SCOR-KV: SIMD-Aware Client-Centric and Optimistic RDMA-based Key-Value Store for Emerging CPU Architectures*

Dipti Shankar, Xiaoyi Lu, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University
{*shankard, luxi, panda*}*@cse.ohio-state.edu*

***Abstract*—Modern distributed key-value store-based applications rely on bulk-read operations like 'Multi-Get' (MGet) to accelerate their data serving phase. While state-of-the-art database systems employ SIMD-based techniques to optimize data-parallel operations on their in-memory structures, such as hash-tables, they have not been adapted into high-performance RDMA-accelerated key-value (KV) stores. In this paper, we present a holistic approach to designing high-performance SIMD-aware KV stores for emerging multi-core CPU architectures. Towards this, we first perform an in-depth study of the opportunities and challenges involved in leveraging AVX-512 vectorization-based parallel hash table designs with a state-of-the-art high-performance key-value store like RDMA-Memcached. Based on this, we propose a SIMD-Aware Client-Centric and Optimistic RDMA-based Key-Value Store, SCOR-KV, that optimally exploits 'RDMA+SIMD' to accelerate read-heavy MGet operations. SCOR-KV presents an SIMD-conscious KV store friendly hash table layout, that leverages the vertically vectorized N-way cuckoo hash table design with optimistic KV pair lookup schemes. To complement this, we propose RDMA-optimized SIMD-aware MGet communication protocols that offload the server-side pre-/post-processing overheads to the client, while enabling optimal end-to-end performance. Our performance evaluations over the latest Intel Skylake CPUs and IB EDR interconnects show that our proposed SCOR-KV can achieve up to 3.7-8.6x improvement in server-side Get throughput. Through our SIMD-aware RDMA schemes, SCOR-KV can also improve Multi-Get latencies for read-heavy YCSB workloads by about 2.2x, as compared to the RDMA-Memcached design running over the state-of-the-art CPU-optimized MemC3 hash table design.**

***Keywords*-CPU-SIMD, Key-Value Store, RDMA, AVX-512.**

I. Introduction

Distributed in-memory key-value stores (KV stores) play a vital role in accelerating data-intensive Big Data workloads in multi-tiered data centre architectures and HPC clusters. While they have been traditionally leveraged to accelerate online data serving and caching services [6], [13], they are also being widely used in offline data analytic scenarios in both data centre and HPC environments [23], [26]. Many studies [7], [27] have shown that the performance of KV store-based applications is dominated by reads, i.e., GETs. For instance, based on the real-workload traces from Facebook [3], we see that a single web page request from a user can generate up to 521 distinct key-value (KV) pair items that need to be fetched from the server cluster. High-performance KV store applications [16], [27] attempt to minimize the number of network round trips necessary to fetch all the KV pairs corresponding to the user request by coalescing (or batching) multiple read requests into a single request; to maximize the number of items that can be fetched concurrently. This common data access scenario, referred to as a 'Multi-Get' (MGet), involves coalesced read operations with per-server batches vary between 24 – 96 keys per request [16]. Thus, there is a significant **need for fast and scalable 'Multi-Get' support** for read-mostly KV store workloads.

With the emergence of modern CPU architectures (e.g., Intel Skylake, Intel Cascade Lake) that support vector registers which can fit an entire cache-line (512-bits vectors) [5], there have been several studies directed towards exploiting 'Single Instruction Multiple Data' (SIMD) for accelerating data-intensive operators like scan, sort, join, and bloom filters [8], [12], [19]. SIMD vector instructions have also been leveraged to accelerate lookups over high-performance hash tables [19]–[21] (i.e., CPU-vectorized hash table probing). For instance, Polychroniou et al. [19] propose a data-parallel probing approach with 512-bit vectors (AVX-512), that can enable us to lookup 16 keys with one lookup operation, when using a hash table with 32-bit keys and payloads. These works make it evident that there is **potential for leveraging CPU-SIMD for scaling read-intensive hash table workloads**.

A. Motivation

On the other hand, high-performance 'Remote Direct Memory Access' (RDMA)-based KV stores [9], [10], [14], [25], that are optimized for modern HPC clusters and data centers, use a hash table in their backend to store and index KV pairs in a fast and efficient manner. Together with the above-mentioned 'MGet' support requirement from the applications, these KV store servers can benefit from the ability to search (i.e., 'lookup') multiple KV pairs concurrently using the SIMD-aware CPU-vectorized hash table designs proposed in [19]. However, we find that:

Observation①: The existing RDMA-based KV stores rely on non-SIMD CPU-centric backend designs like tra-

This research is supported in part by NSF grants #CCF-1822987, #CNS-1513120, #ACI-1450440, #CCF-1565414, and NSF ACI1664137.

2640-0316/19/$31.00 ©2019 IEEE
DOI 10.1109/HiPC.2019.00040

ditional chaining hash tables [13] or CPU-optimized bucketized cuckoo hash tables like MemC3 [7]. They have not been adapted to leverage the potential of CPU-SIMD on current and emerging multi-core architectures.

Observation②: The existing state-of-the-art RDMA-based KV store designs mostly focus on improving the scalability and end-to-end latencies for point-to-point queries like `Set(K,V)` and `Get(K)`; i.e., they are not optimized for bulk read operations like `MGet(k1, k2,..,kN)`.

This lack of SIMD-aware backend designs that support read-intensive workloads over high-performance RDMA-enhanced KV stores motivate us to explore the following research challenges on modern multi-core CPU architectures:

Challenge①: What is the potential for leveraging SIMD-aware hash tables for accelerating KV store servers?

Challenge②: Can we easily exploit the performance benefits of CPU-SIMD by naively replacing the hash table backend in the existing state-of-the-art RDMA-based KV stores? If not, what are the performance bottlenecks?

Challenge③: How can we eliminate these bottlenecks to optimally exploit both RDMA and CPU-SIMD in high-performance KV stores, towards supporting read-intensive workloads like 'Multi-Get'?

B. Contribution

To address the above research challenges, we first analyze opportunities for exploiting CPU-SIMD, by performing an in-depth analysis of state-of-the-art SIMD-aware hash table designs [19], [20] with AVX-based vectorization (AVX2 / AVX-512) support. Secondly, to identify the challenges in leveraging these CPU-SIMD designs for accelerating KV store servers, we integrate them with a state-of-the-art CPU-centric KV store like RDMA-Memcached [17], and contrast its performance with that of the non-SIMD cache-optimized designs like MemC3 [7]. Through detailed performance studies, we find that naively replacing the CPU-optimized non-SIMD hash table designs with CPU vectorization-based accelerated designs does not enable us to leverage the optimal performance offered by the SIMD designs.

