Scalable Datacenter and Memcached

Qinghua Zhou
zhou.2595@osu.edu
Computer Science and Engineering
Outline

- Scalable Data Centers and Memcached
  - Memcached Design on High Performance RDMA Capable Interconnects (ICPP '11)
  - SSD-Assisted Hybrid Memory to Accelerate Memcached over High Performance Networks (ICPP '12)
  - High-Performance Hybrid Key-Value Store on Modern Clusters with RDMA Interconnects and SSDs: Non-blocking Extensions, Designs, and Benefits (IPDPS '16)
  - SCOR-KV: SIMD-Aware Client-Centric and Optimistic RDMA-Based Key-Value Store for Emerging CPU Architectures (HiPC '19)
Memcached Design on High Performance RDMA Capable Interconnects


Department of Computer Science and Engineering,
The Ohio State University

(ICPP '11)
Background

• MPI software already utilizes the strengths of RDMA to a great extent.
• Applications using Memcached are still relying on Sockets.
• Most applications now depend on standardized middle-ware APIs such as Memcached, Hadoop etc., not on Sockets API directly.
• Can we re-design Memcached from the ground up to utilize RDMA capable networks while keeping the same external APIs?
Overview of High-Performance Networking Stacks and Memcached

Fig. 1. Overview of High-Performance Networking Stacks and Memcached
Design Unified Communication Runtime (UCR) for Data-Center Environments

- UCR: Unify the communication runtime requirements
  - Scientific parallel programming models: MPI, PGAS
  - Data-Center Middleware (e.g. Memcached)

- Communication Model and Connection Establishment
  - New designed UCR follows an end-point model, client must establish a bi-directional end-point with the server
  - Synchronization with timeouts. Add UCR methods to wait for events.

- Active Messaging in UCR
  - An active message consists of two parts: header and data
  - Use Counters to track progress of active messages
  - Optimization for short messages, header and data may be combined into one network transaction
Active Messaging in UCR

- Active message API:
- Types of counters: origin_counter, target_counter and completion_counter
- UCR internal message is optional (origin/completion counter is NULL)
Design Memcached using UCR

• Connection Establishment
  – UCR client request causes a libevent notification at the server side
  – A worker thread is assigned (in a round-robin manner) for this client
  – Both threads create end points and start communication using active messages over UCR

• Set Operations
  – Client puts an object (item) in Memcached server memory, with a key of hash value
  – Target server chosen by hash function from the pool of available servers.
  – Issue set operation using active message to the server

• Get Operations
  – Client gets an object (item) from a Memcached server
  – Issue get operation using active message to target server (through hash function)
# Experimental Setup

<table>
<thead>
<tr>
<th>Intel Clovertown Cluster(A)</th>
<th>Intel Westmere Cluster(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 compute nodes</td>
<td>144 compute nodes</td>
</tr>
<tr>
<td>Intel Xeon Dual quad-core processor, 2.33 GHz, 6GB RAM</td>
<td>Intel Xeon Dual quad-core processor, 2.67 GHz, 12GB RAM</td>
</tr>
<tr>
<td>PCIe 1.1 interface</td>
<td>PCI-Ex Gen2 interfaces</td>
</tr>
<tr>
<td>ConnectX DDR IB HCA (16 Gbps data rate), Chelsio T320 10GbE Dual Port Adapter with TCP Offload</td>
<td>MT26428 QDR ConnectX HCAs (36 Gbps data rate)</td>
</tr>
<tr>
<td>144 Port Silverstorm IB DDR switch</td>
<td>171-port Mellanox QDR switch</td>
</tr>
<tr>
<td>Fulcrum Focalpoint 10 GigE switch</td>
<td></td>
</tr>
</tbody>
</table>
Performance of Set and Get on single cluster

Fig. 3. Latency of Set and Get Operations on Cluster A

Fig. 4. Latency of Get and Set Operations on Cluster B
Performance of Mixed Set and Get on single cluster

- Non-Interleaved
  - Mix of 10% Set operations and 90% Get operations, 10 Sets followed by 90 Gets

- Interleaved
  - Mix of 50% Set operations and 50% Get operations, 1 Set is followed by 1 Get

Fig. 5. Latency of Small Messages for Non-Interleaved (Set 10% Get 90%) and Interleaved (Set 50% Get 50%) Operations
Performance of Memcached with Multiple Clients

- Clients: 8, 16 on different node than Memcached server
- Total number of aggregate transactions executed per second for 4 and 4K bytes
- Cluster A and B: UCR is 6x better than 10 GigE-TOE and SDP for 4 byte
- SDP performance is lower than 10 GigE-TOE(Cluster A) and IPoIB (Cluster B)

Fig. 6. Number of Transactions per Second for Get Operation
Conclusion

• A novel communication library **Unified Communication Runtime (UCR)** to enable data-center middleware to effectively use features of High performance RDMA capable networks.

