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

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

Abstract

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

We aim at architecting high performance and efficient KVS platforms, and start with a rigorous architectural characterization across system stacks over a collection of representative KVS implementations. Our detailed full-system characterization not only identifies the critical hardware/software ingredients for high-performance KVS systems, but also leads to guided optimizations atop a recent design to achieve a record-setting throughput of 120 million requests per second (MRPS) on a single commodity server. Our implementation delivers 9.2X the performance (RPS) and 2.8X the system energy efficiency (RPS/watt) of the best-published FPGA-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.

ISCA'15, June 13-17, 2015, Portland, OR USA
ACM 978-1-4503-3402-0/15/06.
http://dx.doi.org/10.1145/2749469.2750416

1. Introduction

Distributed in-memory key-value stores such as memcached [7] have become part of the critical infrastructure for large scale Internet-oriented datacenters. They are deployed at scale across server farms inside companies such as Facebook [36], Twitter [8], Amazon [1], and LinkedIn [4,7]. Unfortunately, traditional KVS implementations such as the widely used memcached do not achieve the performance that modern hardware is capable of: They use the operating system's network stack, heavyweight locks for concurrency control, inefficient data structures, and expensive memory management. These impose high overheads for network processing, concurrency control, and key-value processing. As a result, memcached shows poor performance and energy efficiency when running on commodity servers [32].

As a critical layer in the datacenter infrastructure, the performance of key-value stores affects the QoS of web services [36], whose efficiency in turn affects datacenter cost. As a result, architects and system designers have spent significant effort improving the performance and efficiency of KVSs. This has led to two different research efforts, one hardware-focused and one software-focused. The hardware-based efforts, especially FPGA-based designs [14,15,32], improve energy efficiency by more than 10X compared to legacy code on commodity servers. The software-based research [18,19,26,31,34,35,38] 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 [31], 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, it is still unclear: 1) whether the software optimizations can exploit the true potential of modern platforms; 2) what the essential optimization ingredients are and how these ingredients improve performance in isolation and in collaboration; 3) what the implications are for future platform architectures. We believe the answers to these questions will help architects design the next generation of high performance and energy efficient KVS platforms.

We begin with a rigorous and detailed characterization across system stacks, from application to OS and to bare-metal hardware. We evaluate four KVS systems, ranging from the most recent (MICA) to the most widely used (memcached). Our holistic system characterization provides important *full-stack* insights on how these KVSs use modern platforms, from *compute* to *memory* and to *network* subsystems. This paper is the first to reveal the important (yet hidden) synergistic implications of modern platform features (e.g., direct cache access [2,23], multi-queue NICs with flow-steering [3], prefetch, and beyond) to high performance KVS systems. Guided by these insights, we optimize MICA and achieve record-setting performance (120 Million RPS) and energy efficiency (302 kilo RPS/watt) on our commodity system—over 9.2X the performance (RPS) and 2.8X the system energy efficiency (RPS/watt) of the best-published FPGA-based claims [14], respectively. Finally, based on these full-stack insights, we craft a set of design principles for a future manycore-based and throughput-optimized platform architecture, with right *system balance* among compute, memory, and network. We extend the McSimA+ simulator [11] to support the modern hardware features our proposal relies upon 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 comprise 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 [36]. 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—e.g., Facebook operates a memcached KVS cluster serving over a billion requests/second for trillions of items [36]. However, the original memcached, the most widely used KVS system, can achieve only sub-million to a few million requests per second (RPS) on a single IA server [31,32] because of overheads from in-kernel network processing and locking [27,32]. 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 [14,15,28,32] in this direction achieve up to 13.2MRPS [14] with a 10GbE link and more than 10X improvements on energy efficiency compared to commodity servers running stock memcached. There are also non-FPGA-based architecture proposals [20,30,33,37] for accelerating memcached and/or improving performance and efficiency of various datacenter workloads.

On the software side, recent work [18,19,26,31,34,35,38] has optimized the major components of KVSs: network processing, concurrency control, key-value processing, and memory management, either in isolation or 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 [31], which achieves 77 MRPS on a dual-socket server with Intel® Xeon™ E5-2680 processors.[1]

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 continues to increase. For example, Intel Xeon processors today have as many as 18 powerful cores with 45MBs of LLC. These multi-/manycore CPUs provide high aggregate processing power.

Modern NICs, aside from rapid improvements in bandwidth and latency, offer several new features to better work with high-core-count systems: multiple queues, receiver-side scaling (RSS), and flow-steering to reduce the CPU overhead of NIC access [17,41]. 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) [23],[2] implemented as Intel Data Direct I/O Technology (Intel DDIO) [2] in Intel processors, allow both traditional and RDMA-capable NICs to inject packets directly into processor 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

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

[2] This paper always refers to DCA as the wauDCA design [23] (e.g., Intel Data Direct I/O Technology [2]) instead of the simplified Prefetch Hint [23] based implementation [5].

