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

5

SHENG LI, Intel Labs
HYEONTAEK LIM, Carnegie Mellon University
VICTOR W. LEE, Intel Labs
JUNG HO AHN, Seoul National University
ANUJ KALIA, Carnegie Mellon University
MICHAEL KAMINSKY, Intel Labs
DAVID G. ANDERSEN, Carnegie Mellon University
SEONGIL O and SUKHAN LEE, Seoul National University
PRADEEP DUBEY, Intel Labs

Distributed in-memory key-value stores (KVSs), such as memcached, have become a critical data serving layer in modern Internet-oriented data center infrastructure. Their performance and efficiency directly affect the QoS of web services and the efficiency of data centers. Traditionally, these systems have had significant overheads from inefficient network processing, OS kernel involvement, and concurrency control. Two recent research thrusts have focused on 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 also 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 leads to guided optimizations atop a recent design to achieve a record-setting throughput of 120 million requests per second (MRPS) (167MRPS with client-side batching) on a single commodity server. Our system delivers the best performance and energy efficiency (RPS/watt) demonstrated to date over existing KVSs including the best-published FPGA-based and GPU-based claims. We craft a set of design principles for future platform architectures, and via detailed simulations demonstrate the capability of achieving a billion RPS with a single server constructed following our principles.

CCS Concepts: • **Information systems** → **Key-value stores**; • **Networks** → **Cloud computing**; • **Computer systems organization** → **Multicore architectures**; **Client-server architectures**

Additional Key Words and Phrases: Key-value stores, many core, storage performance, cloud and network, energy efficiency

---

This work is an extended version of Li et al. [2015] published in ISCA '15.
This work was supported in part by the National Science Foundation under award CNS-1345305 and by the Intel Science and Technology Center for Cloud Computing. Jung Ho Ahn was supported in part by the National Research Foundation of Korea grant funded by the Korea government (2014R1A2A1A11052936).
Authors' addresses: S. Li, V. W. Lee, M. Kaminsky, and P. Dubey, Intel Labs; emails: {sheng.r.li, victor.w.lee, michael.e.kaminsky, pradeep.dubey}@intel.com; H. Lim, A. Kalia, and D. G. Andersen, Carnegie Mellon University; emails: {hl, akalia, dga}@cs.cmu.edu; J. H. Ahn, S. O, and S. Lee, Seoul National University; emails: {gajh, swdfish, infy1026}@snu.ac.kr.

ACM 0734-2071/2016/04-ART5 $15.00
DOI: http://dx.doi.org/10.1145/2897393

ACM Reference Format:
Sheng Li, Hyeontaek Lim, Victor W. Lee, Jung Ho Ahn, Anuj Kalia, Michael Kaminsky, David G. Andersen, Seongil O, Sukhan Lee, and Pradeep Dubey. 2016. Full-stack architecting to achieve a billion-requests-per-second-throughput on a single key-value store server platform. ACM Trans. Comput. Syst. 34, 2, Article 5 (April 2016), 30 pages.
DOI: http://dx.doi.org/10.1145/2897393

1. INTRODUCTION

Distributed in-memory key-value stores such as memcached [memcached 2003] have become part of the critical infrastructure for large-scale Internet-oriented data centers. They are deployed at scale across server farms inside companies such as Facebook [Nishtala et al. 2013], Twitter [Twitter 2012], Amazon [Amazon 2012], and LinkedIn [Linkedin 2014]. Unfortunately, traditional key-value store (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 [Lim et al. 2013].

As a critical layer in the data center infrastructure, the performance of key-value stores affects the QoS of web services [Nishtala et al. 2013], whose efficiency in turn affects data center 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 [Lim et al. 2013; Blott et al. 2013; Chalamalasetti et al. 2013; Tanaka and Kozyrakis 2014], improve energy efficiency by more than 10 times compared to legacy code on commodity servers. The software-based research [Lim et al. 2014; Fan et al. 2013; Mitchell et al. 2013; Dragojević et al. 2014; Kalia et al. 2014; Mao et al. 2012; Ongaro et al. 2011] 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 [Lim et al. 2014], 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, and (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, key-value (KV) processing, memory management, and concurrency control. On the hardware 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 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* to *network* subsystems. This article

is the first to reveal the important (yet hidden) synergistic implications of modern platform features (e.g., direct cache access [DDIO 2014; Huggahalli et al. 2005], multiqueue NICs with flow steering [FlowDirector 2014], prefetch, and beyond) to high-performance KVS systems. Guided by these insights, we optimize MICA and achieve record-setting performance of 120 MRPS (167MRPS with client-side batching) and energy efficiency of 302 kilo RPS/watt (401 kilo RPS/watt with client-side batching) on our commodity CPU-based system. Our system delivers the best performance and energy efficiency (RPS/watt) demonstrated to date over existing KVSs including the best-published FPGA-based [Tanaka and Kozyrakis 2014] and GPU-based [Zhang et al. 2015; Hetherington et al. 2015] claims. Finally, based on these full-stack insights, we craft a set of design principles for a future many-core-based and throughput-optimized platform architecture, with the right *system balance* among compute, memory, and network. We extend the McSimA+ simulator [Ahn et al. 2013] to support the modern hardware features our proposal relies on and demonstrate that the resulting design is capable of exceeding a billion requests per second on a quad-socket server platform.

2. BACKGROUND AND RELATED WORK

In-memory KVSs compose the critical low-latency data serving and caching infrastructure in large-scale Internet services. KVSs provide a fast, scalable storage service with a simple, generic, hash-table-like interface for various applications. Applications store a key-value pair using `PUT(key,value)` and look up the value associated with a key using `GET(key)`.

KVS nodes are often clustered for load balancing, fault tolerance, and replication [Nishtala et al. 2013]. Because each individual store in the cluster operates almost independently, a KVS cluster can offer high throughput and capacity as demonstrated by large-scale deployments—for example, Facebook operates a memcached KVS cluster serving over a billion requests per second for trillions of items [Nishtala et al. 2013]. However, the original memcached, the most widely used KVS system, can achieve only submillion to a few million requests per second (RPS) on a single modern server [Lim et al. 2013, 2014] because of overheads from in-kernel network processing and locking [Kapoor et al. 2012; Lim et al. 2013]. Recent work to improve KVS performance has explored two different paths: hardware acceleration for stock KVSs (mostly memcached) and software optimizations on commodity systems.

The hardware-based approach uses specialized platforms such as FPGAs. Research efforts [Blott et al. 2013; Lim et al. 2013; Chalamalasetti et al. 2013; Lavasani et al. 2013; Tanaka and Kozyrakis 2014] in this direction achieve up to 13.2MRPS [Blott et al. 2013] with a 10GbE link and more than 10× improvements on energy efficiency compared to commodity servers running stock memcached. There are also non-FPGA-based architecture proposals [Gutierrez et al. 2014; Li et al. 2011; Lotfi-Kamran et al. 2012; Novakovic et al. 2014; Zhang et al. 2015; Hetherington et al. 2015] for accelerating memcached and/or improving the performance and efficiency of various data center workloads.

On the software side, recent work [Lim et al. 2014; Fan et al. 2013; Mitchell et al. 2013; Dragojević et al. 2014; Kalia et al. 2014; Mao et al. 2012; Ongaro et al. 2011] has optimized the major components of KVSs: network processing, concurrency control, key-value processing, and memory management, either in isolation or in combination for better performance. Reducing the overhead of these components can significantly improve performance on commodity CPU-based platforms. As of this writing, the fastest of the new KVS software designs is MICA [Lim et al. 2014], which achieves 77MRPS on a dual-socket server with Intel® Xeon™ E5-2680 processors.[1]

[1]Intel and Xeon are trademarks of Intel Corporation in the United States and/or other countries.

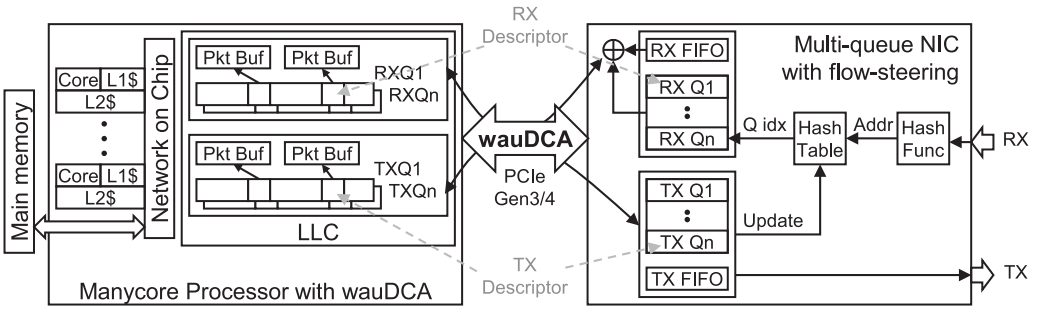

Fig. 1. A modern system with write-allocate-write-update-capable Direct Cache Access (wauDCA), for example, Intel DDIO [DDIO 2014], and a multiqueue NIC with flow steering, for example, Intel Ethernet Flow Director [FlowDirector 2014], to support high-performance network-intensive applications.

3. MODERN PLATFORMS AND THE KVS DESIGN SPACE

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

3.1. Modern Platforms

The core count and last-level cache (LLC) size of modern platforms continue to increase. For example, Intel Xeon processors today have as many as 18 powerful cores with 45MB of LLC. These multi-/many-core 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 [Dobrescu et al. 2009; Pesterev et al. 2012]. 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) [Huggahalli et al. 2005],[2] implemented as Intel Data Direct I/O Technology (Intel DDIO) [DDIO 2014] 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, multiqueue NICs enable efficient parallel transmission from multiple cores by eliminating queue contention, and enable parallel reception by providing flow steering, implemented as Intel Ethernet Flow Director (Intel Ethernet FD) [FlowDirector 2014] 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

[2]This article always refers to DCA as the wauDCA design [Huggahalli et al. 2005] (e.g., Intel Data Direct I/O Technology [DDIO 2014]) instead of the simplified Prefetch Hint [Huggahalli et al. 2005]-based implementation (e.g., Intel I/OAT [IOAT 2014]).

Table I. Taxonomy of Design Space of Key-Value Store (KVS) Systems

Network Stack	Example Eystems
Kernel	memcached [memcached 2003], MemC3 [Fan et al. 2013]
Userspace	Chronos [Kapoor et al. 2012], MICA [Lim et al. 2014]

Concurrency Control	Example Systems
Mutex	memcached, MemC3
Versioning	Masstree [Mao et al. 2012], 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 [Ongaro et al. 2011]
Circular log	MICA

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 nonnetworked 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 [Han et al. 2010] and Intel Data Plane Development Kit (DPDK) [DPDK 2014], can utilize the full capacity of high-speed networks. By eliminating the overheads of heavyweight 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* [Rizzo 2012; Han et al. 2010]. Furthermore, userspace networking can also be kernel managed [Peter et al. 2014; Belay et al. 2014] 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 multigigabit links. Thus, we navigate through the KVS design space to shed light on how KVSs should exploit modern platforms.