Based on the analysis with the state-of-the-art, in this paper, we present a holistic approach to designing SIMD-aware in-memory KV store, called SCOR-KV. SCOR-KV presents an SIMD-aware Client-Centric and Optimistic RDMA-aware design for Key-Value Stores, that co-designs the end-to-end KV store 'Multi-Get' pipeline, by proposing:

①: a KV-friendly SIMD-aware hybrid and partitioned hash table design for the server backend. It employs an 'optimistic partial key' lookup scheme that facilitates the KV store to reap the benefits of employing AVX-based vector instructions to lookup multiple KV pairs in parallel.

②: an RDMA-optimized 'optimistic lookup'-aware response processing engine at the server, and a 'Client-centric' SIMD-aware request offload engine that employs zero-copy and optimistic MGet request protocols.

Our performance evaluations over the latest Intel Skylake CPUs and InfiniBand EDR (IB EDR) interconnects, show that SCOR-KV can achieve about 3.7-8.6x improvement in the server-side Get throughput. With our proposed SIMD-aware RDMA-optimized Multi-Get schemes, SCOR-KV can improve the 'Multi-Get' latency for the read-heavy Yahoo! Cloud Serving Benchmark (YCSB) [4] workload by about 2.2x, as compared to RDMA-Memcached [17] running with: (a) the state-of-the-art CPU-optimized MemC3 hash table, and, (b) a naively integrated SIMD-based hash table, in its backend. The rest of the paper is organized as follows. Section II presents the necessary background and Section III presents our motivational analysis. We discuss the SCOR-KV design in Section IV and present experimental evaluations in Section V. Section VI discusses the related work. We conclude in Section VII with future work.

II. Background & Motivation

In this section, we provide an overview of the state-of-the-art non-SIMD and SIMD-aware hash table (HT) designs.

A. CPU-Optimized Key-Value Store Designs

State-of-the-art CPU-optimized HT designs based on Memcached [13], such as MemC3 [7], have been proposed to cater to the 'read-heavy' characteristics of popular online KV store workloads [27]. MemC3 introduces: (a) a CLOCK-based LRU-approximating eviction algorithm to maintain cache freshness while eliminating inter-thread synchronization, and, (b) a cache-optimized optimistic concurrent cuckoo hashing (that supports multiple concurrent readers and a single writer) via atomic key-version counters.

As shown in Figure 1, MemC3 employs a bucketized cuckoo HT (BCHT) with four slots (i.e., places for key/payload) per hash bucket with 2-way cuckoo hashing. Each slot contains a 1-byte key signature and a KV pair location pointe, and every key is associated with one key version counter to enable lock-free read/write for concurrency. Since

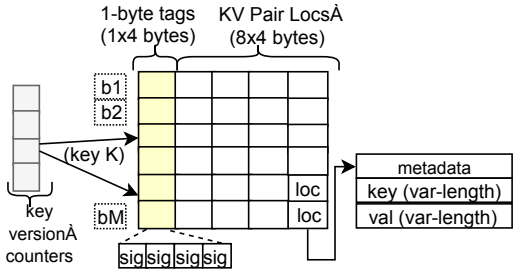


Figure 1: MemC3 Hash Table Design

each bucket fits into a single cache-line (40 B), it enables high load factor of about 90-95%[1] and cache-optimized lookups; unlike Memcached's default cache-unaware chaining HT [9], [10], [13].

[1]load factor of a HT (LF) = (number of KV pairs stored / number of hash buckets); determines max. achievable hit rate for HT lookups.

B. SIMD-Aware Hash Table Lookups

Firstly, using SIMD instructions for HT operations has been proposed as a way to build bucketized hash tables (e.g., BCHT with 'M' slots-per-bucket). Rather than comparing the input key against each slot in the designated hash bucket individually, we can compare all 'M' keys in the hash bucket with the input key using a single SIMD vector comparison. This approach, referred to as *horizontal vectorization* [20], [21], is however not always optimal, as: (a) we may expect to search fewer than 'M' buckets on average, and, (b) it still requires probing the 'n' input keys sequentially.

1) Vertical Vectorization for Data Parallel HT Lookups: To enable true SIMD parallelism for lookups, we need a single HT probe operation to iterate over 'w' input keys in parallel (SIMD width 'w ' >1). The fundamental principle is to process a different key per SIMD lane. Since it exploits SIMD to process multiple keys in parallel and returns a vector of payloads corresponding to the matching keys (for a KV store, 'payload' refers to KV object pointer), this approach is referred to as *vertical vectorization*. A vertical vectorization-based HT lookup algorithm for an N-way cuckoo HT (BCHT with 'M'=1 and N hash functions) is presented in [19]; a single iteration of which involves the following steps (Figure 2 illustrates the same for 'w'=4):

① We load all SIMD lanes with input keys (ki) into a SIMD register (K) with SIMD load operation.

② For each probe key (ki) in SIMD register 'K', we compute-and-store the corresponding hash buckets (hi) into SIMD register 'H', based on hash function h1.

③ Using the hash bucket computed in 'H', we perform a 'Gather' operation load buckets from the cuckoo HT into SIMD register 'KH'.

④ We perform an SIMD compare to match the original probe keys (SIMD register K) with the keys we retrieved from the HT (SIMD register 'KH') to obtain a bitmask for potential key matches (green boxes in the mask indicate matches and red boxes indicate misses). Steps (1), (2), (3), and (4) are repeated up to 'N-1' times with hash functions (h2,..,hN) for keys indicated as missing in the bitmask.

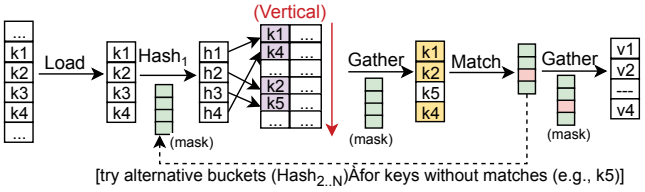


Figure 2: Vectorized operations on a 'N-way' Cuckoo Hash Table (Vertical Vectorization); Illustration of one iteration with probing 4 keys `(k1,k2,k3,k4)` in parallel

Now, if we consider the latest Intel CPUs that enable 512-bit vectors (AVX-512) [5], for an 'N'-way cuckoo HT with 32-bit keys and payloads, we can lookup 16 key in parallel using one probe operation. Thus, we can lookup 'n' keys in a maximum of `N*n/16` iterations, as compared to non-SIMD designs that potentially perform `N*n` total iterations.

2) Stand-Alone HT Performance with Vertical Vectorization: If we consider the latest Intel CPUs that enable 512-bit vectors (AVX-512) [5], for an 'N'-way cuckoo HT with 32-bit keys and payloads, we can lookup 16 key in parallel using one probe operation. Thus, we can lookup 'n' keys in a maximum of `N*n/16` iterations, as compared to non-SIMD designs that may potentially perform `N*n` iterations in total. To evaluate this, we study the stand-alone performance of vertically vectorized 3-way cuckoo hashing with AVX-512, i.e., 'Cuckoo-Ver (AVX-512)', based on the algorithm presented in [19] (detailed in Section II-B1). Since 3-way cuckoo HT enables a load factor close to 90%, we contrast it with the performance of the state-of-the-art CPU-optimized non-SIMD MemC3 [7], i.e., 'MemC3 (Scalar)'.

