbps bandwidth. On-chip integrated NICs [25,33] will be an interesting design choice for improving system total cost of ownership (TCO) and energy efficiency, but we leave it for future exploration.

Cache hierarchy and memory subsystem: Our cache hierarchy contains two levels with a 32KB L1D cache. Our performance analysis and simulations reveal that a 32KB L1D cache is sufficient and a small private L2 cache in the presence of large L3 does not provide noticeable benefits due to high L2 miss rate. An LLC is critical not only to high performance KV processing on CPUs but also to high speed communication between CPUs and NICs. If the LLC cannot hold all RX/TX queues and associated packet buffers, LLC misses generated by NICs during directly injecting packets to the LLC via wauDCA will cause undesired main memory traffic leading to slow network and performance degradation.

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

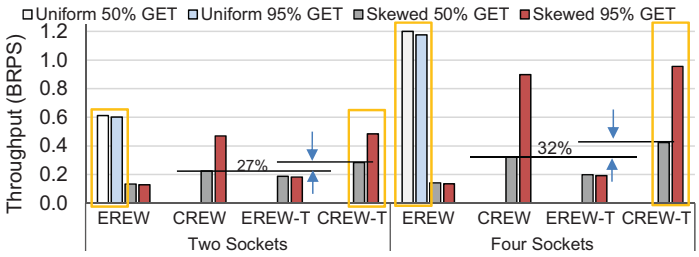

Figure 9: End-to-end performance of dual- and quad-socket servers. CREW and Turbo Boost (EREW-/CREW-T) are only applicable to, and thus are only shown for, skewed workloads. All 95^{th} percentile latencies are less than 100µs.

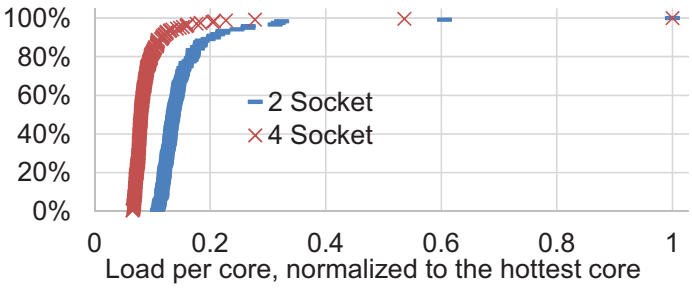

Figure 10: CDF of the partition load on different cores.

Moreover, contention between CPUs and NICs can cause LLC thrashing. For example, NICs can evict previously injected packets and even KV processing data structures (prefetched by CPUs) out of the LLC before they are consumed by CPUs. And even more cache conflicts will be generated when CPUs fetch/prefetch those data back from main memory for processing. Our detailed design space exploration sweeps shared LLC capacity (per core) from 256KB to 2MB and identifies the sweet spot to be 768KB. Therefore, we adopt the LLC design with 768KB per core (45MB per processor) in our manycore architecture

Like all KVS systems, MICA is memory intensive and sensitive to memory latency. Fortunately, its intensive SW prefetch mechanism is effective in trading memory bandwidth for latency [23, 24], which is favored by modern memory systems whose latency lags bandwidth significantly [36]. Thus, when designing the main memory subsystem, we provision sufficient memory bandwidth without over-architecting it for low memory latency. Should our optimized MICA reach 1 BRPS on the target 4-sockets platform, each socket will generate at least $\frac{1}{4} \cdot 4$ billion cache line requests per second from DRAM, for 64GB/s of DRAM bandwidth. We deploy six memory controllers with single-channel DDR4-2400 for a total of 118 GB/s aggregated memory bandwidth to ensure enough headroom for the bandwidth overhead because of MICA's software prefetching and the random traffic in key-value processing.

Discussions: Despite a carefully crafted system architecture, our platform remains general purpose in terms of its core architecture (3-issue with 64-entry ROB is midway in the design spectrum of modern OOO cores), its processor architecture (many cores with high speed I/O), and its system architecture (upcoming commodity memory and network subsystem). This generality should allow our proposed platform to perform well for general workloads.