3.2. 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 [Fan et al. 2013], Pilaf [Mitchell et al. 2013], MICA [Lim et al. 2014], FaRM-KV [Dragojević et al. 2014], and HERD [Kalia et al. 2014], optimize different parts of the KVS system to improve performance. As a complex system demanding hardware and software codesign, 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 data center with flow-steering-capable networking (e.g., Intel Ethernet FD) has a different subset of essential ingredients of an appropriate KVS design from a data center without it. Table I

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, DPDK and RDMA drivers [DPDK 2014; Mellanox 2014] expose 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 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 [Pagh and Rodler 2004] and hopscotch hashing [Herlihy et al. 2008]. 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 [Fan et al. 2013]; 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()` [memcached 2003], to reduce TLB misses via huge pages [memcached 2003; Fan et al. 2013], and to help enforce the replacement policy [memcached 2003; Lim et al. 2014]. One common scheme is SLAB, which 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 [Ongaro et al. 2011], including a circular log that optimizes memory access for KV insertions and simplifies garbage collection and item eviction [Lim et al. 2014].

It is noteworthy that new KVSs benefit from recent hardware trends described in Section 3.1, especially in their network stack and concurrency control schemes. For example, MICA and HERD actively exploit multiple queues in the NIC by steering remote requests to a specific server core to implement data partitioning, rather than passively accepting packets distributed by RSS. While these systems always involve the server CPU to process key-value requests, they alleviate the burden by directly using the large CPU cache that reduces the memory access cost of DMA significantly.

4. EXPERIMENTAL METHODS

Our ultimate goal is to achieve a billion RPS on a single KVS server platform. However, software and hardware codesign/optimization for KVS is challenging. Not only does a KVS exercise all the main system components (compute, memory, and network), but also the design space of both the system architecture and KVS algorithms and the

Table II. Implementations of the KVS Systems Used in Our Experiments. Mcd-D and MC3-D Are Modified from Their Original Code to Use DPDK for Network I/O. MICA Is Optimized for Higher Throughput and Operates in Its Cache Mode

Name	KVS Codebase	Network Stack	Concurrency Control
Mcd-S	memcached	Kernel (libevent)	Mutex
Mcd-D	memcached	Userspace (DPDK)	Mutex
MC3-D	MemC3	Userspace (DPDK)	Mutex+versioning
MICA	MICA-cache	Userspace (DPDK)	None in EREW/versioning in CREW

		Key-Value Processing		Memory
Name	KVS Codebase	Indexing	Replacement Policy	Management
Mcd-S	memcached	Chained hash table	Strict LRU	SLAB
Mcd-D	memcached	Chained hash table	Strict LRU	SLAB
MC3-D	MemC3	Cuckoo hash table	CLOCK	SLAB
MICA	MICA-cache	Lossy hash index	FIFO/LRU/Approximate-LRU	Circular log

implementation are huge, as described in Section 3. We therefore use a multistage approach. We first optimize the software to exploit the full potential of modern architecture with efficient KVS designs. Second, we undertake rigorous and cross-layer architectural characterization to gain full-stack insights on essential ingredients for both hardware and software for KVS designs, where we also extend our analysis to a collection of KVS designs to reveal system implications of KVS software design choices. Finally, we use these full-stack insights to architect future platforms that can deliver over a billion RPS per KVS server.

4.1. KVS Implementations

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 I, which is a nearly impossible task. Fortunately, Lim et al. [2014] have explored the design space to some extent and demonstrated that their MICA design achieves 77MRPS on a single KVS server, orders of magnitude faster than other KVSs. Thus, we take MICA as the starting point for optimization (leaving RDMA-based KVS designs for future work) to fully exploit the potential of modern platforms and include popular memcached [memcached 2003] and MemC3 [Fan et al. 2013] (a major yet nondisruptive improvement over memcached) to study the system implications of KVS design choices. Table II gives an overview of the KVS implementations used in this work.

Mcd-S is the original memcached. This implementation is commonly used in numerous studies. Mcd-S uses socket I/O provided by the OS and stores key-value items in SLAB. It uses multiple threads to access its key-value data structures concurrently, which are protected by fine-grained mutex locks. It uses a chained hash table to locate items and maintains an LRU list to find items to evict.

Mcd-D is a DPDK-version of Mcd-S. It replaces the network stack of Mcd-S with a userspace networking stack enabled by DPDK and advanced NICs to perform efficient network I/O. It reuses other parts of Mcd-S, that is, concurrency control and key-value data structures.

MC3-D is a DPDK version of MemC3. MemC3 replaces the hashing scheme of memcached with the more memory-efficient *concurrent cuckoo hashing*, while still using fine-grained mutex for interthread concurrency control. It also substitutes the strict LRU of memcached with CLOCK, which eliminates the high cost of LRU updates. While these changes triple its throughput [Fan et al. 2013], MC3-D still suffers from overhead caused by the code base of the original memcached.

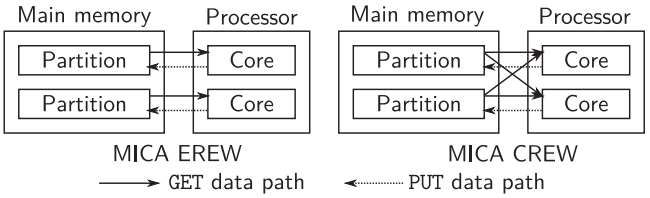

Fig. 2. Partitioning/sharding in MICA.

Table III. Workloads Used for Experiments

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

MICA is a KVS that uses a partitioned/sharded design and high-speed key-value structures. 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 (while keeping the same constraint for `PUT` requests), which requires MICA to use a versioning-based optimistic locking scheme. We use MICA's cache mode, which is optimized for memcached-like workloads; MICA maintains a lossy index that resembles the set-associative cache of CPUs for locating key-value items, and a circular log that stores variable-length items with FIFO eviction. In this design, 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 3.1 and making clients specify the partition of the key explicitly in the packet header. MICA supports FIFO, LRU, and approximate LRU by selectively reinserting recently used items and removing most inactive items in the circular log. MICA performs intensive software memory prefetching for the index and circular log to reduce stall cycles.

While MICA is for co-optimizing hardware and software to reap the full potential of modern platforms, other KVS designs are important to understand system implications of key design choices. For example, from Mcd-S to Mcd-D, we can see the implications on moving from OS to userspace network stack. And from Mcd-D to MC3-D, we can find the implications for using more efficient key-value processing schemes over traditional chaining with an LRU policy.

4.2. Experimental Workloads

We use YCSB for generating key-value items for our workload [Cooper et al. 2010]. While YCSB is originally implemented in Java, we use MICA's high-speed C implementation, which can generate up to 140MRPS using a single machine. The workload has three relevant properties: average item size, skewness, and read-intensiveness. Table III summarizes four different item counts and sizes used in our experiment. The packet size refers to the largest packet size including the overhead of protocol headers (excluding the 24-byte Ethernet PHY overhead); it is typically the `PUT` request's size because it carries both the key and value, while other packets often omit one or the other (e.g., no value in `GET` request packets). To demonstrate realistic KVS performance for large datasets, we ensure that the item count in each dataset is sufficiently high so that the overall memory requirement is at least 10GB, including per-object space

overhead. The different datasets also reveal implications of item size in a well-controlled environment for accurate analysis. We use relatively small items because they are more challenging to handle than large items, which rapidly become bottlenecked by network bandwidth [Lim et al. 2013]. They are also an important workload in data center services (e.g., Facebook reports that in one memcached cluster [Atikoglu et al. 2012], "requests with 2-, 3-, or 11-byte values add up to 40% of the total requests").

We use two distributions for key popularity: Uniform and Skewed. In Uniform, every key has equal probability of being used in a request. In Skewed, the popularity of keys follows a Zipf distribution with skewness 0.99, the default Zipf skewness for YCSB. The Zipf distribution captures the key request patterns of realistic workloads [Atikoglu et al. 2012; Lim et al. 2013] and traces [Lim et al. 2013]. Skewed workloads often hit system bottlenecks earlier than uniform workloads because they lead to load imbalance, which makes them useful for identifying bottlenecks.

Read-intensiveness indicates the fraction of `GET` requests in the workload. We use workloads with 95% and 50% GET ratios to highlight how KVSs operate for read-intensive and write-intensive applications, respectively.

We define the `STANDARD` workload as a uniform workload with tiny items and a 95% GET ratio. This workload is used in several of our experiments later.

4.3. Experiment Platform

Our experiment system contains two dual-socket systems with Intel® Xeon™ E5-2697 v2 processors (12 core, 30MB LLC, 2.7GHz). These processors are equipped with Intel DDIO (an implementation of wauDCA on Intel processors) and thus enable NICs to inject network I/O data directly into LLC. Each system is equipped with 128GB of DDR3-1600 memory and four Intel® X520-QDA1 NICs, with four 10Gbps Ethernet (10GbE) ports on each NIC. The NICs support flow steering via the built-in Intel Ethernet Flow Director. The two systems are directly connected via their NICs for simplicity, with one system acting as a client and the other acting as a server.

CentOS 7.0 is installed with kernel 3.10.0-123.8.1. All code is compiled with gcc 4.8.2. For application-, system-, and OS-level analysis, we use Systemtap 2.4. For hardware-level performance analysis, we use Intel® VTune™ to collect statistics from hardware performance counters. We measure the total power supplied to the server from a wall socket using Watts-Up-Pro. Inside the server, we use a National Instruments DAQ-9174 to measure the power of the two processors (via the 12V rail of the voltage regulator) and one of the PCIe NICs (via the 3.3V and 12V rails on the PCIe slot).

5. THE ROAD TO 120 MILLION (160 MILLION WHEN WITH CLIENT-SIDE BATCHING) RPS PER KVS SERVER

We first describe our optimizations guided by detailed full-system characterization, achieving 120MRPS 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. We also demonstrate that our system can achieve 160 MRPS performance when with client-side batching and provide detailed analysis on the system implications of client-side batching and assistance for high performance KVSs. Finally, we perform detailed comparison among state-of-the-art high performance KVSs, including our design.