We mimic two key access patterns: (a) a uniformly random pattern (Uniform), and, (b) a skewed access pattern (Skewed) based on Facebook's Memcached workload generator [3], [15]. We use an input 32-bit column of 1G keys with about 90% selectivity (selectivity = % of the input that is likely to find a match in the HT), and the output is a 32-bit column with matching payloads, over a table with a load factor of 90%. We run this test on a shared HT across all 28 cores of a dual 14-core Intel Skylake node (see Section V).

Summary & Observations: Figure 3 presents the HT probing throughput for the two designs, for varying HT sizes. From this figure, we can observe that for HT sizes that can fit into L2 cache (512 KB), 'Cuckoo-Ver' outperforms MemC3 by about 2.7x – 6.6x. When HT size exceeds L2 cache, we observe that, while 'Cuckoo-Ver' performance drops by 3x–5x, it still maintains a gain of 1.63x–2.6x over MemC3 HT. From the above experiments, we can observe that, while the performance of the vertical-vectorized SIMD-aware HT is bound by memory, it can maintain a consistent improvement over cache-optimized non-SIMD HT designs.

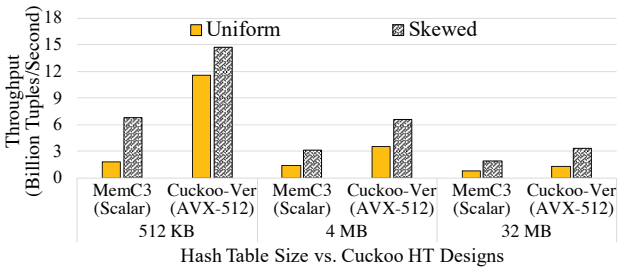


Figure 3: Stand-Alone HT Probing Performance on the 28-core Intel Skylake CPU, over a 3-way Cuckoo HT vs. non-SIMD CPU-optimized MemC3 HT with 32-bit key/payload

With this as the motivation, we present an in-depth analysis of integrating the above SIMD-aware HT into an RDMA-accelerated KV store.

III. Integrating CPU-SIMD into RDMA KV Store

To identify the potential and corresponding challenges in leveraging CPU-SIMD for accelerating KV stores, we integrate the vertically vectorized HT discussed in Section II-B with the state-of-the-art RDMA-Memcached [17] design.

A. Extending RDMA KV Store with SIMD HT

We focus on workloads of 'MGet' operations, i.e., `MGet(K1, K2,..,Kn)`, which batches together multiple KV pairs requests directed to a particular server in KV store cluster. For fair comparison, we replace the traditional chaining HT in RDMA-Memcached [9] with the CPU-optimized non-SIMD MemC3 HT design [7]. As shown in Figure 4, the client-to-server pipeline for an MGet operation can be broken down into three basic phases:

(1) Request Phase: In this phase, each key in MGet(K1,..,Kn) is mapped to a specific Memcached server using consistent hashing, and the requests are batched-by-server. These batched requests are sent to their respective servers in one or more messages. From [16], typical batch sizes vary from 24–96 KV pair read requests, with key sizes between 200 B to 12 KB. With RDMA-Memcached's current non-SIMD aware 'Get' protocol, these small message transfers entail using fast two-sided SENDs.

(2) Server Data Access: Upon receiving an MGet request batch, forwarded from the server's communication engine, the assigned Memcached worker performs the following:

(a) Pre-Processing: The incoming request of 'W' keys ('W' <= MGet size 'n') is parsed to extract the individual keys. For each key, a corresponding 32-bit hash value, signature, etc., are computed to prepare it for the HT lookup.

(b) KV Pair Search: In this phase, the HT is probed to locate the payload (e.g., a KV pair memory pointer) corresponding to the 32-bit key hash (i.e., *HT Lookup Phase*). In this case, we can potentially leverage CPU-SIMD data parallelism to accelerate key lookups. Once probing is successful, the KV pair identified is located and read from backend memory slabs (or data cache) and verified against the client-supplied key string to ensure a full match (i.e., *Key Match Lookup Phase*). These matched KV pairs, with variable length keys (8 B–128 B) [13], are returned to the communication engine.

(3) Post-Processing: Once all the KV pairs in the batch are located at the server's memory slabs, the server worker prepares and posts responses to the client, followed by updating the server's metadata to maintain cache freshness (e.g., LRU updates for Memcached). With RDMA-Memcached current non-SIMD aware 'Get' protocol, this entails posting 'W' individual responses to the client.

B. Multi-Get Performance Analysis

We put together a complete end-to-end MGet pipeline, depicted in Figure 4, with the SIMD-aware vertical vectorized HT design in Section II-B2 and RDMA-Memcached [17].

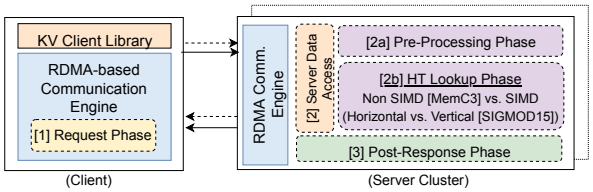


Figure 4: State-of-the-Art End-to-End Flow for MGet

Towards supporting storing pointers to KV pairs, we extend to generate RDMA-Memcached (RDMA-Mem) 32-bit KV pair location IDs, and store it as the payload in 'Cuckoo-Ver (AVX-512)'-based HT backend. Based on this, we study the performance potential of an integrated 'RDMA-Mem+Cuckoo-Ver(AVX-512)' KV store and contrast it with RDMA-Memcached design backed by non-SIMD CPU-optimized MemC3 ('RDMA-Mem+MemC3 (Scalar)'). For a fair comparison, we extend MemC3's optimistic locking and concurrency to the RDMA-Memcached design.

For our analysis, we use two nodes on Cluster A (see Section V), that is equipped with 28-core Intel Skylake nodes and IB EDR (100 Gbps) interconnects. We undertake this experiment over an RDMA-Memcached server running 28 workers with a HT of size 4 M (hashpower=20). We use the "memslap" Multi-Get benchmark [1], configured with 28 clients threads on the client node. We use varying value sizes (32 B and 2 KB) and MGet sizes (i.e., 'N' = 32 or 96), and perform a thorough analysis of the time-wise breakdown into different stages described above.