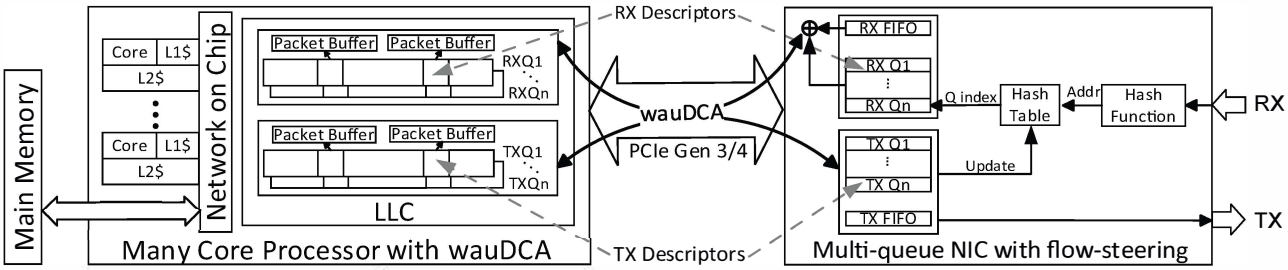

Figure 1: A modern system with write-allocate-write-update-capable Direct Cache Access (wauDCA), e.g., Intel DDIO [2], and a multi-queue NIC with flow-steering, e.g., Intel Ethernet Flow Director [3], to support high performance network intensive applications.

Network stack	Example systems
Kernel	memcached [7], MemC3 [19]
Userspace	Chronos [27], MICA [31]

Concurrency control	Example systems
Mutex	memcached, MemC3
Versioning	Masstree [34], 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 [38]
Circular log	MICA

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

NICs, multi-queue NICs enable efficient parallel transmission from multiple cores by eliminating queue-contention, and parallel reception by providing flow-steering, implemented as Intel Ethernet Flow Director (Intel Ethernet FD) [3] in Intel NICs. With flow-steering enabled NICs, each core is assigned a specific receive queue (RX Q), and the OS or an application requests the NIC to configure its on-chip hash table for flow-steering. When a packet arrives, the NIC first applies a hash function to a portion of the packet header, and uses the result to identify the associated RX Q (and thus the associated core) by looking up the on-chip hash table. After that, the NIC will inject the packet and then the corresponding RX Q descriptor directly into the processor LLC via wauDCA. The core can discover the new packets either by polling or by an interrupt from the NIC. The NIC continues processing new packets. Using wauDCA (e.g., Intel DDIO), network I/O does not always lead to LLC misses: an appropriately structured network application thus has the possibility to be as cache-friendly as non-networked programs do.

With fast network I/O (e.g., 100+ Gbps/node), the OS network stack becomes a major bottleneck, especially for small packets. Userspace network I/O, such as PacketShader I/O [21] and Intel Data Plane Development Kit (DPDK) [24], can utilize the full capacity of high speed networks. By eliminating the overheads of heavy-weight OS network stacks, these packet I/O engines can provide line-rate network I/O for very high speed links (up to a few hundred Gbps), *even for minimum-sized packets* [21, 43]. Furthermore, userspace networking can also be kernel-managed [13, 42] to maximize its benefits.

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

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 [19], Pilaf [35], MICA [31], FaRM-KV [18], and HERD [26], optimize different parts of the KVS system to improve performance. As a complex system demanding hardware and software co-design, it is hard to find a "silver bullet" for KVSs, as the best design always depends on several factors including the underlying hardware. For example, a datacenter with flow-steering-capable networking (e.g., Intel Ethernet FD) has a different subset of essential ingredients of an appropriate KVS design from a datacenter without it. Table 1 shows a taxonomy of the KVS design space in four dimensions: 1) the networking stack; 2) concurrency control; 3) key-value processing; and 4) memory management.

The networking stack refers to the software framework and protocol used to transmit key-value requests and responses between servers and clients over the network. memcached uses OS-provided POSIX socket I/O, while newer systems with higher throughput often use a userspace network stack to avoid kernel overheads and to access advanced NIC features. For example, DPDK and RDMA drivers [10, 24] 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. 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 [39] and hopscotch hashing [22]. Replacement policies specify how to manage the limited memory in the server. memcached maintains a full LRU list for each class of similar-sized items, which causes contention under concurrent access [19]; 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 a custom memory management for various reasons: to reduce the overhead of `malloc()` [7], to allow using huge pages to reduce TLB misses [7, 19], to facilitate enforcing the replacement policy [7, 31], etc. One common scheme is SLAB that defines a set of size classes and maintains a memory pool for each size class to reduce the memory fragmentation. There are also log structure [38]-like schemes including a circular log that optimizes memory access for KV insertions and simplifies garbage collection and item eviction [31].

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 co-design/optimization for KVS is challenging. Not only does a KVS exercises all main system components (compute, memory, and network), the design space of both the system architecture and KVS algorithms and implementation are huge, as described in Section 3. We therefore use a multi-stage 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 1, which is a nearly impossible task. Fortunately, Lim et al. [31] have explored the design space to some extent and demonstrated that their MICA design achieves 77 MRPS 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 [7] and MemC3 [19] (a major yet non-disruptive improvement over memcached) to study the system implications of KVS design choices. Table 2 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, i.e., 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 inter-thread 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 [19], MC3-D still suffers from overhead caused by the code base of the original memcached.