5.1. 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

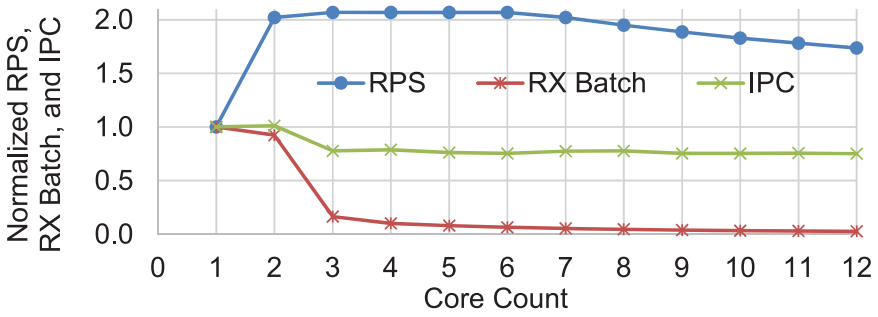

Fig. 3. MICA's number-of-cores scalability with one 10GbE port and one socket. We use `STANDARD` workload and EREW mode. RX batch size is the average number of packets MICA fetches per I/O operation from the NIC via DPDK because RX packets may queue up before CPUs process them, which represents the amount of KV processing work per I/O operation. (This is the natural queuing process between CPU and NIC in all network processing and is completely different from client-side batched KVS operations. Section 5.5 will discuss client-side batching.) All numbers are normalized to their values at one core, where the actual RPS, RX batch size, and IPC are 5.78MRPS, 32 packets per I/O operation, and 1.91 (IPC per core), respectively.

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 9.76Gbps out of the 10Gbps 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 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 80MRPS with 16 cores and eight 10GbE 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 [DDIO 2014]. 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. With this new mapping, the performance reached 100MRPS with 20 cores and ten 10GbE ports, but dropped off slightly with more cores/ports. We analyzed this problem in detail by using Systemtap to track the complete behavior (on-/off-CPU time, call graph, execution cycle breakdown, among others) of all procedure calls and threads in the entire software stack (MICA, DPDK, and OS). We found that several housekeeping functions consumed more than 20% of the execution time when there are more than 20 cores. Examples include statistics collection from NICs (used for flow control, and expensive because of MMIO) and statistics collection from local processors (for performance statistics). We reduced the frequency of these housekeeping tasks to alleviate the overhead without affecting the main functionality. With this optimization, MICA scaled linearly with number of cores and number of ports.

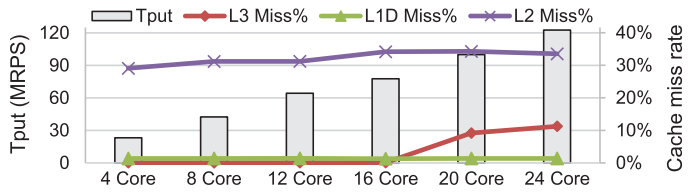

Fig. 4. Throughput scalability and cache[3] miss rates of MICA with our optimizations. The port count used is half the core count. We use `STANDARD` workload and EREW mode.

Table IV. MICA's Resource Utilization. Cores and Ports Are Evenly Distributed Across the Two Sockets. We Use `STANDARD` Workload with EREW Mode

Ports	Cores	Network BW (Gb/s) TX/RX	Mem BW (GB/s) RD/WR	Tput (MRPS)
2 10GbE	4	19.31/19.51	6.21/0.23	23.33
12 10GbE	24	99.66/105.45	34.97/2.89	120.5

Figure 4 shows MICA's throughput with our optimizations. MICA achieves 120MRPS when all 24 cores in both sockets are used. With increasing numbers of cores and ports, L1D and L2 cache misses remain stable, at ∼1.5% and ∼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 (eight per socket). The increased LLC miss rate happens for the same reason that prevents us from increasing beyond 80MRPS before applying the core-to-port mapping optimization, which indicates the importance of sufficient LLC capacity for future many-core processors for high KVS performance even with the mapping optimization.

Hereafter, we refer to MICA with our optimizations as MICA for simplicity. Table IV shows the utilization of hardware components on the dual-socket system with two configurations: two ports with four cores, and 12 ports with 24 cores. The cores and ports are evenly distributed across two NUMA domains. The resource utilization scales almost linearly as more cores and ports are used with the fixed 2:1 core-to-port ratio. For example, the memory bandwidth increases from 6.21GB/s with two ports to 34.97GB/s with 12 ports.

We also performed an architectural characterization of the system implications of simultaneous multithreading (SMT) [Tullsen et al. 1996] on our KVS performance, using Intel Hyperthreading Technology, an implementation of two-way SMT on Intel processors. Our characterization shows that two-way SMT causes a 24% throughput degradation with the full system setup (24 cores and twelve 10GbE ports). This degradation is because, under the full system setup, 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 3.6× 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.

[3]Unlike memcached with 20+% L1I$ miss rate due to the complex code path in the Linux kernel and networking stack [Lim et al. 2013], MICA's L1I$ miss rate is below 0.02% due to the use of userspace networking and kernel bypass.

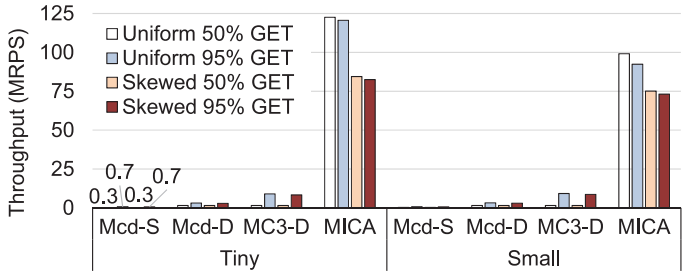

Fig. 5. Throughput of the four KVS systems with different datasets. For all systems, key-value hit rate is within 98%–99.6%, and the 95^{th} percentile of latency is less than 100μs. MICA runs EREW mode.

Table V. KVS Configurations to Achieve Best Performance: Mcd-S, Mcd-D, and MC3-D Have the Same Optimal Configuration

	# NUMA	# Cores/Domain	# 10GbE Ports/Domain
Mcd/MC3-D	2	4	1
MICA	2	12	6

5.2. System Implications of KVS SW Design Choices

Figure 5 shows the measured full-system performance of the four KVS systems (Table II) with tiny and small datasets (Table III) and different GET/PUT ratios. All four systems use huge pages in virtual memory to ensure negligible impact from TLB misses. MICA performs the best regardless of datasets, skew, and GET ratios. For tiny key-value pairs, MICA's throughput reaches 120.5 to 116.3MRPS with the uniform workload and 84.6 to 82.5MRPS for the skewed workload. MICA uses 110 to 118Gbps of network bandwidth under the uniform workload, almost saturating the network stack's sustainable 118Gbps bandwidth on the server (when processing packet I/O only). Other KVSs achieve 0.3 to 9MRPS for the tiny dataset. Because the the system remains the same (e.g., 120GbE network) for all datasets, using larger item sizes shifts MICA's bottleneck to network bandwidth, while other KVSs never saturate network bandwidth for these datasets. Since larger items rapidly become bottlenecked by network bandwidth and thus are much easier to handle even for inefficient KVSs [Lim et al. 2013], large and extra-large datasets have similar results, with shrinking gaps between MICA and other KVSs as the item size increases.

Because of the inherent characteristics of their different design choices, the KVS systems achieve their best performance with different system balances. We sweep the system-resource space for all four KVSs to find their balanced configuration, shown in Table V. While MICA can leverage all 24 cores with twelve 10GbE ports in the tested server, Mcd-S, Mcd-D, and MC3-D can only use four cores and one 10GbE port per domain due to their inherent scalability limitations (Section 5.2.1). Because MICA uses NUMA-aware memory allocation for partitions [Lim et al. 2014], we run other systems with two processes, one on each NUMA domain (with different 10GbE ports), to compare the aggregate performance more fairly.

5.2.1. Inside KVS Software Stack and OS Kernels. Despite MICA's high throughput, it is critical to understand its performance deeply via holistic cross-layer analysis. Figure 6 shows the execution time breakdown between the four major components of KVS software (Section 3.2), obtained by Systemtap. With the best configuration (BT), Mcd-S spends more than 60% of its execution time on network processing because of the high overhead of the kernel's network stack. This result is in line with observations from previous studies on memcached [Kapoor et al. 2012]. The pthread mutex-based concurrency control in Mcd-S consumes about 11% of the execution time and memory

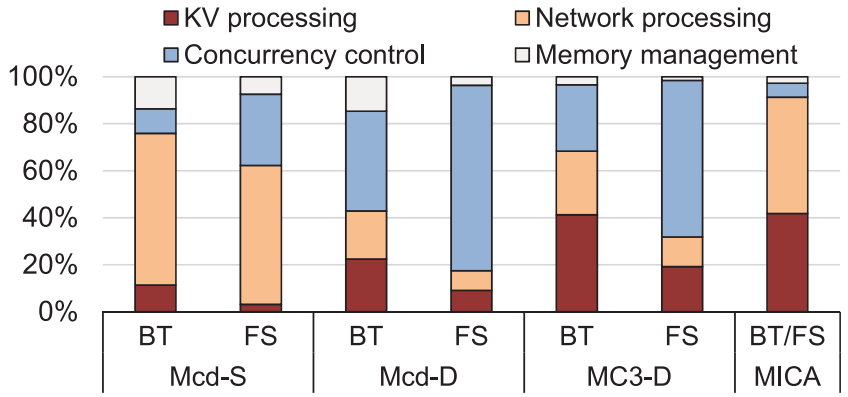

Fig. 6. Execution cycle breakdown of different KVS servers. BT (BesT) refers to the configuration that achieves the best performance (Table V) and FS (Full System) refers to the configuration with 24 cores and twelve 10GbE ports. Experiment settings: STANDARD workload is used; MICA is in EREW mode.

management consumes 13%. As a result, key-value processing work only gets about a 10% share of the execution time, leading to the low performance of Mcd-S (0.3MRPS, Figure 5).

Replacing the kernel's network stack by a more efficient, user-level network stack improves performance, but it is not enough to achieve the platform's peak performance. For example, Mcd-D replaces memcached's network stack by Intel DPDK. This increases throughput dramatically from 0.3MRPS to 3.1MRPS, but it is still less than 3% of MICA's peak throughput. This behavior is because, with the user-level network stack, memcached's bottleneck shifts from network I/O to the heavyweight mutex-based concurrency control. As a result, the actual KV processing still consumes only 26% of the total execution time.

MC3-D attempts to modify memcached's data structures for better concurrent access, leading to a tripled throughput (up to 9MRPS). However, it still performs costly concurrency control, which consumes ∼30% of its execution time. While MC3-D seems to achieve a relatively balanced execution-time breakdown with its best configuration (BT in Figure 6) that uses eight cores and two ports as in Table V, there is significant imbalance with the full system configuration (FS). In the FS mode, Mcd-S, Mcd-D, and MC3-D spend a much smaller share of execution time in key-value processing than in the BT mode, and actually get 2 to 3× *less* performance than the BT mode. MICA shows the most balanced execution time breakdown, with both network and KV processing taking ∼45% of execution time, respectively. This analysis reveals the underlying reason that replacing one component in the complex KVS software is not enough and a holistic redesign of KVSs is the right approach to achieve high performance.