5.2. Performance Evaluation

Figure 9 shows the simulated performance of the target dual- and quad-socket servers. Our simulation infrastructure is based on McSimA+ [1] with extensions to support the modern hardware features our proposal relies upon. Our optimized MICA achieves linear scaling on both dual- and quad-socket servers for uniform workloads, regardless of the GET ratio. As a result, the performance on the quad-socket platform successfully reaches $\sim$ 1.2 billion RPS (BRPS) in EREW mode with uniform workloads. Skewed workloads pose a harder problem on the target platform because of its large number of cores—increasing the number of cores leads to more partitions, which causes a larger load imbalance. In a Zipf-distributed population of size 192×2^{20} (192 million) with skewness 0.99 (as used by YCSB [7]), the most popular key is 9.3×10^6 times more frequently accessed than the average. For a small number of cores (thus partitions), the key-partitioning does not lead to a significant load imbalance [26]. For example, for 24 cores (and partitions), as in our experimental platform (Section 4), the most popular partition is only 97% more frequently requested than the average.

However, in our proposed architecture, the load on hottest partition is 10.6X (on the 240-core quad-socket server) and 5.8X (on the 120-core dual-socket server) of the average load per core, respectively. Although the increased data locality and decreased I/O processing overhead improves the performance of the hottest cores by $\sim$ 50% based on our simulations, it is not enough to bridge the gap between hot and cold partitions/cores. Thus, the hot cores become a serious bottleneck and cause a drastic performance degradation for skewed workloads: The performance on dual- and quad-socket machines is 0.13 BRPS (21% of the system peak performance) and 0.14 BRPS (11% of peak), respectively. Using the CREW mode can help GET-intensive skewed workloads, since in CREW mode all GET requests are sent to all cores to share the load (writes are still sent to only one core). However, for PUT-intensive skewed workloads (Skewed, 50% GET), there is still a large gap between the achieved performance and the peak performance (Figure 9).

Using workload analysis, we found that the load on the partitions (cores) is very skewed. On both systems, there are only two very hot cores (Figure 10). More than 90% of the cores are lightly loaded—less than 20% of the hottest core. This observation leads to an architectural optimization using dynamic frequency/voltage scaling (DVFS) and turbo boost (TB) technologies. We assume that our manycore processor is equipped with recent high efficiency per-domain/core on-chip voltage

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

regulators [19]. Based on the supply voltage and frequency pairs shown in Table 3, we reduce the frequency (and voltage) on the 20 most lightly loaded cores (their load is less than 12% of the load on the hottest core) from 2.5GHz to 1.5GHz and increase the frequency of the 6 most loaded cores to 3.5GHz. Results obtained from DVFS modeling in McPAT [22] show that this configuration actually reduces total processor power by 16%, which ensures enough thermal headroom for turbo boost of the 6 hot cores. Our results in Figure 9 show that with *CREW-T*, the combination of fine-grained DVFS/TB and MICA's CREW mode, the system throughput for the write-intensive skewed workload (Skewed, 50% GET) improves by 32% to 0.42 BRPS and by 27% to 0.28 BRPS on the quad- and dual-socket servers, respectively. Although datacenter KVS workloads are read-heavy with GET ratio higher than 95% on average [3], this architecture design is especially useful for keys that are both hot and write-heavy (e.g., a counter that is written on every page read or click).

Although distributing jobs across more nodes/servers (with fewer cores/sockets per server) works well under uniform workloads, as skew increases, shared-read (CREW, especially our newly proposed *CREW-T*) access becomes more important. Thus, a system built with individually faster partitions is more robust to workload patterns, and imposes less communication fan-out for clients to contact all of the KVS server nodes.

6. Conclusions

As an important building block for large-scale Internet services, key-value stores affect both the service quality and energy efficiency of datacenter-based services. Through a cross-stack whole system characterization and optimization, this paper achieves the record-setting 120 MRPS performance (167 MRPS when with client-side batching) and 302 KRPS/watt (401 KRPS/watt when with client-side batching) energy efficiency on commodity x86 servers. Our full stack study on KVSs also provide insights on the design of high performance KVS software and its main architectural implications. Base on the insights, we propose a future manycore-based and whole-system-optimized platform architecture to illuminate the path to future high performance and energy efficient KVS platforms. Through detailed simulations, we have demonstrated that the proposed system can achieve a billion RPS performance with QoS guarantees on a single four-socket key-value store server platform. These results highlight the impressive possibilities available through careful full-stack hardware/software co-design for increasingly demanding network-intensive and data-centric applications.