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

Name	KVS codebase	Network stack	Concurrency control	Key-value processing		Memory management
				Indexing	Replacement policy	
Mcd-S	memcached	Kernel (libevent)	Mutex	Chained hash table	Strict LRU	SLAB
Mcd-D	memcached	Userspace (DPDK)	Mutex	Chained hash table	Strict LRU	SLAB
MC3-D	MemC3	Userspace (DPDK)	Mutex+versioning	Cuckoo hash table	CLOCK	SLAB
MICA	MICA-cache	Userspace (DPDK)	None/versioning	Lossy hash index	FIFO/LRU/Approx.LRU	Circular log

Table 2: 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.

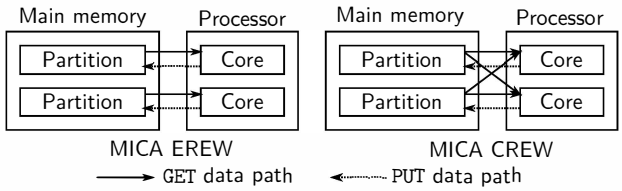

Figure 2: Partitioning/sharding in MICA.

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

Table 3: Workloads used for experiments.

for any concurrency control. In *concurrent read exclusive write* (CREW) mode, MICA relaxes this access constraint by allowing cores to access any partition for `GET` requests (whereas keeping the same constraint for `PUT` requests), which requires MICA to use a versioning-based optimistic locking scheme. 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 to 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 user-space 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 LRU policy.

4.2. Experimental Workloads

We use YCSB for generating key-value items for our workload [16]. While YCSB is originally implemented in Java, we use MICA's high-speed C implementation that can generate up to 140 MRPS using a single machine. The workload has three relevant properties: average item size, skewness, and read-intensiveness. Table 3 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 that rapidly become bottlenecked by network bandwidth [32]. They are also an important workload in datacenter services (e.g., Facebook reports that in one memcached cluster [12], "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 [12,32] and traces [32]. 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 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

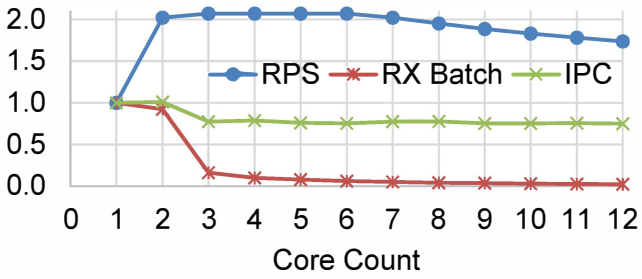

Figure 3: MICA's number-of-cores scalability with 1 10GbE port and one socket. We use STANDARD workload and EREW mode. RX batch size is the average number of packets MICA fetches per I/O operation from the NIC via DPDK. All numbers are normalized to their values at one core, where the actual RPS, RX batch size, and IPC are 11.55 MRPS, 32 packets per I/O operation, and 1.91 (IPC per core), respectively.

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 RPS per KVS Server

We first describe our optimizations guided by detailed full-system characterization, achieving 120 MRPS 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.

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 throughput when using one 10GbE link with an increasing number of CPU cores. While one core of the Intel Xeon processor is not enough to keep up with a 10GbE link, two cores provide close to optimal compute resources, serving 9.76 Gbps out of the 10 Gbps link. Using more cores can squeeze out the remaining 2.4% of the link bandwidth, at the expense of spending more time on network I/O compared to actual key-value (KV) processing. For example, using three cores instead of two reduces the average RX batch size by a factor of 6 (from 32 to 5.29), meaning that cores do less KV processing per I/O operation. Although the IPC does not drop significantly with adding more cores, the newly added cores simply busy-wait on network I/O without doing useful KV processing.

Holding the core to network port ratio as 2:1, we increase the cores and 10GbE ports in lockstep to test the full-system scalability. The maximum throughput achieved in this way is 80 MRPS with 16 cores and 8 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. More total queues also requires the NICs to inject more data into the LLC via Intel DDIO that, however, can only use up to 10% of the LLC capacity [2]. 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 100 MRPS with 20 cores and 10 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.

Figure 4 shows MICA's throughput with our optimizations. MICA achieves 120 MRPS when all 24 cores in both sockets are used. With increasing numbers of cores and ports, L1D and L2 cache misses remain stable, at $\sim$ 1.5% and $\sim$ 32%, respectively. The L1D miss rate stays low because of 1) MICA's intensive software prefetching, which ensures that data is ready when needed; and 2) MICA's careful buffer reuse such as zero-copy RX-TX packet processing. The high L2 cache miss rate is due to packet buffers that do not fit in L1D. The LLC cache miss rate is also low because network packets are placed in LLC directly by the NICs via Intel DDIO and because MICA

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

Table 4: MICA's resource utilization. Cores and ports are evenly distributed across the two sockets. We use `STANDARD` workload with EREW mode.