5.3. Key Implications on Modern Platforms Running Optimized MICA

Trade memory bandwidth for latency via prefetching: MICA is very sensitive to memory latency because it must finish the KV processing before the next batch of incoming packets is injected to LLC by the NICs. If it fails to do so, the packet FIFO in the NIC will overflow. The overflow information is collected by the MICA server that subsequently notifies its clients to slow down, which in turn degrades the system performance. MICA relies on multistaged software (SW) prefetch on both packets and KV data structures to reduce latency and keep up with a high-speed network.

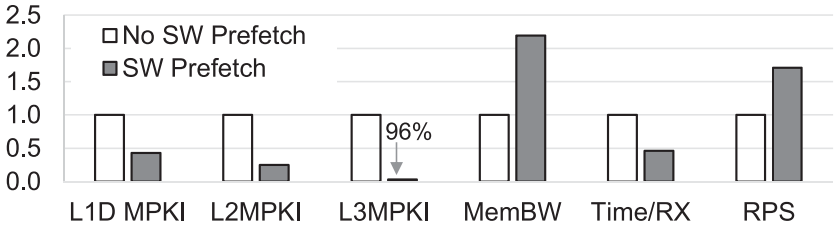

Fig. 7. Implications of prefetch on MPKIs of L1/L2/L3 caches, memory bandwidth (MemBW), time spent on each RX batch (time/RX), and throughput (RPS). All numbers are normalized to that without prefetch. Optimized MICA runs in EREW mode and uses STANDARD workload.

Figure 7 shows the system implications of the multistaged SW prefetch.[4] With SW prefetch, MPKIs of L1D decrease by more than 50%. Because prefetching bypasses L2, the elimination of interferences from both the packet data accesses and the random key-value accesses reduces L2 misses, leading to a 75% reduction in L2 MPKI. Most LLC misses come from the KV data structures, because NICs inject the RX network packets directly into the LLC with sufficient LLC capacity for Intel DDIO (thus, accesses usually do not cause any misses). Because of the randomness in requested keys, LLC has a high cache miss rate without SW prefetch (57%), similar to that observed in other KVSs [Lim et al. 2013]. SW prefetch reduces the LLC miss rate dramatically to 8.04% and thus frees may LLC-miss-induced stall cycles to do KV processing, which improves performance (RPS) by 71% and reduces LLC MPKI by 96%, as shown in Figure 7.

At the system level, the NICs and CPUs form a high-speed hardware producer-consumer pipeline via Intel DDIO. The reduction of cache misses significantly improves latency for consuming requests/packets, eliminating 54% of the time needed to process an RX packet batch, leading to a 71% performance gain. These improvements come at the expense of increasing memory bandwidth use to 34.97GB/s. While the increased memory bandwidth use is mostly due to the performance gain, SW prefetch generates extra memory traffic due to potential cache pollution. For each key-value request, MICA needs one to two random DRAM accesses, for a total of three to four cache lines (some cache lines are adjacent and do not cause DRAM row-buffer conflicts). The network packet that contains the request has high access locality since it is placed in a contiguous segment inside LLC directly by the NICs. Thus, the 120MRPS performance requires ~30GB/s memory bandwidth,[5] which means SW prefetch adds ~17% overhead to memory bandwidth. However, trading memory bandwidth for latency is favorable, because memory latency lags bandwidth significantly [Patterson 2004]. Section 6.1 demonstrates how trading memory bandwidth for latency simplifies the memory subsystem design for future KVS platform architecture.

System implications of skewed workloads: Unlike uniformly distributed (or simply uniform) workloads that evenly spread requests to all partitions, skewed workloads cause uneven load on cores/partitions and create hot and cold cores with different throughput. A cold core spends more time spin-waiting for jobs from external sources, which results in different instruction mixes than on hot cores. Therefore, using traditional metrics such as IPC and cache miss rate for skewed workloads could be misleading. Instead, Figure 8 uses instructions per KV (key-value) operation (IPO) and

[4]MICA uses nontemporal software prefetch, prefetchnta, to bypass L2 because the large and random dataset does not benefit from a small-sized L2. L3 is inclusive of L1 and thus not bypassed. Because of Intel DDIO, packets are injected to L3 directly; thus, not bypassing L3 is naturally better.

[5]Although MICA cannot achieve 120MRPS without SW prefetch, we verified the relationship between throughput and memory bandwidth demand at lower achievable throughput levels with SW prefetch turned off.

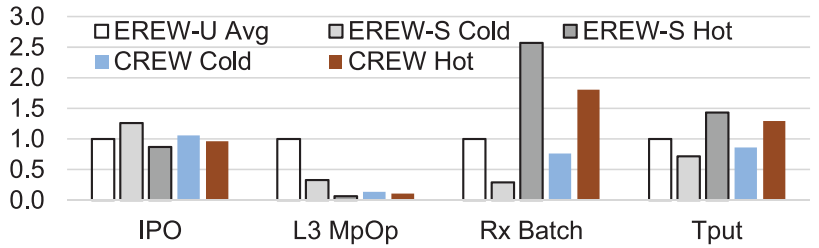

Fig. 8. Behavior of skewed workloads. All results are for a single core. Optimized MICA runs in EREW and CREW modes, using tiny items (8-byte keys and 8-byte values) with 95% GET. Both uniform and skewed workloads are used, with -U and -S, meaning uniform and skewed workloads, respectively. Hot and cold refer to hot and cold cores based on their load. For Rx Batch and Tput, higher is better. For IPO and MpOp, lower is better.

cache misses per KV operation (MpOp), together with overall performance for skewed workloads. We focus on per-core behavior because it differentiates hot and cold cores, which affects overall performance. We normalize to EREW with a uniform workload as the baseline; its whole-system throughput is 120MRPS.

With skewed workloads, the EREW throughput is ∼84MRPS. The per-core throughput of cold cores decreases by 28% to 3.58MRPS on average. The hot cores' throughput, however, increases by 43% to 7.1MRPS, which mitigates the system impact of the skew. The increased locality of the requests in the skewed workload reduces L3 MpOp of the hot cores by over 80%, compared to the cold cores, as shown in Figure 8. Moreover, the hot cores' packets per I/O almost triples from 12.5 packets per I/O with uniform workload to 32 packets per I/O, which reduces the per-packet I/O cost and results in the 14% improvement on IPO on hot cores. While EREW tolerates skewed workloads well, mode CREW in MICA further bridges the gap. In CREW, all cores receive and process GET requests regardless of their partition affinity. The bottleneck due to the hot cores for GET heavy workloads nearly disappears, and the different load on the hot and cold cores is due to PUTs and associated synchronization between GETs and PUTs. CREW generates 86% (4.3MRPS) and 129% (6.5MRPS) throughput/core for cold and hot cores, respectively, compared to the uniform EREW mode. This brings the overall system performance back to 108MRPS, a 10% performance drop from the uniform workload. CREW shows the same trend as EREW, benefiting from the increased locality (MpOp reduction) and reduced I/O overhead (increased RX batch size and reduced IPO) on hot cores.

5.4. Round-Trip Latency (RTT) Versus Throughput

High throughput is only beneficial if latency SLAs (service-level agreements) are satisfied. All the results shown so far are guaranteed with the 95^{th} percentile of latency being less than 100μs. Figure 9 reveals more latency-versus-throughput details. As throughput changes from 10M to 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 (120MRPS) achieves over *two orders of magnitude higher* performance than stock memcached (0.3MRPS). Moreover, even at the highest throughput (120MRPS), the 95^{th} percentile latency of MICA is only 96μs, ∼11× better than the 95^{th} percentile latency of 1,135μs reported by Facebook [Nishtala et al. 2013].

The high system utilization at 120MRPS throughput takes a toll on tail latencies, with 99^{th} and 99.9^{th} percentile latencies at 276μs and 952μs, respectively. However, these latencies are better than widely accepted SLAs. For example, MICA's 99^{th} percentile latency is ∼72× better than the 99^{th} percentile latency of 20ms reported by

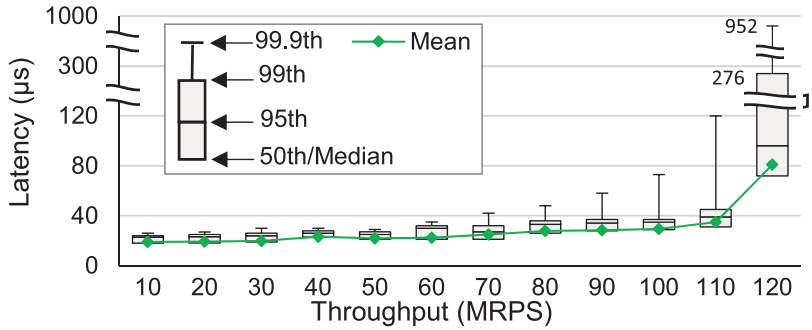

Fig. 9. Round-trip latency (RTT) (including mean and 50^{th}, 95^{th}, 99^{th}, and 99.9^{th} percentiles) for different throughputs. STANDARD workload and MICA's EREW mode are used. Mean is always larger than median, because of the tailing effect. Experiments repeat multiple times to eliminate run-to-run variations.

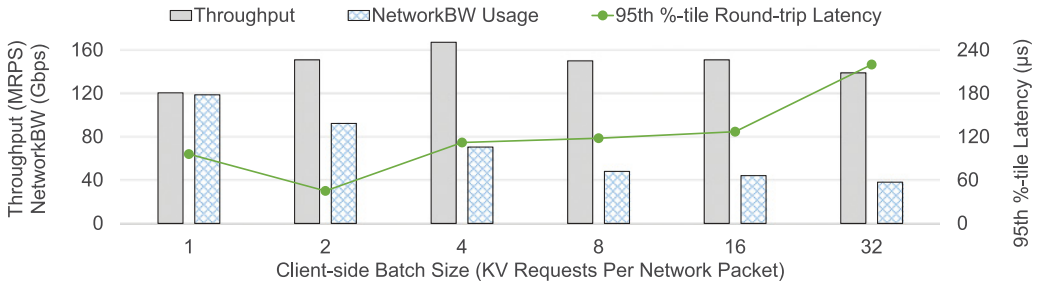

Fig. 10. Performance (throughput), network bandwidth usage, and 95^{th} percentile latency with different client-batch sizes in terms of KV requests per network packet. Optimized MICA runs in EREW mode and uses STANDARD workload.

Netflix [Netflix 2012]. Moreover, a small sacrifice of throughput (8.3%, from 120MRPS to 110MRPS) improves 99^{th} and 99.9^{th} percentile tail-end latencies to 45μs and 120μs, respectively.