References

[1] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *ISPASS*, 2013.
[2] Amazon, "Amazon Elasticache," http://aws.amazon.com/elasticache/, 2012.
[3] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *SIGMETRICS*, 2012.
[4] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion, "IX: A protected dataplane operating system for high throughput and low latency," in *OSDI*, 2014.
[5] M. Blott, K. Karras, L. Liu, K. Vissers, J. Bär, and Z. István, "Achieving 10Gbps line-rate key-value stores with FPGAs," in *HotCloud*, 2013.
[6] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala, "An FPGA Memcached appliance," in *FPGA*, 2013.
[7] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SOCC*, 2010.
[8] I. DDIO, "Intel® data direct i/o technology," http://www.intel.com/content/www/us/en/io/direct-data-i-o.html, 2014.
[9] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting parallelism to scale software routers," in *SOSP*, 2009.
[10] I. DPDK, "Intel Data Plane Development Kit (Intel DPDK)," http://www.intel.com/go/dpdk, 2014.
[11] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *NSDI*, 2014.
[12] B. Fan, D. G. Andersen, and M. Kaminsky, "MemC3: Compact and concurrent memcache with dumber caching and smarter hashing," in *NSDI*, 2013.
[13] I. FlowDirector, "Intel® ethernet flow director," http://www.intel.com/content/www/us/en/ethernet-controllers/ethernet-flow-director-video.html, 2014.
[14] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," in *SIGCOMM*, 2010.
[15] M. Herlihy, N. Shavit, and M. Tzafrir, "Hopscotch hashing," in *Distributed Computing*. Springer, 2008, pp. 350–364.
[16] T. H. Hetherington, M. O'Connor, and T. M. Aamodt, "MemcachedGPU: Scaling-up scale-out key-value stores," in *Proc. SOCC*, 2015.
[17] R. Huggahalli, R. Iyer, and S. Tetrick, "Direct cache access for high bandwidth network I/O," in *ISCA*, 2005.
[18] I. IOAT, "Intel® i/o acceleration technology," http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html, 2014.
[19] R. Jevtic, H. Le, M. Blagojevic, S. Bailey, K. Asanovic, E. Alon, and B. Nikolic, "Per-core DVFS with switched-capacitor converters for energy efficiency in manycore processors," *IEEE TVLSI*, vol. 23, no. 4, pp. 723–730, 2015.
[20] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *SIGCOMM*, 2014.
[21] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable low latency for data center applications," in *SOCC*, 2012.
[22] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.
[23] S. Li, H. Lim, V. W. Lee, J. H. Ahn, A. Kalia, M. Kaminsky, D. G. Andersen, O. Seongil, S. Lee, and P. Dubey, "Architecting to achieve a billion requests per second throughput on a single key-value store server platform," in *ISCA '15*, 2015.
[24] S. Li, H. Lim, V. W. Lee, J. H. Ahn, A. Kalia, M. Kaminsky, D. G. Andersen, O. Seongil, S. Lee, and P. Dubey, "Full stack architecting to achieve a billion requests per second throughput on a single key-value store server platform," *Transactions on Computer Systems. ACM*, 2016.
[25] S. Li, K. Lim, P. Faraboschi, J. Chang, P. Ranganathan, and N. P. Jouppi, "System-level integrated server architectures for scale-out datacenters," in *MICRO*, 2011.
[26] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A holistic approach to fast in-memory key-value storage," in *NSDI*, 2014.
[27] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin Servers with Smart Pipes: Designing SoC accelerators for Memcached," in *ISCA*, 2013.
[28] Linkedin, "How Linkedin uses memcached," http://www.oracle.com/technetwork/server-storage/ts-4696-159286.pdf, 2014.
[29] Y. Mao, E. Kohler, and R. T. Morris, "Cache craftiness for fast multicore key-value storage," in *EuroSys*, 2012.
[30] memcached, "Memcached: A distributed memory object caching system," http://memcached.org/, 2003.
[31] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store," in *USENIX ATC*, 2013.
[32] 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 *NSDI*, 2013.
[33] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "Scale-out NUMA," in *ASPLOS*, 2014.
[34] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in RAMCloud," in *SOSP*, 2011.
[35] R. Pagh and F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, May 2004.

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE

[36] D. A. Patterson, "Latency lags bandwith," *Commun. ACM*, vol. 47, no. 10, pp. 71–75, 2004.
[37] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris, "Improving network connection locality on multicore systems," in *EuroSys*, 2012.
[38] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe, "Arrakis: The operating system is the control plane," in *OSDI*, 2014.
[39] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *USENIX ATC*, 2012.
[40] S. Tanaka and C. Kozyrakis, "High performance hardware-accelerated flash key-value store," in *NVM workshop*, 2014.
[41] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," in *ISCA*, 1996.
[42] Twitter, "Twemcache: Twitter memcached," https://github.com/twitter/twemcache, 2012.
[43] K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, and X. Zhang, "Mega-KV: A case for GPUs to maximize the throughput of in-memory key-value stores," *Proc. VLDB Endow.*, vol. 8, no. 11, Jul. 2015.