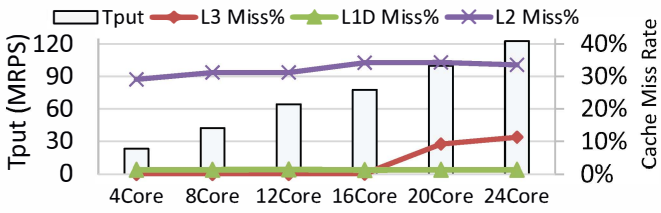

Figure 4: Throughput scalability and cache[3] miss rates of MICA with our optimizations. The port count used is half the core count. We use `STANDARD` workload and EREW mode.

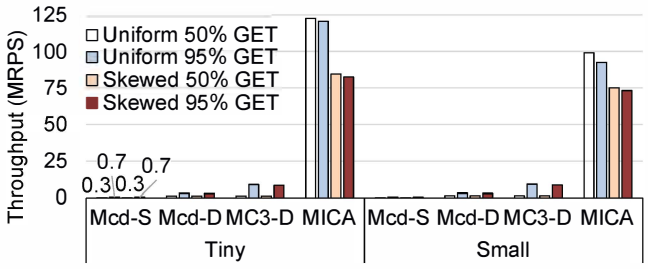

Figure 5: Throughput of the 4 KVS systems with different datasets. For all systems, key-value hit rate is within 98%∼99.6%, and the 95^{th} percentile of latency is less than 100µs. MICA runs EREW mode.

uses intensive software prefetching. While the performance increases linearly, the LLC cache miss rate increases when there are more than 16 active cores (8 per socket). The increased LLC miss rate happens for the same reason that prevents us from increasing beyond 80 MRPS before applying the core-to-port mapping optimization, which indicates the importance of sufficient LLC capacity for future manycore processors for high KVS performance even with the mapping optimization.

Hereafter, we refer to MICA with our optimizations as MICA for simplicity. Table 4 shows the utilization of hardware components on the dual-socket system with two configurations: 2 ports with 4 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-to-1 core-to-port ratio. For example, the memory bandwidth increases from 6.21 GB/s with 2 ports to 34.97 GB/s with 12 ports.

We also performed an architectural characterization of the system implications of simultaneous multithreading (SMT) [44] on our KVS performance, using Intel Hyper-threading Technology, an implementation of 2-way SMT on Intel processors. Our characterization shows that 2-way SMT causes a 24% throughput degradation with the full system setup (24 cores and 12 10GbE ports). This is because the two hardware threads on the same physical core compete on cache hierarchy from L1 to LLC and cause cache thrashing, resulting in a 14%, 27%, and 3.6X increase on L1, L2, and LLC MPKI, respectively. While SMT can improve resource utilization for a wide variety of applications, MICA's relatively simple control structure means that it can incorporate application-specific prefetching and pipelining to achieve the same goal, making single-threaded cores sufficient.

5.2. System Implications of KVS SW Design Choices

Figure 5 shows the measured full-system performance of the four KVS systems (Table 2) with tiny and small datasets (Table 3) and different GET/PUT ratios. MICA performs best regardless of datasets, skew, and GET ratios. For tiny key-value pairs, MICA's throughput reaches 120.5∼116.3 MRPS with the uniform workload and 84.6∼82.5 MRPS for the skewed workload. MICA uses 110∼118 Gbps of network bandwidth under the uniform workload, almost saturating the network stack's sustainable 118 Gbps bandwidth on the server (when processing packet I/O only). Other KVSs achieve 0.3∼9 MRPS for the tiny dataset. Because 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 [32], large and x-large datasets have similar results, with shrinking gaps between MICA and other KVSs as the item size increases.

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

Table 5: KVS configurations to achieve best performance. Mcd-S, Mcd-D, and MC3-D have the same optimal configuration.

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 5. While MICA can leverage all the 24 cores with 12 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 [31], 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 perfor-

[3] Unlike memcached with 20+% L1I$ miss rate due to the complex code path in the Linux kernel and networking stack [32], MICA's L1I$ miss rate is below 0.02% due to the use of userspace networking and kernel bypass.

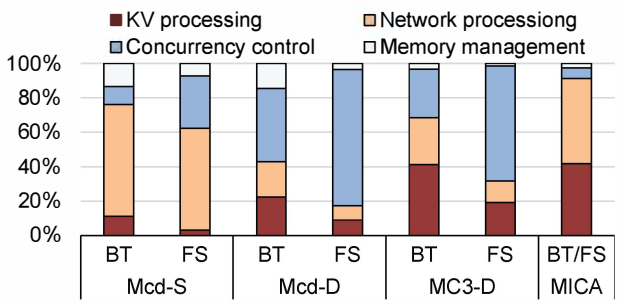

Figure 6: Execution cycle breakdown of different KVS servers. BT (BesT) refers to the configuration that achieves best performance (Table 5) and FS (full system) refers to the configuration with 24 cores and 12 10 GbE ports. Experiment settings: STANDARD workload is used; MICA is in EREW mode.