5.5. Implications of Client-Side Batching

Client-side batching is a common technique used in KVSs to let clients pack multiple key-value requests into a single network query packet. One well-known example of client-side batching is the "getMulti()" API in memcached. Although client-side batching can amortize network I/O overhead and save network bandwidth, it can introduce additional client-side delays for clients to accumulate enough packets to form a batch; this problem is particularly noticeable in large-scale deployments with high fan-out. In this section, we demonstrate the implications of client-side batching on our system in particular and on large-scale KVS deployments in general.

Figure 10 demonstrates the performance, network bandwidth usage, and 95^{th} percentile latency of our optimized MICA with different client-batch sizes. The network bandwidth usage decreases monotonically because the packet and application header overhead are amortized over more KV requests with increased batch size. Our system throughput increases from 120.5MRPS to 167.2MRPS when batch size increases from one to four. This throughput improvement comes from the reduced network I/O overhead: as the number of network packets per request decreases, the CPU can spend more computing power on KV processing than on packet processing. This additional CPU headroom produces an increase in throughput, plus a reduction in latency since the CPUs can process more KV requests before RX queues start to fill up. Once the batch

size is larger than four, however, the throughput starts to drop, reaching 139MRPS with 32 KV requests per batch. This decrease is because large batch sizes produce larger bursts of KV requests for the CPU to process; our current MICA tries to process these bursts all at once, which exceeds the number of memory accesses that the CPU architecture can keep in flight simultaneously. Exceeding this limit produces stalls that lead to throughput degradation. These stalls, combined with the fact that each packet holds more requests, also increase end-to-end latency since more requests stay in RX queues before being processed. The 95^{th} percentile latency is more sensitive than throughput to system load (including the fullness of RX queues); therefore, the round-trip latency starts to increase at two requests per packet (where the throughput still improves). This result offers guidance on the optimal client-side batch size for optimized MICA in our experimental setup: a batch size of four achieves the best throughput with relatively low 95^{th} percentile latency. We use four as the batch size in our subsequent studies on client-side batching.

Client-side batching increases client-side latency because clients must accumulate enough requests that are destined to the same server/partition before sending a network packet. The mathematical expectation of this client-side latency is estimated by Equation (1):[6]

$$E\left[\textit{Client-side latency due to batching}\right] = \frac{1}{b}\sum_{i=1}^{b}\frac{(b-i)\times \textit{partitions per server}\times \textit{fan-out}}{\textit{client throughput}}, \tag{1}$$

where $b = \textit{KV request per packet}$ defines the average number of KV requests packed in a network query packet; *partitions per server* is the number of exclusive partitions in each server;[7] *client throughput* is the rate for a client to generate KV requests for all servers; and *fan-out* defines how many servers a client can potentially contact, which is the size (node count) of the KVS server pool. In the equation, $b - i$ is the number of subsequent requests to the i^{th} request in the same network query packet. *partitions per server* $\times$ *fan-out*/*client throughput* estimates the mean delay to observe another request for the same server and partition.

Examining Equation (1) reveals how various factors contribute to the cost of client-side batching. As the batch size (b) increases, the expected delay will increase (almost) linearly; with no batching ($b = 1$), the client-side latency is zero. The client-side latency is also linearly proportional to the total number of partitions in the KVS server pool, which illustrates the high cost of client-side batching with high fan-out. Similarly, a lower request rate by a client would make client-side batching more expensive by making it take more time to reach a certain batch size for a particular server partition.

The client machines used in our evaluation have 24 partitions/cores per server, high *client throughput* of 250MRPS, and a low *fan-out* of one. Based on Equation (1), the average client-side latency from client-side batching in our setup is negligible, varying from 0.1μs to 1.6μs as client-side batch size changes from two to 32. This result shows that the latency results in Figure 10 are dominated by server-side behavior. This *artificially* low client-side latency that derives from client-side batching is common in several KVS studies (e.g., KVSs [Zhang et al. 2015; Hetherington et al. 2015; Tanaka and Kozyrakis 2014; Blott et al. 2013] as shown in Table VI) that use a few fast client machines to emulate many slow clients. Figure 13 will present a detailed study on

[6]The expectation on latency over all the different latencies experienced by different requests in the same batch. For example, the first request in the batch experiences the longest client-side latency, while the last request in the same batch does not suffer from any extra client-side latency.

[7]For MICA, each core is a partition in EREW mode, while for memcached, an entire server is a partition.

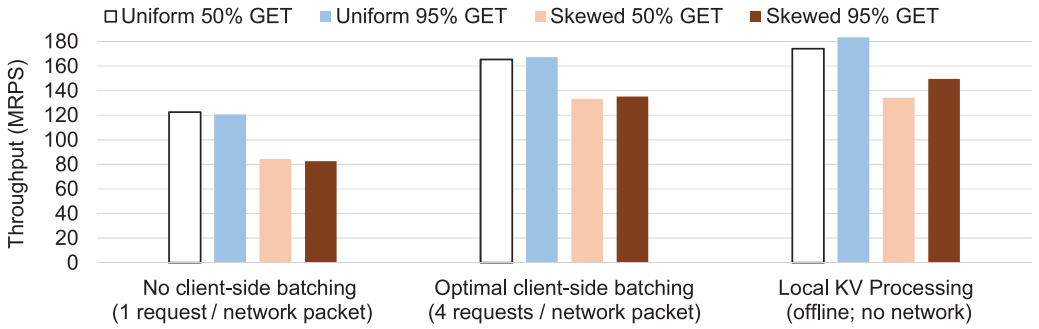

Fig. 11. System performance (throughput) with and without client-side batching as well as local KV processing without network. Optimized MICA runs in EREW mode and uses tiny (8-byte key and 8-byte value) items. Client-side batch size is four KV requests per network packet. Local performance of KV processing (without packet processing, but with local request generation) suggests an upper bound for the end-to-end system throughput (with packet processing, but without local request generation on the server).

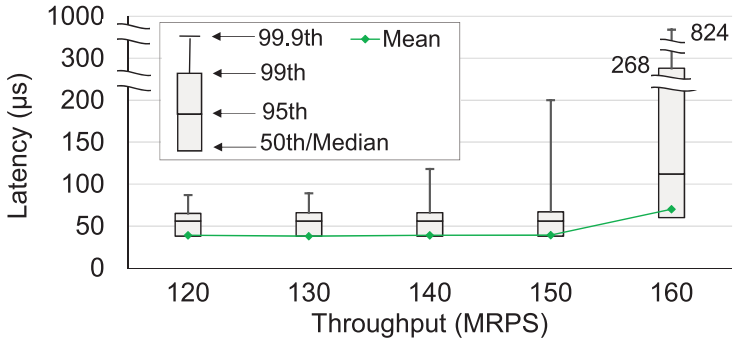

Fig. 12. Round-trip latency (RTT) (including mean and 50^{th}, 95^{th}, 99^{th}, and 99.9^{th} percentiles) for different throughputs. STANDARD workload and MICA's EREW mode are used. Mean is always larger than median, because of the tailing effect. Experiments repeat multiple times to eliminate run-to-run variations. Client-side batch size is four KV-requests per network packet.

client-side latency in large-scale KVS deployments with different client throughput and fan-out.

Figure 11 shows our system performance when running optimized MICA with and without client-side batching. We also include the performance of local KV processing, where KV requests are generated locally in memory on the server without going through network. The local KV processing performance suggests an upper bound for the system throughout.[8] Client-side batching increases throughput in all cases by 39–64% as shown in Figure 10. For example, the performance for the uniform workload with 95% GET increases from 120.5MRPS to 167.2MRPS, and the performance of the skewed workload with 95% GET increases from 82.5MRPS to 135.1MRPS. Because client-side batching helps shift computing power from network processing to KV processing, the end-to-end system performance with client-side batching is much closer to the performance of local KV processing.

Although client-side batching improves system throughput for our optimized MICA, the high throughput is beneficial only if the system can satisfy latency SLAs. Figure 12 presents the latency-versus-throughput curve, similar to the case without client-side

[8]Note that such local request generation is not completely free; therefore, the local KV-processing performance is only approximate and does not serve as a true upper bound of the system throughput.

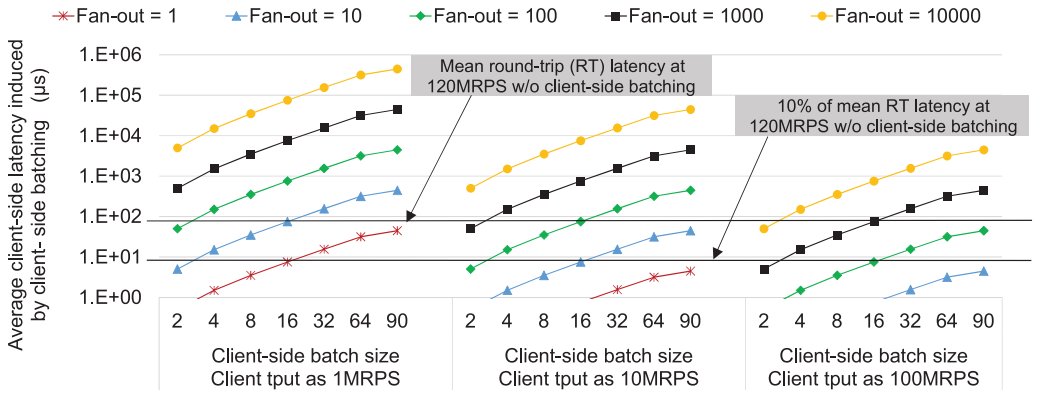

Fig. 13. Average client-side latency induced by client-side batching with different client throughput and fan-out, computed using Equation (1) assuming one partition for each server. This extra latency will be part of the end-to-end round-trip latency if client-side batching is used. The key and value sizes are assumed to be 8 bytes. Client throughput is the rate for a client to generate KV requests for a particular server. The fan-out defines how many servers a client can potentially contact, which is the size (node count) of the KVS server pool.

batching (see Figure 9). Since our optimized MICA achieves 120MRPS without client-side batching and the purpose of client-side batching is to save network overhead so as to improve performance, we focus exclusively on the regime where system performance is higher than 120MRPS.

With client-side batching, our optimized MICA achieves the highest single-server throughput demonstrated to date, with robust SLA guarantees. At 120MRPS without client-side batching, optimized MICA has a relatively high 95^{th} percentile round-trip latency of 96μs (see Figure 9). Because client-side batching allows the system to spend its computing resources on KV processing instead of network processing, the 95^{th} percentile round-trip latency at 120MRPS throughput is 56μs, a 41.7% reduction compared to no client-side batching. As throughput increases from 120MRPS to 167MRPS, latency increases gracefully; the mean goes from 39 to 70μs and the 95^{th} percentile from 58 to 112μs. As in the nonbatched case, higher system throughputs take a toll on tail latencies. At 167MRPS throughput, the 99^{th} and 99.9^{th} percentile latencies are 268μs and 824μs, respectively. Also similar to the nonbatched case shown in Figure 9, a small decrease in throughput (9.5%, from 167MRPS to 151MRPS) improves 99^{th} and 99.9^{th} percentile tail latencies to 67μs and 200μs, respectively.