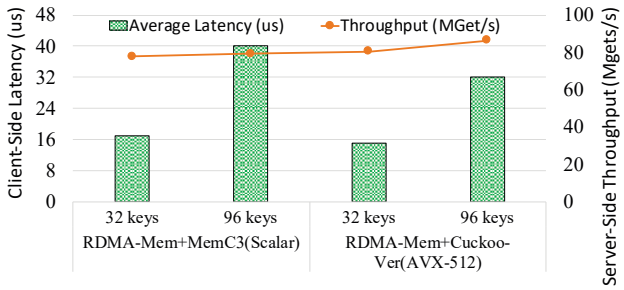


Figure 5: Integrated End-to-End Performance with RDMA-Memcached; Contrasting Vectorization-based Cuckoo HT and Non-SIMD MemC3 backends; KV pair size (20 B,32 B) and MGet sizes of 32 and 96

Figure 5 presents the end-to-end MGet latency with server-side Get throughput and Figure 6 presents the server-side latency breakdown for 16 keys-per-batch, based on the phases in Section III-A, using the "memslap" MGet benchmark. From these figures, we can observe that:

(1) the SIMD-aware 'HT Look-up' phase gains about 3.2x, as seen in Figure 3. But, it is bottlenecked by 'Key Match' phase, and hence SIMD's benefits are not noticeable.

(2) the 'Post-Processing' phase is the most dominant of the different phases in Figure 4. The overhead of

posting individual response for each 'n' keys in the `MGet(k1,k2,..,kn)` dominates the end-to-end latency. This can also be seen as the main factor contributing to the 13%-only gain in the total server performance.

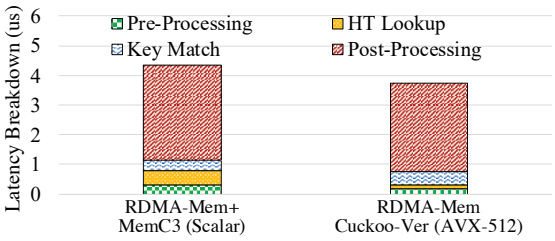


Figure 6: Server-Side Performance Breakdown for RDMA-Memcached with 16 keys/batch; KV pair size (20 B,32 B)

C. Summary

From the analysis in Figures 5 and 6, it is evident that naively replacing the HT backend in existing non-SIMD RDMA-based KV store designs with an SIMD-aware HT cannot give us optimal server scalability and high end-to-end performance. Thus, towards designing a fully functional 'RDMA+SIMD' accelerated KV store, we identify the following challenges and co-design opportunities:

(1) Server-Side Bottlenecks: Existing SIMD-aware HT designs [19] promise up to 3.2x improvement for the 'HT Lookup' phase. However, these are diminished by the non-SIMD aware steps involved in the 'Pre-Processing' and 'KV Pair Search' phases (Phase [2a] and Phase [2b]). Hence, we need a KV store-friendly SIMD-aware HT backend.

(2) Lack of End-to-End SIMD-Aware Designs: The end-to-end latency for MGet is dominated by response communication times ('Post-Processing' phase). We find that this is because the state-of-the-art RDMA-aware KV stores [9], [10], [14] are not optimized for bulk reads. Hence, we need SIMD-aware RDMA protocols for MGets.

IV. SCOR-KV: Design Details

Towards overcoming the challenges identified above, we propose a holistic approach to design an SIMD-aware RDMA-based KV store, which we refer to as 'SCOR-KV' (we use RDMA-based Memcached [17] as our basis). SCOR-KV presents an SIMD-aware Client-Centric and Optimistic RDMA-aware design for Key-Value Stores. Figure 7 presents an overview of our proposed end-to-end for MGet pipeline for SCOR-KV, that introduces the following enhancements into the MGet pipeline:

(1) KV-friendly SIMD-aware hybrid hash table (HT) for the 'Server Data Access' phase (Figure 4 Phase [2b]), that employs an 'optimistic partial key' lookup schemes.

(2) RDMA-optimized 'optimistic lookup'-aware response processing engine at the server (Figure 4 Phase [3]).

(3) Client-centric SIMD-aware request offload engine (Figure 4 Phase [1]).

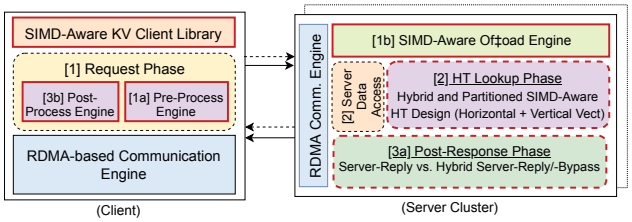


Figure 7: Proposed End-to-End Flow for MGet in SCOR-KV

A. SIMD-Aware Hybrid Hash Table with Optimistic Lookups

Typically, cuckoo HTs and its high-performance variants (presented in Figure 3) inherently enable a constant lookup performance. However, from Section III-A, we can see that the 'HT Lookup' phase entails: (a) an SIMD-aware parallel probing phase to locate KV pair object from the HT using 32-bit key hashes, followed by, (b) a non-SIMD aware key-matching phase that deals with variable length keys. Thus, to maintain the near-constant performance enabled by CPU-SIMD HT probing throughout the 'HT Lookup' phase, we need to make the key-matching phase 'SIMD-aware' and overcome the need to match variable key-lengths. Towards this, we present a hybrid and partitioned SIMD-aware KV-friendly HT design, as shown in Figure 8. It stores a fixed-length 'partial-key' for every KV object indexed in the HT, and matches this 'partial key (`PKey`)' to enable near-constant 'HT Lookup' performance.

1) Partitioned Layout: As in Figure 8, we partition the `(key-hash, KV-obj-ptr)` stored in the typical HT among two distinct tables, i.e., 'Key-Sig HT (`SIMD-HT`)' and 'Partial-Key Table (`PKey-Table`) ', respectively.

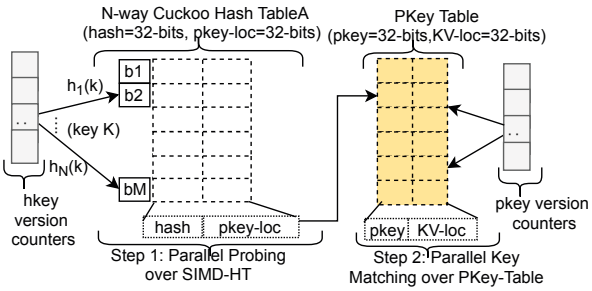


Figure 8: SCOR-KV Hybrid Table Design (+HybridHT): KV-friendly SIMD-aware HT layout with PKey-Table

(1) `SIMD-HT` stores the following tuple: `(key-hash, PKey-Table-ptr)`. It leverages an SIMD-aware vertically vectorized N-way cuckoo HT design (see Section II-B).

(2) `PKey Table` follows a columnar table format (i.e., non-bucketized vertical HT) and stores the following tuple: `(PKey, KV-obj-ptr)`.

(3) As in [28], we use a slab memory management where each KV object is assigned with a 32-bit location ID (i.e, 32-bit KV object pointers), and 32-bit values for `(key-hash, kv-obj-ptr, PKey-Table-ptr)`.