• A new design of Memcached using UCR to dramatically improve performance when using RDMA capable networks.

• Provide a detailed performance comparison of Memcached design compared to unmodified Memcached using IPoIB, Sockets over RDMA and 10 Gigabit Ethernet networks with hardware-accelerated TCP/IP.
SSD-Assisted Hybrid Memory to Accelerate Memcached over High Performance Networks

Xiangyong Ouyang, Nusrat S. Islam, Raghunath Rajachandrasekar, Jithin Jose, Miao Luo, Hao Wang and Dhabaleswar K. Panda

Department of Computer Science and Engineering,
The Ohio State University

(ICPP ‘12)
Background

• Memcached
  – As a memory caching layer to cache database request results
  – Spare memory in data-center servers can be aggregated to speed up lookups of frequently accessed information
  – Difficult to further scale the memory pool size by packing more RAM into individual servers, or by expanding the server arrays.

• Solid State Drive (SSD)
  – NAND-flash based Solid State Disk has attracted a lot of attention as an alternative storage device
  – Very fast in random read, and it has a superior write bandwidth than hard drives
  – Studies try to expand effective memory size by mapping SSD into the virtual memory
Design Challenge

- Limitation of mapping SSD into virtual memory
  - Significant performance loss due to the overhead in Virtual Memory management to treat SSD as a swap device
  - Entire flash pages have to be loaded/overwritten even if a single byte is to be read/written.
  - Conventional virtual memory management also incurs heavy software stack overhead inside the kernel

Fig. 1. Memcached Get Latency at 1KB Object Size. “IB-RAM” is Memcached Running on Native InfiniBand with Data in RAM. “IB-VirtualMem” is Similar to “IB-RAM”, but Data is Stored at Virtual Memory Mapped SSD. The SSD is Preloaded with 30 GB Data. “SSD-Read” Gives the SSD Read Latency. The SSD used is Fusion-io ioDrive 80 GB SLC.
Design Alternatives

- Data missing in Memcached server has price of performing a database query with high latency
- Straightforward Virtual Memory Swap incurs substantial overhead

<table>
<thead>
<tr>
<th>Method</th>
<th>InfiniBand Verbs</th>
<th>InfiniBand IPoIB</th>
<th>10 GigE</th>
<th>1 GigE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Query Latency</td>
<td>N/A</td>
<td>10763</td>
<td>10724</td>
<td>11220</td>
</tr>
<tr>
<td>Memcached Get Latency (in RAM)</td>
<td>10</td>
<td>60</td>
<td>40</td>
<td>150</td>
</tr>
<tr>
<td>Memcached Get Latency (in SSD mapped Virtual Memory)</td>
<td>347</td>
<td>387</td>
<td>362</td>
<td>455</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSD Latency</th>
<th>Random Read</th>
<th>Random Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE I**

Latency (in $\mu$s) of Memcached and MySQL with different network (1KB record size). SSD latency is measured with Fusion-io ioDrive [19] 80GB at 1 KB access size.
Hybrid Memory Architecture and Design

- Augment RAM with SSD to build an efficient memory-stack to store key-value pairs

Fig. 2. Possible Approaches to Exploit SSD in Memcached
SSD-Assisted Hybrid Memory Architecture

- Read/Write Buffer
- In-memory Lookup Table
- Log-Structured SSD Storage

Fig. 3. SSD-Assisted Hybrid Memory Architecture
Analytical Model Of Memcached

- **Basic approach**

  \[ L_b = h_1 t + (1 - h_1)(t + q + t) = (1 - h_1)q + 2t - h_1 t \]

  - In-RAM hit ratio: \( h_1 = 0.2 \)
  - Network round trip time: \( q = 11,000 \ \mu s \)
  - Database query latency: \( t = 10 \ \mu s \)
  - \( L_b = 8,818 \ \mu s \)

- **Hybrid approach**

  \[ L_h = h_1 t + (1 - h_1)[h_2(t + r) + (1 - h_2)(t + q + t)] = (1 - h_2)(q + t) + t + h_2 r \]

  - In-SSD hit ratio: \( h_2 = 0.8 \)
  - \( q = 11,000 \ \mu s \), and \( t = 10 \ \mu s \)
  - SSD read/write latency \( r = 68 \ \mu s \), \( w = 70 \ \mu s \)
  - \( L_h = 2,266 \ \mu s \) (3.9X Speedup)
Experimental Setup