An important consequence of client-side batching is increased client-side delays; these delays arise because clients must accumulate enough packets for batching, particularly in large-scale deployments. Our previous results show that client-side batching imposes a negligible client-side latency penalty in our experimental setup, while the penalty can become significant if clients are slow and/or each client contacts many nodes in the KVS server pool. Figure 13 uses Equation (1) to show the implications of latency caused by client-side batching on KVS deployments with different client throughput and fan-out. Although MICA treats each core as a partition in EREW mode, it uses the entire server as a single partition in concurrent access mode. Widely used KVSs such as memcached also treat an entire server as one partition. For simplification and generalization, the results in Figure 13 therefore assume one partition for each server (having multiple partitions per server does not change the conclusion of this study, only strengthens it). An MTU-sized Ethernet packet can contain up to approximately 90 key-value pairs with 8-byte keys and values; thus, we explore batch sizes from two to 90 with a power of two as the incremental step.

As shown in Figure 13, client-side batching can introduce a large latency penalty with low client throughput and/or high fan-out. For example, with client throughput as low as 1MRPS and large-scale fan-out of 10,000 [Nishtala et al. 2013; Facebook 2014], even the minimum batch size of two leads to an average extra client-side delay of 5,000μs—about two orders of magnitude more than the average round-trip latency without client-side batching. Although our client can achieve a 250+ MRPS injection throughput, achieving this rate in real large-scale data centers is challenging since KVS clients often do additional CPU-intensive work (e.g., running web servers) in addition to KVS lookups. Figure 13 also shows the full and 10% of mean round-trip latency without client-side batching. If we want the extra average client-side latency to be less than 10% of the mean round-trip latency without client-side batching (i.e., client-side batching will increase the total average round-trip latency by no more than 9%), no client-side batching can be done for clients with 1MRPS throughput and fan-out higher than 10. For large-scale real KVS deployments with thousands of nodes [Nishtala et al. 2013; Facebook 2014], the long extra average latency induced by client-side batching is not acceptable even with 10$\times$ to 100$\times$ higher (10MRPS or even 100MRPS) client throughput. Therefore, although client-side batching looks promising in a controlled lab environment with emulated clients performing fast workload generation, a clear understanding of its implications (as shown in Figure 13) and careful assessment are necessary to achieve desired outcomes in real large-scale deployments.

5.6. Implications of Client-Side Offloading/Assistance

Besides client-side batching, client-side offloading/assistance is another common technique to improve KVS performance and efficiency. Although client-side offloading/assistance can reduce server side bottlenecks for performance improvement, a good client-side offloading/assistance technique should maintain system balance, especially when the systems need to deal with scenarios with and without client-side offloading. For example, MICA uses client-assisted hardware request direction to send KV requests to the target core/partition. Several recent RDMA-based KVSs [Mitchell et al. 2013; Dragojević et al. 2014] are another kind of client-side offloading, where clients use CPU bypass to read server memory and process the results locally. Mega-KV [Zhang et al. 2015] uses a "getk" API to offload the final key comparison (to confirm a potential match) from servers to clients for higher throughput.

The client-side key matching for "getk" increases network traffic because the server must send both the key and value for each potential match. When using large packets to amortize packet head overhead, this network bandwidth increases because "getk" can be $\sim$50% when keys and values are equally sized or even higher when keys are much longer than values (e.g., $\sim$20-byte keys and 2-byte values in Facebook's "USR" pool [Atikoglu et al. 2012]). Thus, a KVS using "getk" shifts the system bottleneck from servers to the network and relies on overprovisioning of the network bandwidth (and thus an unbalanced system) to improve performance. For example, when using "getk" with 8-byte keys and values and as much client-side batching as possible to achieve 160MRPS, Mega-KV requires 20+Gbps[9] of network bandwidth. The Mega-KV experiments overprovision the network bandwidth to be 40Gbps, which can silently absorb the aforementioned 50% extra network traffic caused by "getk" to allow the performance gain but can underutilize the network when "get" is needed by the clients.

[9]Eight-byte keys and values at 160MRPS with getk require 20.48Gbps (16 *Byte* $\times$ 8 $\times$ 160 *Million*) for transmitting all keys and values. Because "clients batch requests and Mega-KV batches responses in an Ethernet frame as much as possible," the packet header overhead is amortized over the large Ethernet frame and thus becomes very small.

As a comparison, MICA's client-assisted hardware request steering improves system performance without any network overhead.

Pure-RDMA-based KVSs [Mitchell et al. 2013; Dragojević et al. 2014] cause overhead on network traffic by offloading KV processing entirely from servers to clients. The tradeoff is that these solutions require extra round trips across the network for GETs, except in limited scenarios when values can be inlined in hash tables. As demonstrated in the HERD design [Kalia et al. 2014], pure-RDMA-based KVSs [Mitchell et al. 2013; Dragojević et al. 2014] exhibit lower performance than a hybrid CPU-RDMA-based KVS [Kalia et al. 2014] despite engaging the server's CPU in the latter solution.

5.7. Energy Efficiency

The power consumption of our KVS platform without client-side batching is 399.2W; power distribution to the two CPUs accounts for 255W, the four NICs consume 62W, and other system components account for the remaining 82W (mostly the 128GB memory and motherboard). At 120.5MRPS, our KVS platform achieves 302 kilo RPS/watt (KRPS/W) energy efficiency. Although client-side batching yields a 39% increase in performance—167.2MRPS for uniform workload with 95% GET—the system power consumption increases by only 4.4% to 416.7W. The reason for the small power increase despite the large performance improvement is that the same processing power shifts from network processing to KV processing. At 167.2MRPS with client-side batching is enabled, our KVS platform achieves energy efficiency of 401KRPS/W.

This unprecedented energy efficiency shows that efficient, balanced use of traditional processors can provide higher performance *as well as higher energy efficiency* than both FPGA- and GPU-based KVS platforms (see Section 5.8 for comparison details). The OS network stack and concurrency control overheads limit the performance and scalability of conventional KVSs. For example, memcached deployments running on a high-end server often underutilize the compute, memory, and network resources, which continue to consume power. Software optimizations can help reap the full potential of the system and ensure that each system resource is used efficiently, allowing a commodity system to provide high performance and energy efficiency.

5.8. Comparing with State-of-the-Art KVSs

This section compares MICA against reported results in other state-of-the-art KVS platforms, as shown in Table VI. Because different KVSs use different assumptions, workloads, use scenarios, and hardware, perfect comparisons are hard to perform. The main purpose of this discussion is not to compare end results but to help paint a more complete picture of the implications of different modern KVS designs. We compare performance, round-trip latency, energy efficiency, and 10GbE line-rate packet size (LPS; a packet size used to saturate the 10GbE link bandwidth).

Performance: In addition to the demonstrated two orders of magnitude better performance than stock memcached and variants on CPU platforms as shown in Section 5.2, MICA achieves the highest throughput among all KVSs including GPU- and FPGA-based KVS platforms, as shown in Table VI. We report MICA results both without and with client-side batching; MemcachedGPU and both FPGA-based designs report only results without client-side batching, and Mega-KV reports only results with client-side batching. The table indicates which setups use client-side batching when comparing MICA and the other KVSs. For GET-intensive workloads, MICA (167MRPS with client-side batching) outperforms Mega-KV (120MRPS with client-side batching), the best GPU-based design, by 39% when servers for both designs return accurate results without relying on clients to do the final key comparison to confirm a potential match. MICA also performs 5× better than FPGA-T, the best FPGA-based design. MICA and the FPGA-based designs are robust to changes in the access patterns and

Table VI. Comparisons Among MICA and Other Major KVSs of Mega-KV [Zhang et al. 2015], MemcachedGPU [Hetherington et al. 2015], FPGA-X [Blott et al. 2013], and FPGA-T [Tanaka and Kozyrakis 2014]. Results Not Reported or Applicable Are Noted by "—". Comparisons on Performance, Round-Trip Latency, and Energy Efficiency Are done with Results Both Without Client-Side Batching (*N-CSB*) and with Client-side Batching (CSB), Separated by "/". For Performance and Energy Efficiency, Higher Is Better. For Round-Trip Latency and 10GbE Line-Rate Packet Size (LPS), Lower Is Better

	Performance (MRPS)		Energy	Round-Trip		10GbE
	†GET-Intensive	‡SET-Intensive	Efficiency (KRPS/W)	Latency (μs)		Line-Rate Packet
KVS Platform	N-CSB/CSB		*N-CSB/CSB*	*N-CSB/CSB*		Size (B)
MICA	120.5/167.2	122.6/165.2	302/401	95^{th} %-tile = 96	/112	88
Mega-KV	—/120[§]	—/—	—/—	95^{th} %-tile = —	/410	∼1500
MemcachedGPU	13/—	0.032/—	62/—	95^{th} %-tile = 1100	/—	96
FPGA-X	13.5/—	12.8/—	106.7/—	3.5–4.5	/—	96
FPGA-T	20/—	18.3/—	173/—	4	/—	∼168

†The GET% of the GET-intensive workload varies from 95% to 100% in different systems.
‡The SET% of the SET-intensive workload varies from 50% to 100% in different systems.
§The Mega-KV results with "get" instead of "getk" (whose performance is 160–166MRPS) are used to ensure fair comparison, because unlike other KVSs, Mega-KV servers with getk cannot return final accurate results to clients, which also requires overprovisioning of network bandwidth for the increased performance.

workloads, as they have equivalent performance on GET- and SET-intensive workloads. The GPU-based designs, however, show lower throughput for SET-intensive workloads than GET-intensive ones. MemcachedGPU shows poor performance (only 0.032MRPS) for SET-intensive workloads on their GPUs, though the authors of MemcachedGPU suggest that further optimizations may lead to potential performance gain [Hetherington et al. 2015]. Although Mega-KV does not report end-to-end performance for SET-intensive workloads, its GPU hash table performance shows that Insert/Delete (SET) operations have ∼33% performance degradation over Search (GET) operations. Since the GPU hash table is a critical stage in the Mega-KV processing pipeline, the performance degradation for SET operations on the GPU hash table implies a potential major performance loss for SET-intensive workloads on Mega-KV.