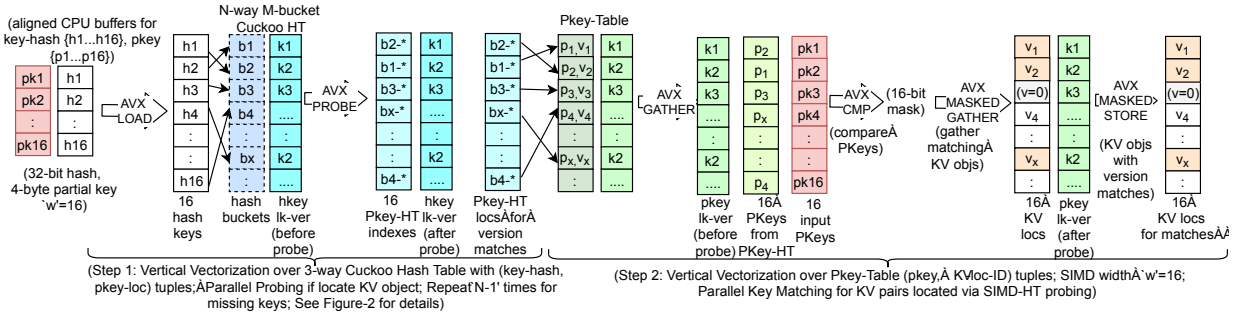


Figure 9: Vectorized operations on the SCOR-KV Hybrid HT with AVX-512 (SIMD width 'w'=16); SIMD-aware 3-way Cuckoo Hashing for HT Probing and SIMD-aware Partial Key Matching

With SCOR-KV's hybrid HT layout, every insert/update `Set(K,V)` operation entails: (a) retrieving a free `PKey-Table` location, (b) updating key into `SIMD-HT` with this `PKey-Table-ptr`, (b) updating first 'P' bytes of the KV pair's variable length key-string into the `PKey-Table` along with the `PKey-Table-ptr`.

2) SIMD-Aware Optimistic Lookups: Similar to typical CPU vectorization-based accelerated designs discussed in Section II-B, we lookup the 'N' keys in the MGet request in the `SIMD-HT` to exploit SIMD data parallelism. For hash-key matches located in `SIMD-HT`, we employ the `PKey-Table-ptr` to index the `PKey-Table` and finding KV pair object with matching keys. These two steps to enable data-parallel key lookups and matches are illustrated in detail in Figure 9. Since we match 'partial keys', we refer to the SCOR-KV's HT Lookups as 'Optimistic'.

As vector gather instructions on the latest CPUs enable loading up to 64-bits per-SIMD lane, we can have `PKey` lengths ('P') of up to 8 bytes (and powers of 2). For instance, if we store 4-byte `PKey` and 4-byte `pointer-to-KV-obj`, we can leverage two 64-bit wide AVX-512 gather (`_mm512_i64gather_epi32`) 32-bit shuffle (`_mm512_shuffle_epi32`), to extract and match 16 keys in the input in parallel. This eliminates server's non-SIMD overheads during the 'HT Lookup' phase.

3) Enabling SIMD-aware Concurrency: While KV store workloads are read-dominated, they are comprised of a mix of Set/MGet/Get operations. So, we need to ensure safe concurrent reads and updates to the backend HT. With SIMD-aware vertical vectorized cuckoo HT, that enable lookup up 'w' distinct keys in parallel, employing traditional concurrency schemes (e.g., fine-grained locks in Memcached [13]) is not feasible as we need to lock 'w' distinct hash buckets. To alleviate this, we extend the optimistic locking mechanism in MemC3 [7], that employs key-version counters, to enable lock-free single-writer/multiple-reader concurrency. Each key is associated with a 'key version' counter, which is atomically incremented only during updates. For both `SIMD-HT` and `PKey-Table`, we use AVX-512 gather-and-compare (`_mm512_gather_i32_epi32`) to check the 'key version' before and after the probing/matching processes to ensure data consistent lookups. A version mismatch signifies an update, and the particular key lookup is retried.

B. RDMA-optimized Response Engine (+PostOptm)

To alleviate the bottlenecks during 'Post-Processing' phase at the RDMA-based KV store server, we need to minimize the round-trip communication times involved per MGet. Based on [9], we propose a latency-optimized 'Hybrid Server-Reply/-Bypass' RDMA protocol, that partially offload the server-side 'Post-Response' phase (Figure 4 phase [3]) to the client, as depicted in phases [3a] and [3b] in Figure 7. It reduces the 'n' two-sided SENDs posted by the servers in [9], [10] per MGet to just one per-server. At each server, all responses are aggregated into a pre-allocated buffer and only the buffer address (and remote HCA key) is communicated to the client, which retrieves them in one or more server-bypassed RDMA Reads.

While 'SIMD-aware optimistic partial key lookups' are ideal, a full comparison of the variable-length key (corresponding to the KV object) is still vital for ensuring the basic MGet semantics. Therefore, unlike default designs [13], we send the 'key' in-line with the corresponding 'value' in the response. Upon receiving batched responses, the client invokes the 'Post-Process' engine ([3b] in Figure 7) to parse and ensure full key matches. This is illustrated in Figure 10. If the optimistic matches fail, the request can be re-sent as a single key 'Get' or with the next concurrent 'MGet'.

C. Client-Centric Request Offloading (+PreOptm)

To alleviate the overheads at the server that is incurred during 'Pre-Processing' phase, we present a client-centric SIMD-aware request engine for MGets. It offloads the server-side 'Pre-Processing' phase (Figure 4 phase [2a]) to the client, as depicted in phases [1a] and [1b] in Figure 7. The RDMA-based SCOR-KV client: (a) performs 'n' hash/signature computations locally, (b) packs them into a contiguous buffer with aligned 'n' 8-byte partial keys, and, (c) posts an RDMA-Write-with-Immediate (RWImm) directly into a pre-leased CPU-aligned server buffers at the

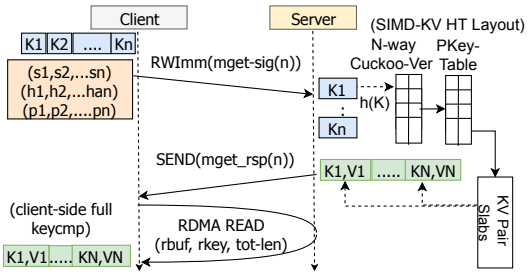


Figure 10: SCOR-KV Hybrid Server-Reply/-Bypass MGet Schemes; with Client-Centric Request Offload ('MGet(hash, pkey)'+RDMA-Write-Imm) and Post-Processing (SEND+RDMA-Read) Engine; backed by SIMD-Aware Hybrid HT with Optimistic Lookups

server. Upon receiving an 'immediate' work request (32-bit immediate value to identify the pre-lease buffer), the server launches the 'Server Data Access' phase via the SIMD-Aware MGet offload engine ([1b] in Figure 7). This is also illustrated in Figure 10. Based on this, we present detailed evaluations of SCOR-KV in Section V.

V. Performance Evaluation

In this section, we present the results of our in-depth analysis of the proposed 'RDMA+CPU-SIMD'-aware SCOR-KV designs. We present evaluations with 'memslap" MGet benchmark [1] and YCSB application benchmark [4].