Sheng Li is a staff research scientist and a technical lead on enterprise big data at Intel's Parallel Computing Lab, where his research focuses on computer architecture and systems at scale for emerging and demanding applications. He has published more than 30 technical papers and holds 34 patents (both awarded and pending). He was a senior research scientist at HP Labs, where his work influenced the HP moonshot hyperscale server architecture. He has a Ph.D. in Electrical Engineering from the University of Notre Dame and is a senior member of IEEE.

Hyeontaek Lim is a postdoctoral researcher in the Computer Science Department at Carnegie Mellon University. His research interests include resource-efficient and flexible distributed systems, networking, and operating systems. He has a Ph.D. in Computer Science from Carnegie Mellon University. He is a member of IEEE, ACM, SIAM, and USENIX.

Victor Lee is a principal engineer and research scientist at Intel's Parallel Computing Lab leading the research on the next generation processor for HPC and Big Data. His research focuses on improving algorithm efficiency as well as architecture efficiency. Victor joined Intel in 1997 where he worked on the Intel Pentium Pro processor, the Intel Pentium 4 processor, the Intel Itanium processor and the QPI interconnect. He joined Intel Labs in 2002 and spearheaded the many-core research which eventually lead to the Intel Many Integrated Core architecture and the first Intel Xeon Phi coprocessor product. Victor received a B.S. in Electrical Engineering from University of Washington in 1994, S.M. in Electrical Engineering and Computer Science from Massachusetts Institute of Technology in 1996. He is a senior member of IEEE.

Jung Ho Ahn is an associate professor in the Graduate School of Convergence Science and Technology at Seoul National University, where he leads the Scalable Computer Architecture Laboratory. He is interested in bridging the gap between the performance demand of emerging applications and the performance potential of modern and future massively parallel systems. Ahn has a PhD in electrical engineering from Stanford University, and is a senior member of IEEE.

Anuj Kalia is a PhD student at Carnegie Mellon University. His research interests include networked systems and distributed systems. He is part of the Parallel Data Lab (PDL) at CMU, and also works closely with Intel Labs.

Michael Kaminsky is a Senior Research Scientist in Intel Labs and an adjunct faculty member of the Computer Science Department at Carnegie Mellon University. He is part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), based in Pittsburgh, PA. His research interests include distributed systems, operating systems, and networking.

David Andersen is an associate professor in the Computer Science department at Carnegie Mellon University. He received his Ph.D. and M.S. degrees from MIT, and and received B.S. degrees in Computer Science and Biology from the University of Utah. Before joining MIT, he was a co-founder and CTO of an Internet Service Provider in Salt Lake City. His research interests center on computer systems in the networked environment.

Seongil O is a senior engineer at Samsung Electronics. His research interests include memory system and DRAM microarchitecture. O has a PhD in intelligent convergence system from Seoul National University, where he completed the work for this article.

Sukhan Lee is a PhD student in department of transdisciplinary studies at the Seoul National University. His research interests include computer architectures for big data system, memory microarchitecture, and a system on chip. Sukhan has a BS and MS in electrical and electronic engineering from Yonsei University.

Pradeep Dubey is an Intel Fellow and Director of Parallel Computing Lab (PCL), part of Intel Labs. His research focus is computer architectures to efficiently handle new compute- and data-intensive application paradigms for the future computing environment. Dubey previously worked at IBM's T.J. Watson Research Center, and Broadcom Corporation. He has made contributions to the design, architecture, and application-performance of various microprocessors, including IBM® Power PC, Intel® i386™, i486™, Pentium® Xeon®, and the Xeon Phi™ line of processors. He holds over 36 patents, has published over 100 technical papers, won the Intel Achievement Award in 2012 for Breakthrough Parallel Computing Research, and was honored with Outstanding Electrical and Computer Engineer Award from Purdue University in 2014. Dr. Dubey received a PhD in electrical engineering from Purdue University. He is a Fellow of IEEE.

Digital Object Indentifier 10.1109/MM.2016.13 0272-1732/$26.00 2016 IEEE