Energy efficiency: MICA achieves not only the best performance but also the best energy efficiency among all KVSs including GPU- and FPGA-based KVS platforms, as shown in Table VI. In particular, MICA demonstrates a 1.75× full system energy efficiency improvement over FPGA-T, the most energy-efficient FPGA-based KVS (and the second-most energy-efficient KVS overall), and a 4.8× energy efficiency improvement over MemcachedGPU. These results do not use client-side batching. These results further highlight that our commodity system achieves not only the highest performance but also the highest energy efficiency among all systems demonstrated to date.

Round-trip latency: High performance and energy efficiency are only meaningful with satisfactory QoS (i.e., round-trip latency). Because of direct hardware implementation, FPGA-based designs achieve the best round-trip latency of less than 5μs. MICA's round-trip latency is higher, but still much better than that of the GPU-based designs. For example, MICA's 95^{th} percentile round-trip latency is 27.3% and 8.7% of that of Mega-KV and memcachedGPU, respectively.

10GbE line-rate packet size (LPS): Besides the common metrics of performance, energy efficiency, and latency, line-rate packet size (LPS) is an important yet rarely discussed metric for evaluating KVS platforms. As reported in a recent study from Facebook [Atikoglu et al. 2012], small key-value items are a common building block in data center services. Unfortunately, small items are much harder to handle than large items because (1) the small item performance cannot be improved by adding more resources, such as network ports, to the KVS server, and (2) it is harder to amortize

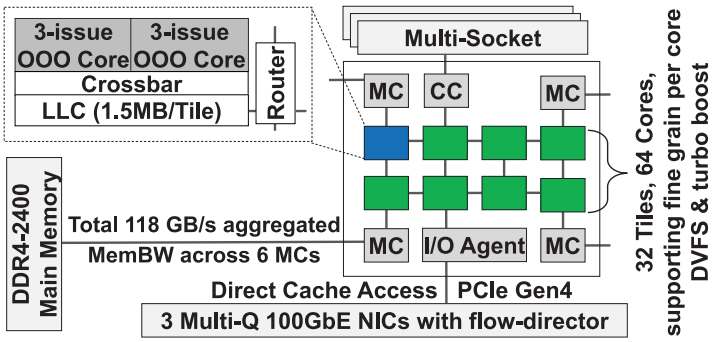

Fig. 14. Proposed platform architecture for high-performance KVS systems.

per-packet and per-request processing overhead for small packets versus large ones. Inflating the packet size by using client-side batching may incur a large increase in latency, as shown in Section 5.3; this increase may make client-side batching less applicable to practical large-scale KVS clusters.

Recent KVS studies [Blott et al. 2013; Lim et al. 2013; Tanaka and Kozyrakis 2014; Hetherington et al. 2015] demonstrate that smaller key-value items (and thus smaller network packet sizes) make it harder for a KVS to achieve line rate on 10GbE links. Table VI compares the 10GbE LPS for the various KVS systems. For sufficiently large packet sizes, performance of the different KVS systems will most likely converge. The differentiation happens with the smaller packet sizes; lower LPS means the system is capable of handling more challenging workloads that consist of smaller items. MICA has the smallest 10GbE LPS (88 bytes) among the evaluated designs, even better than the 10GbE LPS on FPGA-based systems (the best reported result is 96 bytes). Moreover, MICA achieves a 10GbE LPS of 88B even when running at full speed with all twelve 10GbE links.

6. ACHIEVING A BILLION-REQUESTS-PER-SECOND-PER-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 (Section 5). This section focuses on our final grand challenge: designing future KVS platforms to deliver BRPS throughput using a single multisocket server.

6.1. Architecting a Balanced Platform Holistically

As shown in Figure 14 and Table VII, the proposed platform consists of multiple many-core 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 [Li et al. 2009]. Each processor is paired with three multiqueue 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 many-core processors, and we evaluate both dual- and quad-socket servers.

We now explain the reasons behind our main design choices, based on the insights gained from the full-stack system analysis and our simulations. Designing a BRPS-

Table VII. 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 [Li et al. 2009]) are for Normal (N), Low Power (LP), and Turbo Boost (TB). wauDCA Can Use up to 10% of LLC [DDIO 2014]

CPU (w/ wauDCA similar to Intel DDIO [DDIO 2014])	
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	32KB, 8-way, 64B
L2 (LLC) Cache/tile	1.5MB (768KB/core), 16-way, 64B
# 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 [FlowDirector 2014])	
Multiqueue NICs	Three 100GbE, PCIe4.0 x8 per NIC

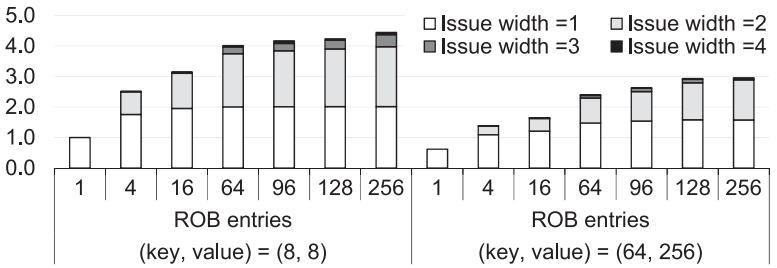

Fig. 15. Relative performance for different item size when varying the ROB size and issue width of cores.

level KVS platform requires the right system balance among compute, memory, and network.

Compute. Figure 3 shows that MICA's IPC is up to 1.9 on the CPU, which indicates that four-issue OOO cores could be overkill. Thus, we perform a sensitivity study for core weight through simulations (see Section 6.2 for simulation infrastructure details). Figure 15 shows the platform's performance in normalized RPS as the reorder buffer (ROB) size (number of entries) and issue width are varied for datasets 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 15, 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 superlinear increase in complexity with larger window sizes and issue width, using a core more powerful than three-issue with 64 ROB entries is not cost effective. Thus, we choose three-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 5.1), 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.

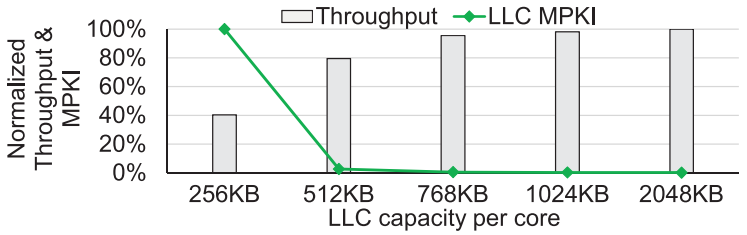

Fig. 16. Throughput and LLC MPKI for different LLC capacities per core. Simulations use `Standard` workload and EREW.

Thus, each 60-core processor can provide enough processing power for 300Gbps bandwidth. We assume that our platform will use emerging 100Gbps Ethernet NICs (similar to Mellanox [2015]). Each 100GbE NIC requires at least 100Gbps of I/O bandwidth—an eight-lane (upcoming) PCIe 4.0 slot will be enough with its 128Gbps bandwidth. On-chip integrated NICs [Li et al. 2011; Novakovic et al. 2014] will be an interesting design choice for improving system total cost of ownership (TCO) and energy efficiency, but we leave it for future exploration.

Memory subsystem and cache hierarchy. 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 (Section 5.3), which is favored by modern memory systems whose latency lags bandwidth significantly [Patterson 2004]. Thus, when designing the main memory subsystem, we provision sufficient memory bandwidth without overarchitecting it for low memory latency. Using the same analysis as in Section 5.3, should our optimized MICA reach 1BRPS on the target four-socket 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 118GB/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.

Our cache hierarchy contains two levels, because our performance analysis (Section 5.1) and simulations reveal that a small private L2 cache in the presence of large L3 does not provide noticeable benefits due to high L2 miss rate. An LLC[10] 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. 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.

Figure 16 shows the platform performance and LLC misses with different LLC capacities, with wauDCA consuming up to 10% [DDIO 2014] of LLC capacity. While the 256KB (per-core) LLC cannot hold all queues and packet buffers from the network, increasing LLC capacity to 512KB accommodates most of them without thrashing against KV processing on CPU, leading to a major performance gain (97%) and cache miss reduction (98%). Increasing LLC capacity further to 768KB fully accommodates

[10]Our performance analysis (Section 5.1) and simulations confirm that a 32KB L1D cache is sufficient. We focus on detailed analysis of LLC in this article because of its high importance and the limited article space.

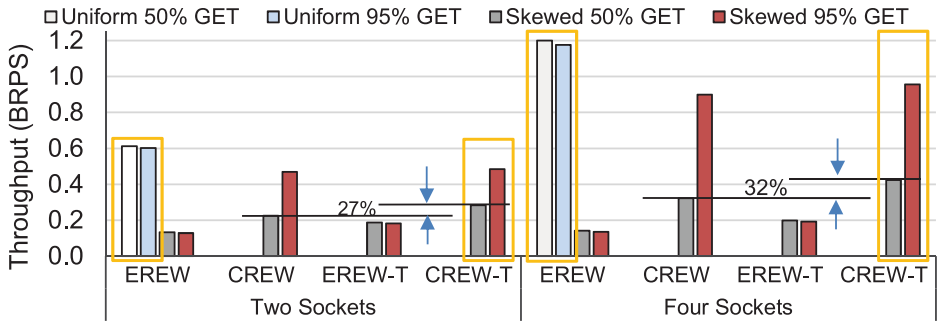

Fig. 17. End-to-end performance of dual- and quad-socket servers. CREW and Turbo Boost (EREW-/CREW-T) are only applicable to, and thus are only shown for, skewed workloads. All 95^{th} percentile latencies are less than 100μs.

network I/O injected directly into the LLC by the NICs and eliminates the interference among the two cores in the same tile, leading to extra performance gain (20%) and LLC miss reduction (82%). Further increasing LLC capacity to 2MB brings diminishing returns with only 4.6% additional gain. Therefore, we adopt the LLC design with 768KB per core (45MB per processor) in our many-core architecture.

Large items demonstrate similar trends, with smaller performance gain and LLC miss reduction when increasing LLC capacity. The reason is that large items rapidly become bottlenecked by network bandwidth. Thus, the faster-degraded network I/O provides more time slack than what is needed by CPUs to fetch extra cache lines because of increased item size for KV processing.

Discussions. Despite a carefully crafted system architecture, our platform remains general purpose in terms of its core architecture (three-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. With proper support, the proposed platform should be able to run standard OSes (e.g., Linux).

6.2. Performance Evaluation

Our simulation infrastructure is based on McSimA+ [Ahn et al. 2013], a many-core simulator that models multithreaded in-order and out-of-order cores, caches, directories, on-chip networks, and memory controllers and channels in detail. We augmented McSimA+ with a multiqueue NIC model and MOESI cache coherence protocol to model wauDCA. We also extended the interconnect model of McSimA+ to simulate intersocket communication. Because the kernel bypassing and memory pinning used in MICA render OS features less important, our simulation results are accurate regardless of the OS used (and thus regardless of McSimA+'s inability to model detailed OS-level activities). To reduce simulation complexity, McSimA+ uses a ghost client to send and receive key-value requests without modeling the execution of the client. However, it is the same from the simulated server's perspective, and the server can apply the same flow control mechanism as if it were talking to a real client.