A. Experimental Setup

We use the **OSU-RI2-Skylake (Cluster A)** for our evaluations. Each node in this testbed is provisioned with Intel Skylake dual fourteen-core processors (28-cores), 128 GB DRAM, connected via Mellanox InfiniBand EDR interconnects (100 Gbps). We study the performance of the following SIMD-aware and non-SIMD KV store designs: (a) SCOR-KV (illustrated in Figure 10), (b) RDMA-enhanced Memcached with Non-SIMD MemC3 [7] backend (MemC3+ RDMA-Mem), and, (c) the naively integrated 'RDMA-Mem+Cuckoo-Ver (AVX-512)' presented in Section III.

For our micro-benchmark studies, we employ the "memslap" MGet benchmark [1], configured with 28 streaming client threads on a single node on Cluster A. We enhance "memslap" MGet benchmark to employ FB's Memcached workload generator (mutilate [15]) to mimic KV store's skewed data access pattern. Since the SIMD-aware HT is employed per-server, we focus on a single KV store server running full-subscription (i.e., 28 workers) on a node on Cluster A. We configure the RDMA-Memcached and SCOR-KV servers with an HT size of 32 MB with 90% load factor. This refers to 2^{20} buckets with $4slots/bucket$ for MemC3, and, 2^{24} hash buckets with 32-bit keys/payloads for Cuckoo-Ver (AVX-512) and SCOR-KV.

B. Performance with Varying KV Pair / MGet Sizes

For the first experiment, we study the end-to-end performance with `MGet(k1,k2,..,kn)` using uniformly random (Uniform) access pattern. We use an MGet size 'n=32', using KV pair sizes: (a) 16 B key and 64 B value, (b) 32 B key and 512 B value, and, (c) 128 B key and 1 KB value. From Figure 11a, we observe that SCOR-KV can improve the overall MGet latency by about 2.6x for small KV pair sizes and about 12% for larger sizes; as compared to the non-SIMD 'RDMA-Mem+MemC3' and naive SIMD-based 'RDMA-Mem+Cuckoo-Ver (AVX-512)' design.

For the second experiment, we study the end-to-end performance with `MGet(k1,k2,..,kn)` with MGet size 'n=16, 64, 96', using KV pair size of 16 B key and 64 B value. We employ the skewed access pattern (Skewed) that mimics KV store data popularity in [3], [15]. From Figure 11b, we can see that SCOR-KV can improve the overall MGet latencies from 1.6x to 3.6x as the MGet request length 'n' increases. Thus, we observe that:

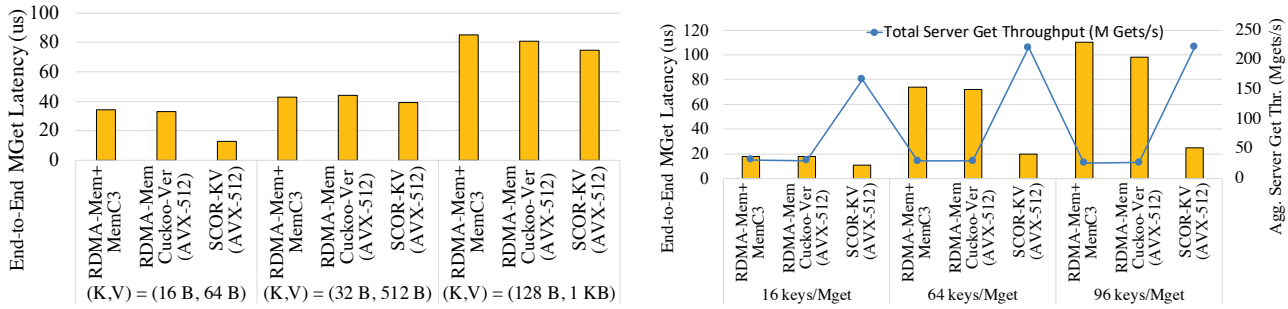
Observation①: For smaller KV pair sizes, irrespective of the MGet size, SIMD-aware and optimistic RDMA-accelerated lookups in SCOR-KV can maintain up to 3x improvement in end-to-end performance, and server-side throughput by about 8.56x. Also, for both uniform and skewed patterns, naively integrated 'RDMA-Mem+Cuckoo-Ver (AVX-512)' does not show significant performance benefits, due to its inherent non-SIMD aware design. On further analysis, we find that the overhead of increased network I/O and memory-bound key matches with large KV pair sizes, seem to dominate over the benefits of CPU-SIMD.

C. Server-Side Performance with SCOR-KV

For our third experiment, we investigate how each of the individual optimizations in SCOR-KV contributes to the overall server performance. We extend the server-side time-wise breakdown in Figure 6 to include SCOR-KV optimizations, as shown in Figure 12a. We use an MGet size 'n=32', using varying KV pair sizes (K, V) of: (a) (16 B, 64 B), and, (b) (128 B, 1 KB). Since AVX-512 allows a max. SIMD width of 'w'=16 (i.e, probe 'w' in parallel), we present the average breakdown for batch sizes of 16 keys.

With 'RDMA-Mem+MemC3' as our baseline, we cumulatively add optimizations proposed in SCOR-KV, as follows: **(1) Cuckoo-Ver**: This refers to naively replacing the non-SIMD HT in the KV store backed with an SIMD-aware (vertical vectorized 'Cuckoo-Ver') HT design. No other additional changes are made to the KV store's MemC3 backend for RDMA-Memcached (as in Section III-A).

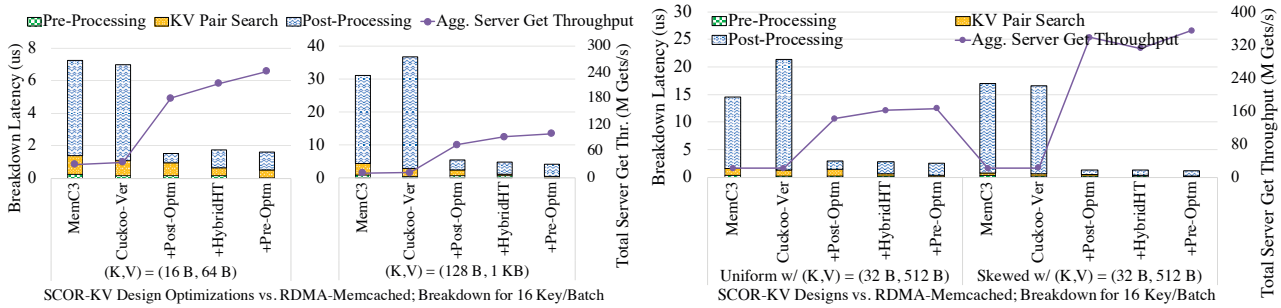
(2) +HybridHT: This is the hybrid/partitioned SIMD-aware HT design optimistic partial key lookups, discussed in Section IV-A, to optimize the 'HT Lookup' phase. The default non-SIMD MGet protocol in RDMA-Memcached, involving two-sided SENDs per key, is employed.