mance deeply via holistic cross-layer analysis. Figure 6 shows 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 is in line with observations from previous studies on memcached [27]. The pthread mutex-based concurrency control in Mcd-S consumes about 11% of execution time and memory management consumes 13%. As a result, key-value processing work only gets about 10% share of the execution time, leading to the low performance of Mcd-S (0.3 MRPS, 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 0.3 MRPS to 3.1 MRPS, but still less than 3% of MICA's peak throughput. This 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 9 MRPS). However, it still performs costly concurrency control, which consumes $\sim$ 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 8 cores and 2 ports as in Table 5, 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$\sim$3x *less* performance than the BT mode. MICA shows the most balanced execution time break down, with both network and KV processing taking $\sim$45% of execution time respectively. This analysis reveals the underlying reason why replacing one component in the

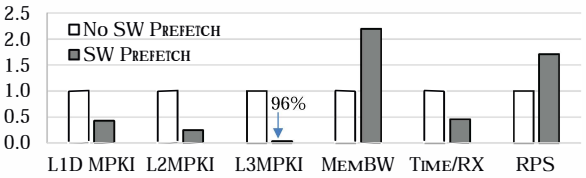

Figure 7: Implications of prefetch on MPKIs of L1/L2/L3 caches, memory bandwidth (MemBW), time spent on each RX batch (time/RX), and throughput (RPS). All numbers are normalized to that without prefetch.

complex KVS software is not enough and a holistic re-design 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 multi-staged software (SW) prefetch on both packets and KV data structures to reduce latency and keep up with high speed network.

Figure 7 shows the system implications of the multi-staged SW prefetch.[4] With SW prefetch, MPKI 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 [32]. 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

[4] MICA uses non-temporal 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.

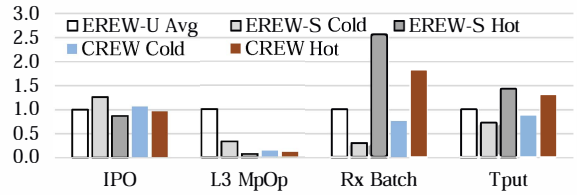

Figure 8: Behavior of skewed workloads. All results are for a single core. Optimized MICA runs in EREW and CREW modes. -U, -S, mean uniform and skewed workloads, respectively. Hot/-cold refer to hot/cold cores based on their load. For Rx Batch and Tput, higher is better. For IPO and MpOp, lower is better.

cache pollution. For each key-value request, MICA needs 1∼2 random DRAM accesses, for a total of 3∼4 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 ∼30 GB/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 [40]. 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 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 120 MRPS.

With skewed workloads, the EREW throughput is ∼84 MRPS. The per-core throughput of cold cores decreases by 28% to 3.58 MRPS on average. The hot cores' throughput, however, increases by 43% to 7.1 MRPS, 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.

[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.

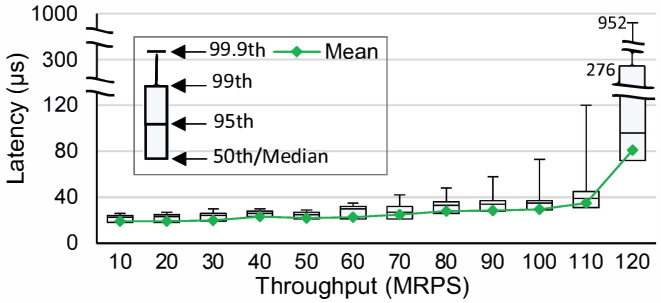

Figure 9: Round trip latency (RTT) (including mean, 50^{th}, 95^{th}, 99^{th}, and 99.9^{th} percentile) for different throughputs. `STANDARD` workload and MICA's EREW mode are used. Mean is always larger than median, because of the tailing effect. Experiments repeat multiple times to eliminate run-to-run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

While EREW tolerates skewed workloads well, 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% (4.3 MRPS) and 129% (6.5 MRPS) throughput/core for cold and hot cores respectively, compared to the uniform EREW mode. This brings the overall system performance back to 108 MRPS, 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) vs. Throughput

High throughput is only beneficial if latency SLAs (service level agreement) 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-vs-throughput details. As throughput changes from 10M∼120M RPS, latency changes gracefully (e.g., mean: 19∼81µs; 95^{th}: 22∼96µs). Our optimized MICA achieves high throughput with robust SLA guarantees. Figure 5 shows that with the same 95^{th} percentile latency (less than 100µs), MICA (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, ∼11X better than the 95^{th} percentile latency of 1135µs reported by Facebook [36].

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 ∼72X better than the 99^{th} percentile latency of 20*ms* reported by Netflix [9]. Moreover, a small sacrifice of throughput (8.3%, for 120MRPS to 110MRPS) improves 99^{th} and 99.9^{th} percentile tail-end latencies to 45µs and 120µs, respectively.