- Cluster: 8 processor cores on 2 Intel Xeon 2.33GHz Quad-core CPUs, 6 GB Main memory, 250GB ST3250310NS Hard drive
- Mellanox MT25208 DDR (16Gbps) HCA
- 10GigE adapter by Chelsio Communications (T320) and standard 1GigE adapter
- Integrate the proposed hybrid memory as the memory allocator into Memcached-1.4.5
- A Fusion-io ioDrive 80GB SSD is plugged into the Memcached server as part of the SSD-assisted hybrid memory
- Hybrid memory Read/Write Buffer is fixed to be 256 MB
Hybrid Memory: Basic Operation Performance

- Random Read Latency: 3.6x-3.8x speedup
- Random Write Latency: 3.2x-3.4x speedup
- Insert New object latency: 1.1x-2.6x speedup

Fig. 4. Hybrid Memory Basic Operation Latency. Have Preloaded 30GB data, Hybrid Memory Buffer Size=256MB.
Hybrid Memory: Operation Throughput

- Steady increase in total operation throughput
- Read-only has a slightly higher throughput than mixed read and write accesses, minimum drop in write throughput compare to read throughput
Memcached Performance with hybrid memory

- Memcached Latency:
  - Improve get latency by $3.7x$, set latency by $3.0x$.

- Memcached Throughput:
  - Output perform VMS by $3.6x$ to $5.3x$. Close to native hybrid memory throughput.

Fig. 9. Memcached Performance. Hybrid Memory vs. VMS. Have Preloaded 30 GB data, Hybrid Memory Buffer Size=256 MB. “Get” Reads Existing Objects, “Set” Inserts New Objects.
Hybrid Memory Performance in a Datacenter

- Basic approach: relatively small RAM size, frequent expensive database queries
- Hybrid approach: cache more data in the larger SSD
  30GB SSD: 95% hit ratio, 11.3X speedup
- Analytical Model matches very well with the actual result

<table>
<thead>
<tr>
<th></th>
<th>Basic</th>
<th>0.33</th>
<th>0.50</th>
<th>0.76</th>
<th>0.90</th>
<th>0.95</th>
<th>0.99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit Ratio</td>
<td></td>
<td>0.33</td>
<td>0.50</td>
<td>0.76</td>
<td>0.90</td>
<td>0.95</td>
<td>0.99</td>
</tr>
<tr>
<td>Latency</td>
<td>7349</td>
<td>5628</td>
<td>2692</td>
<td>1154</td>
<td>650</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Analytical Model Error</td>
<td>1.4%</td>
<td>2.4%</td>
<td>2.6%</td>
<td>5.0%</td>
<td>3.6%</td>
<td>6.7%</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE II**
Resemble a datacenter environment: Memcached get latency (μs) with hybrid memory and InfiniBand network.
Conclusion

- First work that integrates the cutting edge SSD and InfiniBand-verbs into Memcached to accelerate its performance.
- Proposes SSD-Assisted Hybrid Memory that expands RAM with SSD to enlarge the available memory size.
- The random read and write of a 1KB object is 3.1X and 3.5X faster respectively with hybrid memory.
- Demonstrates a 3.7X and 3.0X speedup to perform a Memcached Get and Set operation. Memcached operation throughput is improved by up to 5.3 times.
High-Performance Hybrid Key-Value Store on Modern Clusters with RDMA Interconnects and SSDs: Non-blocking Extensions, Designs, and Benefits


Department of Computer Science and Engineering, The Ohio State University
(IPDPS '16)
Background

• High-performance key-value stores play a crucial role in accelerating the data-intensive applications.

• High-performance storage (e.g. SATA SSD, NVMe SSD) and interconnect (e.g. InfiniBand) on modern HPC clusters has directed significant efforts towards leveraging a ‘RAM+SSD’ hybrid storage, to increase the data retention in existing in-memory key-value stores.