Figure 17 shows the performance of the target dual- and quad-socket servers. Running on our proposed platform in simulation, 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 ~1.2BRPS in EREW mode with uniform workloads. Skewed workloads pose a

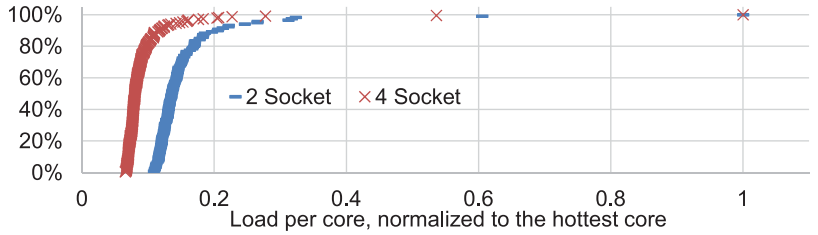

Fig. 18. CDF of the partition load on different cores.

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 [Cooper et al. 2010]), 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 [Lim et al. 2014]. For example, for 24 cores (and partitions), as in our experimental platform (Section 5), the most popular partition is only 97% more frequently requested than the average.

However, in our proposed architecture, the load on the hottest partition is 10.6× (on the 240-core quad-socket server) and 5.8× (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 improve the performance of the hottest cores by ∼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 0.13BRPS (21% of the system peak performance) and 0.14BRPS (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 17).

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 18). More than 90% of the cores are lightly loaded—less than 20% of the hottest cores. This observation leads to an architectural optimization using dynamic frequency/voltage scaling (DVFS) and turbo boost (TB) technologies. We assume that our many-core processor is equipped with recent high-efficiency per-domain/core on-chip voltage regulators [Jevtic et al. 2015]. Based on the supply voltage and frequency pairs shown in Table VII, 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 six most loaded cores to 3.5GHz. Results obtained from DVFS modeling in McPAT [Li et al. 2009] show that this configuration actually reduces total processor power by 16%, which ensures enough thermal headroom for turbo boost of the six hot cores. Our results in Figure 17 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 0.42BRPS and by 27% to 0.28BRPS on the quad- and dual-socket servers, respectively. Although data center KVS workloads are read heavy with a GET ratio higher than 95% on average [Atikoglu et al. 2012], 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.

7. CONCLUSIONS

As an important building block for large-scale Internet services, key-value stores affect both the service quality and the energy efficiency of data-center-based services. Through a cross-stack whole-system characterization, this article evaluates (and improves) the scaling and efficiency of both legacy and cutting-edge key-value implementations on commodity $\times 86$ servers. Our cross-layer system characterization provides important *full-stack* insights (software through hardware) for KVS systems. For example, the evaluation sheds new light on how both software features, such as prefetching, and modern hardware features, such as wauDCA and multiqueue NICs with flow steering, can work synergistically to serve high-performance KVS systems.

Beyond optimizing to achieve the record-setting 120MRPS performance (167MRPS with client-side batching) and 302KRPS/watt (401KRPS/watt with client-side batching) energy efficiency on our commodity dual-socket KVS system, we propose a future many-core-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 codesign for increasingly demanding network-intensive and data-centric applications.

ACKNOWLEDGMENTS

We thank Luke Chang, Patrick Lu, Srinivas Sridharan, Karthikeyan Vaidyanathan, Venkyand Venkatesan, Amir Zinaty, and the anonymous reviewers for their valuable feedback.

REFERENCES

Jung Ho Ahn, Sheng Li, Seongil O, and Norman P. Jouppi. 2013. McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling. In *ISPASS*.

Amazon. 2012. Amazon Elasticache. Retrieved from http://aws.amazon.com/elasticache/.

Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *SIGMETRICS*.

Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: A protected dataplane operating system for high throughput and low latency. In *OSDI*.

Michaela Blott, Kimon Karras, Ling Liu, K Vissers, J Bär, and Z István. 2013. Achieving 10Gbps line-rate key-value stores with FPGAs. In *HotCloud*.

Sai Rahul Chalamalasetti, Kevin Lim, Mitch Wright, Alvin AuYoung, Parthasarathy Ranganathan, and Martin Margala. 2013. An FPGA memcached appliance. In *FPGA*.

Brian Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *SOCC*.

Intel DDIO. 2014. Intel® Data Direct I/O Technology. Retrieved from http://www.intel.com/content/www/us/en/io/direct-data-i-o.html.

Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. 2009. RouteBricks: Exploiting parallelism to scale software routers. In *SOSP*.

Intel DPDK. 2014. Intel Data Plane Development Kit (Intel DPDK). Retrieved from http://www.intel.com/go/dpdk.

Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast remote memory. In *NSDI*.

Facebook. 2014. Introducing mcrouter: A memcached protocol router for scaling memcached deployments. Retrieved from https://code.facebook.com/posts/296442737213493/introducing-mcrouter-a-memcached-protocol-router-for-scaling-memcached-deployments/.

Bin Fan, David G. Andersen, and Michael Kaminsky. 2013. MemC3: Compact and concurrent memcache with dumber caching and smarter hashing. In *NSDI*.

Intel FlowDirector. 2014. Intel® Ethernet Flow Director. Retrieved from http://www.intel.com/content/www/us/en/ethernet-controllers/ethernet-flow-director-video.html.

Anthony Gutierrez, Michael Cieslak, Bharan Giridhar, Ronald G. Dreslinski, Luis Ceze, and Trevor Mudge. 2014. Integrated 3D-stacked server designs for increasing physical density of key-value stores. In *ASPLOS*.

Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2010. PacketShader: A GPU-accelerated software router. In *SIGCOMM*.

Maurice Herlihy, Nir Shavit, and Moran Tzafrir. 2008. Hopscotch hashing. In *Distributed Computing*. Springer, 350–364.

Tayler H. Hetherington, Mike O'Connor, and Tor M. Aamodt. 2015. MemcachedGPU: Scaling-up scale-out key-value stores. In *Proc. SOCC*.

Ram Huggahalli, Ravi Iyer, and Scott Tetrick. 2005. Direct cache access for high bandwidth network I/O. In *ISCA*.

Intel IOAT. 2014. Intel® I/O Acceleration Technology. Retrieved from http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html.

Ruzica Jevtic, Hanh-Phuc Le, Milovan Blagojevic, Stevo Bailey, Krste Asanovic, Elad Alon, and Borivoje Nikolic. 2015. Per-core DVFS with switched-capacitor converters for energy efficiency in manycore processors. *IEEE TVLSI* 23, 4 (2015), 723–730.

Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA efficiently for key-value services. In *SIGCOMM*.

Rishi Kapoor, George Porter, Malveeka Tewari, Geoffrey M. Voelker, and Amin Vahdat. 2012. Chronos: Predictable low latency for data center applications. In *SOCC*.

Maysam Lavasani, Hari Angepat, and Derek Chiou. 2013. An FPGA-based in-line accelerator for memcached. In *HotChips*.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*.

Sheng Li, Hyeontaek Lim, Victor W. Lee, Jung Ho Ahn, Anuj Kalia, Michael Kaminsky, David G. Andersen, O. Seongil, Sukhan Lee, and Pradeep Dubey. 2015. Architecting to achieve a billion requests per second throughput on a single key-value store server platform. In *ISCA*.

Sheng Li, Kevin Lim, Paolo Faraboschi, Jichuan Chang, Parthasarathy Ranganathan, and Norman P. Jouppi. 2011. System-level integrated server architectures for scale-out datacenters. In *MICRO*.

Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. 2014. MICA: A holistic approach to fast in-memory key-value storage. In *NSDI*.

Kevin Lim, David Meisner, Ali G. Saidi, Parthasarathy Ranganathan, and Thomas F. Wenisch. 2013. Thin servers with smart pipes: Designing SoC accelerators for memcached. In *ISCA*.

Linkedin. 2014. How Linkedin uses memcached. Retrieved from http://www.oracle.com/technetwork/server-storage/ts-4696-159286.pdf.

Pejman Lotfi-Kamran, Boris Grot, Michael Ferdman, Stavros Volos, Onur Kocberber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, Sachin Idgunji, Emre Ozer, and Babak Falsafi. 2012. Scale-out processors. In *ISCA*.

Yandong Mao, Eddie Kohler, and Robert Tappan Morris. 2012. Cache craftiness for fast multicore key-value storage. In *EuroSys*.

Mellanox. 2014. Mellanox® OpenFabrics Enterprise Distribution for Linux (MLNX_OFED). Retrieved from http://www.mellanox.com/page/products_dyn?product_family=26.

Mellanox. 2015. Mellanox® 100Gbps Ethernet NIC. Retrieved from http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-4_VPI_Card.pdf.

memcached. 2003. Memcached: A distributed memory object caching system. Retrieved from http://memcached.org/.

Christopher Mitchell, Yifeng Geng, and Jinyang Li. 2013. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. In *USENIX ATC*.

Netflix. 2012. Netflix EVCache. Retrieved from http://techblog.netflix.com/2012/01/ephemeral-volatile-caching-in-cloud.html.

Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. 2013. Scaling memcache at Facebook. In *NSDI*.

Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. 2014. Scale-out NUMA. In *ASPLOS*.

Diego Ongaro, Stephen M. Rumble, Ryan Stutsman, John Ousterhout, and Mendel Rosenblum. 2011. Fast crash recovery in RAMCloud. In *SOSP*.

R. Pagh and F.F. Rodler. 2004. Cuckoo hashing. *J. Algorithms* 51, 2 (May 2004), 122–144.

David A. Patterson. 2004. Latency lags bandwith. *Commun. ACM* 47, 10 (2004), 71–75.

Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and Robert T. Morris. 2012. Improving network connection locality on multicore systems. In *EuroSys*.

Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2014. Arrakis: The operating system is the control plane. In *OSDI*.

Luigi Rizzo. 2012. netmap: A novel framework for fast packet I/O. In *USENIX ATC*.

Shingo Tanaka and Christos Kozyrakis. 2014. High performance hardware-accelerated flash key-value store. In *NVM Workshop*.

Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, and Rebecca L. Stamm. 1996. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *ISCA*.

Twitter. 2012. Twemcache: Twitter Memcached. https://github.com/twitter/twemcache. (2012).

Kai Zhang, Kaibo Wang, Yuan Yuan, Lei Guo, Rubao Lee, and Xiaodong Zhang. 2015. Mega-KV: A case for GPUs to maximize the throughput of in-memory key-value stores. *Proc. VLDB Endow.* 8, 11 (July 2015).

Received December 2015; accepted January 2016