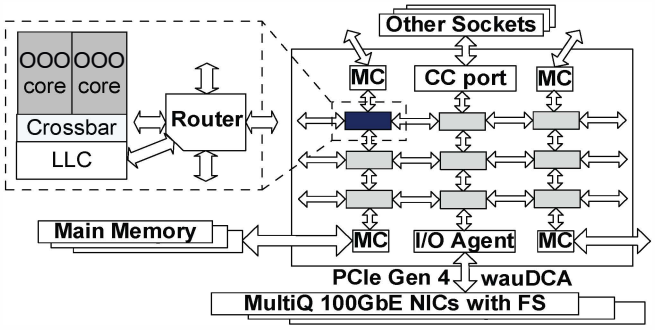

Figure 10: Proposed platform architecture for high performance KVS systems.

5.5. Energy Efficiency

With the energy characterization setup, we measure the power consumption of our KVS platform as 399.2W, with power distribution to the two CPUs and four NICs being 255W and 62W, respectively. The remaining 82W is consumed by the remaining system components, mostly the 128GB memory and motherboard. At 120.5MRPS, our KVS platform achieves 302 kilo RPS/watt (KRPS/W; higher is better) energy efficiency. The best published FPGA-based memcached system so far is from Xilinx [14], with performance of 13.02 MRPS and energy efficiency of 106.7 KRPS/watt (254.8 KRPS/watt when the FPGA host's power is excluded) for minimum-sized packets. Our CPU-based optimized MICA server achieves not only 9.2X the performance (RPS) but also 2.8X (1.18X even when the FPGA host's power is excluded) the system energy efficiency (RPS/watt) of the best-published results for FPGA-based memcached implementations.

This shows that, when fully exposing the system capability, high-performance processors can provide higher performance *as well as higher energy efficiency*. The OS network stack and concurrency control overheads limit the performance and scalability of memcached. When memcached is executed on a high-end server, the compute, memory, and network resources are severely underutilized, but continue to consume power. Software optimizations can help reap the full potential of the system and ensure that each system resource is well utilized, making our commodity system not only a high performance platform but also an energy efficient one for KVSs.

6. Achieving a Billion RPS 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 a billion RPS (BRPS) throughput using a single multi-socket server.

6.1. Architecting a Balanced Platform Holistically

As shown in Figure 10 and Table 6, the proposed platform consists of multiple manycore processors. Each processor

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

Table 6: 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 [29]) are for normal (N), low power (LP), and turbo boost (TB). wauDCA can use up to 10% of LLC [2].

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 [29]. Each processor is paired with three multi-queue 100GbE NICs with flow-steering. The NICs communicate with the processor through PCIe 4.0 and inject packets directly to the LLC via wauDCA. The target server contains two or four such manycore processors, 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-level (billion requests per second) 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 4-issue OOO cores could be an overkill. Thus, we perform a sensitivity study for core weight through simulations (see Section 6.2 for simulation infrastructure details). Figure 11 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 11, 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 per-

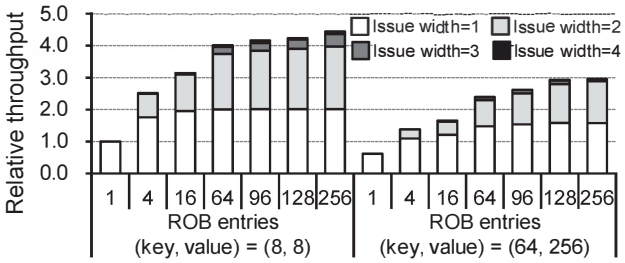

Figure 11: Relative performance for different item size when varying the ROB size and issue width of cores.

formance by only 5%. Considering the super-linear increase in complexity with larger window sizes and issue width, using a core more powerful than 3-issue with 64 ROB entries is not cost effective. Thus, we choose 3-issue OOO cores with 64 ROB entries in the target system.

Network and I/O subsystem: MICA (or any KVS) is a network application. Because our optimized MICA achieves near-perfect scaling (Section 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. 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 [6]). Each 100GbE NIC requires at least 100Gbps of I/O bandwidth—an 8 lane (upcoming) PCIe 4.0 slot will be enough with its 128Gbps bandwidth. On-chip integrated NICs [30, 37] 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 [40]. Thus, when designing the main memory subsystem, we provision sufficient memory bandwidth without over-architecting it for low memory latency. Using the same analysis as in Section 5.3, should our optimized MICA reach 1 BRPS on the target 4-sockets platform, each socket will generate at least $\frac{1}{4} \cdot 4$ billion cache line requests per second from DRAM, for 64GB/s of DRAM bandwidth. We deploy six memory controllers with single-channel DDR4-2400 for a total of 118 GB/s aggregated memory bandwidth to ensure enough headroom for the bandwidth overhead because of MICA's software prefetching and the random traffic in key-value processing.

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[6]

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

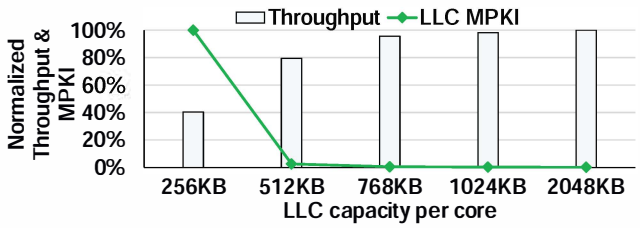

Figure 12: Throughput and LLC MPKI for different LLC capacity per core. Simulations use `Standard` workload and EREW.