(a) Performance with Varying KV Pair Sizes (Uniform Pattern)

(b) Performance with Varying MGet Sizes (Skewed Pattern)

Figure 11: End-to-End MGet Latency: Contrasting SCOR-KV with RDMA-Memcached+MemC3 / RDMA-Memcached+Cuckoo-Ver(AVX-512)



(a) Varying KV Pair Sizes

(b) Uniform vs. Skewed Access Pattern

Figure 12: Server-Side Time-wise Breakdown for MGet Processing with SCOR-KV optimizations

(3) +PostOptm: This refers to the 'Post-Processing Phase' optimizations in Section IV-B, i.e., the 'Hybrid Server-Reply/Server-Bypass' RDMA-MGet protocol, along with 'optimistic offloading' of key matching to the client-side.

(4) +PreOptm: This refers to the 'Pre-Processing Phase' optimizations in Section IV-C, that enables the server-side pre-processing overheads to be offloaded to the client.

From Figure 12a, we can observe that:

(1) +PostOptm can enable a 5.09x improvement to the overall server-side processing time per MGet, as compared to the default MGet protocol in RDMA-Memcached.

(2) +HybridHT enables a constant KV pair search performance, irrespective of the key size. SCOR-KV can improve the 'HT Lookup' phase by about 2.6x over MemC3 and 1.3x over naively integrated Cuckoo-Ver(AVX-512).

(3) +PreOptm optimizations in SCOR-KV reduces the pre-processing server-side time by about 90%. However, since it accounts for a small percentage of the total MGet processing time, the overall improvement is about 19%.

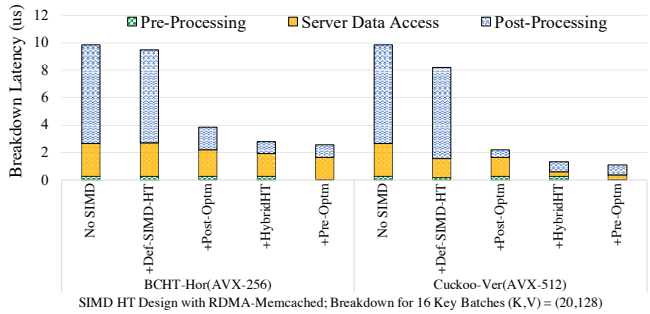
(4) SCOR-KV can improve the total server Get throughput by 3.74x–8.69x over the non-SIMD RDMA-based KV stores (both Non-SIMD MemC3 and naively integrated SIMD-HT), as evident from the line graph (purple) in Figure 12a.

We also extend this time-wise breakdown to study the impact of uniform and skewed access patterns on the server's data processing time. We use an MGet size 'n=32', using varying KV pair size of 32 B key and 512 B value. From Figure 12b, we can observe similar performance impacts across both patterns. However, SCOR-KV can improve the performance by an additional 1.87x over its performance with the uniform access pattern. With a skewed pattern, it is intuitive that the popularly queried KV pairs can be cached in L2. Thus, SCOR-KV can maintain the high performance of the 'HT-fits-in-Cache' scenario, even when the overall HT size exceeds the cache. Therefore:

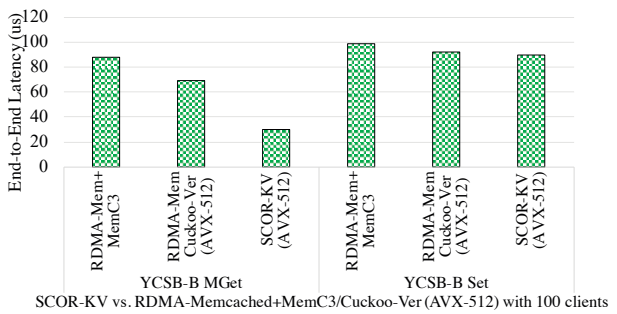
Observation②: SCOR-KV can maintain the benefits of SIMD-aware data-parallel HT probing to improve the overall server-side Get performance. SIMD-aware hybrid HT with optimistic lookups with RDMA-based schemes are more beneficial for skewed KV workloads, as they can exploit the inherent caching behavior of multi-core CPUs more efficiently than the uniform data access pattern.

D. SCOR-KV with Horizontal Vectorized SIMD HT

To demonstrate that SCOR-KV's SIMD-aware optimizations apply to SIMD-based designs beyond the 3-way vertical cuckoo HT employed in this paper, we present evaluations with a horizontal vectorized HT, i.e, bucketized cuckoo HT (BCHT) with 4 slots/bucket and 32-bit key and payloads using AVX-2 (256-bit) vectors to probe hash bucket slots in parallel. We replace SCOR-KV's N-way Cuckoo



(a) Performance with SIMD-based for Bucketized Cuckoo HT (Horizontal Vectorization)



(b) Concurrency with YCSB Workload B; 112 clients over 4 nodes and a 28 worker KV store server

Figure 13: SCOR-KV Performance with Different Vectorization Approaches and YCSB Application Workload

HT with this 'BCHT-Hor (AVX-256)' design, and keep our proposed hybrid HT layout and client-centric RDMA-optimized offloading schemes.

We extend the server-side time-wise breakdown studies presented in Section V-C. From Figure 13a, we can observe that, as compared to the RDMA-Memcached server with Non-SIMD MemC3, SCOR-KV can improve the 'HT Lookup' phase by 1.47x for SIMD-HT based on BCHT-Hor(AVX-256). This is about 2x lesser than the benefits observed with Cuckoo-Ver (AVX-512). We attribute this to the need to lock and probe keys individually over the horizontal vectorized BCHT design.

E. Evaluations with YCSB

Finally, to evaluate the performance of SCOR-KV with KV store mixed workloads (read/write mix with MGet and Set operations), we present end-to-end latency evaluations with the popular YCSB benchmark [4]. We employ the YCSB workload B (YCSB-B) with 95:5 Read:Write mix, with KV pair size of (16 B, 64 B) and an MGet length of 16 keys (we employ coalesce individual Gets into an MGet). We set up a small cluster with a single KV store server with 28 workers on a node, and run a total of 112 clients over 4 client nodes, on Cluster A. From Figure 13b, we can observe that SCOR-KV can maintain 2.2x performance improvement for MGet over SIMD-based 'RDMA-Mem+Cuckoo-Ver (AVX-512)' and non-SIMD 'RDMA-Mem+MemC3' KV store designs. Also, we observe that SCOR-KV can maintain the RDMA KV store's Set performance, and the hybrid and partitioned HT layout does not incur any performance overheads for KV store update operations.

Thus, our proposed SIMD-aware MGet schemes can benefit the overall KV store performance. While SCOR-KV's optimizations are presented for MGets, the optimistic client-centric designs can be applied to any in-memory storage middleware that needs to scale the server-side performance.