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 12 shows the platform performance and LLC misses with different LLC capacity, with wauDCA consuming up to 10% [2] 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 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 manycore 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 (3-issue with 64-entry ROB is midway in the design spectrum of modern OOO cores), its processor architecture (many cores with high speed I/O), and its system architecture (upcoming commodity memory and network subsystem). This generality should allow our proposed platform to perform well for general workloads. With proper support, the proposed platform should be able to run standard OSes (e.g., Linux).

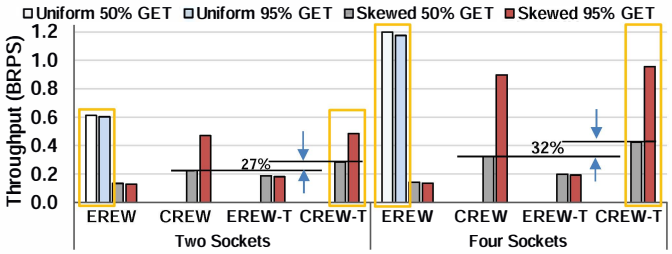

Figure 13: End-to-end performance of dual- and quad-socket servers. CREW and Turbo Boost (EREW-/CREW-T) are only applicable to, and thus are only shown for, skewed workloads. All 95^{th} percentile latencies are less than 100µs.

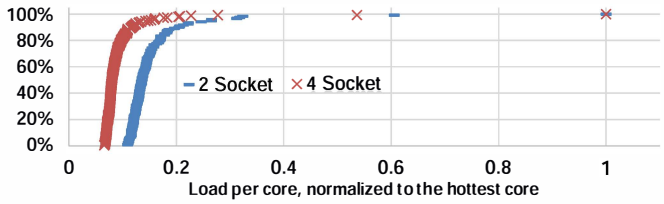

Figure 14: CDF of the partition load on different cores.

6.2. Performance Evaluation

Our simulation infrastructure is based on McSimA+ [11], a manycore 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 multi-queue NIC model and MOESI cache coherence protocol to model wauDCA. We also extended the interconnect model of McSimA+ to simulate inter-socket 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 was talking to a real client.

Figure 13 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 $\sim$ 1.2 billion RPS (BRPS) in EREW mode with uniform workloads. Skewed workloads pose a harder problem on the target platform because of its large number of cores—increasing the number of cores leads to more partitions, which causes a larger load imbalance. In a Zipf-distributed population of size 192×2^{20} (192 million) with skewness 0.99 (as used by YCSB [16]), 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 [31]. 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 hottest partition is 10.6X (on the 240-core quad-socket server) and 5.8X (on the 120-core dual-socket server) of the average load per core, respectively. Although the increased data locality and decreased I/O processing overhead improves the performance of the hottest cores by $\sim$ 50% based on our simulations, it is not enough to bridge the gap between hot and cold partitions/cores. Thus, the hot cores become a serious bottleneck and cause a drastic performance degradation for skewed workloads: The performance on dual- and quad-socket machines is 0.13 BRPS (21% of the system peak performance) and 0.14 BRPS (11% of peak), respectively. Using the CREW mode can help GET-intensive skewed workloads, since in CREW mode all GET requests are sent to all cores to share the load (writes are still sent to only one core). However, for PUT-intensive skewed workloads (Skewed, 50% GET), there is still a large gap between the achieved performance and the peak performance (Figure 13).

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 14). More than 90% of the cores are lightly loaded—less than 20% of the hottest core. This observation leads to an architectural optimization using dynamic frequency/voltage scaling (DVFS) and turbo boost (TB) technologies. We assume that our manycore processor is equipped with recent high efficiency per-domain/core on-chip voltage regulators [25]. Based on the supply voltage and frequency pairs shown in Table 6, we reduce the frequency (and voltage) on the 20 most lightly loaded cores (their load is less than 12% of the load on the hottest core) from 2.5GHz to 1.5GHz and increase the frequency of the 6 most loaded cores to 3.5GHz. Results obtained from DVFS modeling in McPAT [29] show that this configuration actually reduces total processor power by 16%, which ensures enough thermal headroom for turbo boost of the 6 hot cores. Our results in Figure 13 show that with *CREW-T*, the combination of fine-grained DVFS/TB and MICA's CREW mode, the system throughput for the write-intensive skewed workload (Skewed, 50% GET) improves by 32% to 0.42 BRPS and by 27% to 0.28 BRPS on the quad- and dual-socket servers, respectively. Although datacenter KVS workloads are read-heavy with GET ratio higher than 95% on average [12], 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 energy efficiency of datacenter-based services. Through a cross-stack whole system characterization, this paper evaluates (and improves) the scaling and efficiency of both legacy and cutting-edge key-value implementations on commodity x86 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 multi-queue NICs with flow-steering, can work synergistically to serve high performance KVS systems.