VI. Related Work

The idea of leveraging CPU-SIMD to improve the performance of critical database operations has been well researched in literature. For instance, CPU vector instructions are being leveraged to accelerate database scan, sort, aggregation [8], [18], [19], [29], hash join operations [19], [20], and bloom filters [12]. Similarly, network packet processing applications dealing with batched HT lookups [21], also leverage CPU-SIMD. Additionally, in-depth performance analysis and micro-benchmarks for these state-of-the-art SIMD-aware HT designs on the latest CPU architectures have been presented in [24]. On the other hand, Zhang et al. [28] leverage the high memory bandwidth of modern GPGPUs to scale the KV store server's performance. While its GPU-optimized cuckoo HT outperforms its CPU-optimized counterpart, the client-side latency suffers due to the GPU kernel launch and server-side batching overheads.

Consequently, several works have proposed on exploiting RDMA-based communication engines for accelerating KV store performance [9]–[11], [14], [22], [25]. In addition to these, holistic designs like MICA [11], that focus on partitioning data access among cores to avoid concurrency overheads have been proposed. However, these state-of-the-art designs are mostly focused on optimizing performance for point-to-point queries like Set/Get, and are not optimized for bulk read requests like MGet. Unlike these works, in this paper, we propose an SIMD-aware RDMA-optimized KV store that co-designs the client-to-server data processing pipeline to exploit existing CPU-vectorized HT designs without compromising on the end-to-end performance.

VII. Conclusion & Future Work

In this paper, we propose an SIMD-Aware Client-Centric and Optimistic RDMA-based Key-Value Store, SCOR-KV, that optimally exploits 'RDMA+SIMD', to accelerate read-heavy MGet operations in KV stores on emerging multi-core CPU architectures. SCOR-KV presents an SIMD-conscious KV store friendly hash table layout, based on the vertically vectorized N-way cuckoo hashing design. It proposes optimistic KV pair lookups schemes, that enable offloading server-side pre- and post-processing overheads to the client while maintaining the end-to-end 'Multi-Get' semantics.

To complement this, we present 'Multi-Get'-aware RDMA schemes to maximize the overall performance.

Our performance evaluations on the latest Intel Skylake CPUs and IB EDR interconnects, show that our proposed SCOR-KV can achieve up to 3.7-8.6x improvement in server-side Get throughput, while improving end-to-end MGet latencies for read-heavy YCSB workloads by about 2.2x over the state-of-the-art RDMA-based Memcached server running over non-SIMD cache-optimized MemC3 backend. In the future, we plan to extend our proposed designs to: (a) larger-scale distributed Memcached workloads, and, (b) leverage SCOR-KV's generic designs into other memory-centric key-value stores like Redis [2].

REFERENCES

[1] "memslap," http://docs.libmemcached.org/bin/memslap.html.

[2] "Redis," https://redis.io/.

[3] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload Analysis of a Large-Scale Key-Value Store," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1. ACM, 2012, pp. 53–64.

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *The Proceedings of the ACM Symposium on Cloud Computing (SoCC '10)*, Indianapolis, Indiana, June 2010.

[5] J. Doweck, W. Kao, A. K. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, March 2017.

[6] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast Remote Memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[7] B. Fan, D. G. Andersen, and M. Kaminsky, "MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 371–384.

[8] P. Jiang and G. Agrawal, "Efficient SIMD and MIMD Parallelization of Hash-based Aggregation by Conflict Mitigation," in *Proceedings of the International Conference on Supercomputing*. ACM, 2017, p. 24.

[9] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, and D. K. Panda, "Memcached Design on High Performance RDMA Capable Interconnects," in *Proceedings of the 2011 International Conference on Parallel Processing*, ser. ICPP '11, Washington, DC, USA, 2011.

[10] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA Efficiently for Key-Value Services," in *Proceeding of SIGCOMM '14*, Aug 2014.

[11] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A Holistic Approach to Fast In-memory Key-value Storage," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2014.

[12] J. Lu, Y. Wan, Y. Li, C. Zhang, H. Dai, Y. Wang, G. Zhang, and B. Liu, "Ultra-Fast Bloom Filters using SIMD techniques," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, June 2017, pp. 1–6.

[13] "Memcached: High-Performance, Distributed Memory Object Caching System," http://memcached.org/.

[14] C. Mitchell, Y. Geng, and J. Li, "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store," in *Proceeding of USENIX ATC '13*, Jun 2013.

[15] Mutilate, https://github.com/leverich/mutilate.

[16] 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 *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, 2013.

[17] NOWLAB, "High-Performance Big Data (HiBD)," http://hibd.cse.ohio-state.edu, 2019.

[18] M. Pilman, K. Bocksrocker, L. Braun, R. Marroquín, and D. Kossmann, "Fast Scans on Key-Value Stores," *Proceedings of the VLDB Endowment 10*, no. 11, pp. 1526–1537, 2017.

[19] O. Polychroniou, A. Raghavan, and K. A. Ross, "Rethinking SIMD Vectorization for In-Memory Databases," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 1493–1508.

[20] K. A. Ross, "Efficient Hash Probes on Modern Processors," in *IEEE 23rd International Conference on Data Engineering (ICDE '07)*. IEEE, 2007, pp. 1297–1301.

[21] N. L. Scouarnec, "Cuckoo++ Hash Tables: High-Performance Hash Tables for Networking Applications," in *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*. ACM, 2018, pp. 41–54.

[22] D. Shankar, X. Lu, N. Islam, M. Wasi-Ur-Rahman, and D. K. Panda, "High-Performance Hybrid Key-Value Store on Modern Clusters with RDMA Interconnects and SSDs: Non-blocking Extensions, Designs, and Benefits," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 393–402.

[23] D. Shankar, X. Lu, and D. K. Panda, "Boldio: A Hybrid and Resilient Burst-Buffer Over Lustre for Accelerating Big Data I/O," in *2016 IEEE International Conference on Big Data*, Dec 2016, pp. 404–409.

[24] D. Shankar, X. Lu, and D. K. Panda, "SimdHT-Bench: Characterizing SIMD-Aware Hash Table Designs on Emerging CPU Architectures," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, November 2019.

[25] M. Su, M. Zhang, K. Chen, Z. Guo, and Y. Wu, "RFP: When RPC is Faster Than Server-Bypass with RDMA," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17, 2017, pp. 1–15.

[26] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu, "BurstMem: A High-Performance Burst Buffer System for Scientific Applications," in *2014 IEEE International Conference on Big Data*, October 2014.

[27] Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Characterizing Facebook's Memcached Workload," *IEEE Internet Computing*, vol. 18, no. 2, pp. 41–49, 2014.

[28] K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, and X. Zhang, "Mega-KV: A Case for GPUs to Maximize the Throughput of In-Memory Key-Value Stores," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1226–1237, 2015.

[29] J. Zhou and K. A. Ross, "Implementing Database Operations using SIMD Instructions," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. ACM, 2002, pp. 145–156.