Beyond optimizing to achieve the record-setting 120 MRPS performance and 302 KRPS/watt energy efficiency on our commodity dual-socket KVS system, this paper sets forth principles for future throughput-intensive architectural support for high performance KVS platforms. Through detailed simulations, we show that these design principles could enable a billion RPS performance on a single four-socket key-value store server platform. These results highlight the impressive possibilities available through careful full-stack hardware/software co-design for increasingly demanding network-intensive and data-centric applications.

Acknowledgments

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

References

[1] *Amazon Elasticache*, http://aws.amazon.com/elasticache/.
[2] *Intel® Data Direct I/O Technology*, http://www.intel.com/content/www/us/en/io/direct-data-i-o.html.
[3] *Intel® Ethernet Flow Director*, http://www.intel.com/content/www/us/en/ethernet-controllers/ethernet-flow-director-video.html.
[4] *How Linkedin uses memcached*, http://www.oracle.com/technetwork/server-storage/ts-4696-159286.pdf.
[5] *Intel® I/O Acceleration Technology*, http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html.
[6] *Mellanox® 100Gbps Ethernet NIC*, http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-4_VPI_Card.pdf.
[7] *Memcached: A distributed memory object caching system*, http://memcached.org/.
[8] *Memcached SPOF Mystery*, https://blog.twitter.com/2010/memcached-spof-mystery.
[9] *Netflix EVCache*, http://techblog.netflix.com/2012/01/ephemeral-volatile-caching-in-cloud.html.
[10] *Mellanox® OpenFabrics Enterprise Distribution for Linux (MLNX_OFED)*, http://www.mellanox.com/page/products_dyn?product_family=26.
[11] J. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *ISPASS*, 2013.
[12] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *SIGMETRICS*, 2012.
[13] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion, "IX: A protected dataplane operating system for high throughput and low latency," in *OSDI*, 2014.
[14] M. Blott, K. Karras, L. Liu, K. Vissers, J. Bär, and Z. István, "Achieving 10Gbps line-rate key-value stores with FPGAs," in *HotCloud*, 2013.
[15] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala, "An FPGA Memcached appliance," in *FPGA*, 2013.
[16] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SOCC*, 2010.
[17] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting parallelism to scale software routers," in *SOSP*, 2009.
[18] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *NSDI*, 2014.
[19] B. Fan, D. G. Andersen, and M. Kaminsky, "MemC3: Compact and concurrent memcache with dumber caching and smarter hashing," in *NSDI*, 2013.
[20] A. Gutierrez, M. Cieslak, B. Giridhar, R. G. Dreslinski, L. Ceze, and T. Mudge, "Integrated 3D-stacked server designs for increasing physical density of key-value stores," in *ASPLOS*, 2014.
[21] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," in *SIGCOMM*, 2010.
[22] M. Herlihy, N. Shavit, and M. Tzafrir, "Hopscotch hashing," in *Distributed Computing*. Springer, 2008, pp. 350–364.
[23] R. Huggahalli, R. Iyer, and S. Tetrick, "Direct cache access for high bandwidth network I/O," in *ISCA*, 2005.
[24] Intel, "Intel Data Plane Development Kit (Intel DPDK)," http://www.intel.com/go/dpdk, 2014.
[25] R. Jevtic, H. Le, M. Blagojevic, S. Bailey, K. Asanovic, E. Alon, and B. Nikolic, "Per-core DVFS with switched-capacitor converters for energy efficiency in manycore processors," *IEEE TVLSI*, vol. 23, no. 4, pp. 723–730, 2015.
[26] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *SIGCOMM*, 2014.
[27] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable low latency for data center applications," in *SOCC*, 2012.
[28] M. Lavasani, H. Angepat, and D. Chiou, "An FPGA-based in-line accelerator for Memcached," in *HotChips*, 2013.
[29] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.
[30] S. Li, K. Lim, P. Faraboschi, J. Chang, P. Ranganathan, and N. P. Jouppi, "System-level integrated server architectures for scale-out datacenters," in *MICRO*, 2011.
[31] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A holistic approach to fast in-memory key-value storage," in *NSDI*, 2014.
[32] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin Servers with Smart Pipes: Designing SoC accelerators for Memcached," in *ISCA*, 2013.
[33] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *ISCA*, 2012.
[34] Y. Mao, E. Kohler, and R. T. Morris, "Cache craftiness for fast multicore key-value storage," in *EuroSys*, 2012.
[35] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store," in *USENIX ATC*, 2013.
[36] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling Memcache at Facebook," in *NSDI*, 2013.
[37] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "Scale-out NUMA," in *ASPLOS*, 2014.
[38] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in RAMCloud," in *SOSP*, 2011.
[39] R. Pagh and F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, May 2004.
[40] D. A. Patterson, "Latency lags bandwith," *Commun. ACM*, vol. 47, no. 10, pp. 71–75, 2004.
[41] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris, "Improving network connection locality on multicore systems," in *EuroSys*, 2012.
[42] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe, "Arrakis: The operating system is the control plane," in *OSDI*, 2014.
[43] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *USENIX ATC*, 2012.
[44] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," in *ISCA*, 1996.