• Advanced designs have been proposed to enhance the performance of key-value stores, from the perspective of communication and I/O.
Comparison of Memcached Designs

- Default Memcached: IPoIB-Mem
- Leverage the native RDMA protocol: RDMA-Mem
- Hybrid memory designs with SSD support: H-RDMA-Def, FatCache

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA-based Communication</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Hybrid Memory with SSD Basic</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Adaptive I/O enhancements</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NVMe-SSD</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Non-Blocking API Extensions</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 1: Design Comparison with Existing Work
Performance characteristics of different Memcached designs

- **Data fits in memory**
  - RDMA-Mem and H-RDMA-Def outperform IPoIB-Mem

- **Data does not fit in memory**
  - Additional caching layer miss penalty to access a backend database
  - H-RDMA-Def: 15-17x performance degradation due to SSD access
Bottleneck Analysis in RDMA-Enhanced Hybrid Memcached

- Main time cost: Time blocking for completion of set/get operation and direct SSD IO
- Solution: Non-blocking and adaptive I/O
Non-blocking Memcached APIs

- Memcached Request Structure:
  - memcached isset and memcached iget
  - memcached bset and memcached bget
  - memcached_wait and memcached_test
- Separate the request issue and completion phases
- Avoid significant overheads due to client-side blocking waits and server-side SSD I/O.
Proposed Design for Accelerating Hybrid Memcached for HPC Clusters

A. Enhanced Runtime Design For Non-Blocking Libmemcached API

- Supporting Non-Blocking API Semantics: `memcached_iset` and `memcached_iget`
  - Send out request header+Key, return completion flag to guarantee request completion

- Guarantee Re-usable Key-Value Buffers at Client: `memcached_bset` and `memcached_bget`
  - Wait for the underlying RDMA communication engine to communicate that it has sent out the data (memcached_bset)/ header(memcached_bget)
  - Return a completion flag to check actual communication completion, user-provided buffers are reusable

- Communication Completion
  - `memcached_wait` (blocking call): wait until it receives server’s response
  - `memcached_test` (non-blocking call): check if server’s response is available
  - sender buffer is now free to reuse for iget/iset operations
New Memcached API Design with Non-Blocking Operations
Proposed Design for Accelerating Hybrid Memcached for HPC Clusters (Cont’d)

• B. Enhancements to Hybrid Memcached Server
  – Facilitating Non-Blocking API On The Server
    • Separate the memory access/update phase and communication phase, cache update phase
  – SSD I/O Data Managament Schemes
    • The slab allocator switches between mmaped-I/O and cached I/O, when data is being evicted to the SSD due to insufficient memory slabs.
# Experimental Setup

<table>
<thead>
<tr>
<th>SDSC Comet (Cluster A)</th>
<th>OSU NowLab (Cluster B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 12-core Intel Xeon E5-2680 v3 (Haswell) processors, 128GB DDR4 DRAM</td>
<td>2 10-core 2.6 GHz Intel Xeon E5-2660 v3 (Haswell) processors, 64 GB main memory</td>
</tr>
<tr>
<td>320GB local SATA-SSD</td>
<td>Intel P3700 NVMe-SSD</td>
</tr>
<tr>
<td>CentOS operating system</td>
<td>Red Hat Enterprise Linux Server release 6.5</td>
</tr>
<tr>
<td>56Gbps FDR InfiniBand with rack-level full bisection bandwidth, 4:1 over-subscription cross-rack bandwidth</td>
<td>Mellanox 56Gbps FDR InfiniBand HCAs PCI Express Gen3 interfaces</td>
</tr>
</tbody>
</table>
Overall Performance Evaluation

- H-RDMA-Opt-NonB-i/b have similar performance as in-memory design RDMA-mem
- H-RDMA-Opt-Block (SSD adaptive I/O) delivers 2x improvement over H-RDMA-Def
- H-RDMA-Opt-NonB-i/b get 10-16x improvement over H-RDMA-Def, 3.3-8x over H-RDMA-Opt-Block
Overlap with Non-blocking API

- H-RDMA-NonB-i (iset/iget): up to 92% overlap for both workloads, H-RDMA-NonB-b (bset/bget): up to 89% overlap for read-only workload
- H-RDMA-NonB-b: <12% overlap for write-heavy workload (ensure buffer reusability)
- H-RDMA-Opt-NonB-i/b can improve performance by about 65-89% over H-RDMA-Opt-Block and H-RDMA-Def-Block.
Scalability with Non-Blocking API

- Adaptive I/O schemes used in H-RDMA-Opt-Block improve the throughput by about 1.3x over the current direct I/O-based design of H-RDMA-Def-Block.
Evaluation with NVMe SSD

- H-RDMA-Opt-Block: 54-83% improvement over H-RDMA-Def-Block
- H-RDMA-Opt-NonB-b/i: 48-80% improvement over both H-RDMA-Opt-Block and H-RDMA-Def-Block
Conclusion

• Propose **non-blocking** APIs for Memcached with operation completion guarantees to **overlap** the client-side waits and the server-side SSD access times with other computations or communications.

• Suggest **adaptive I/O** enhancements for the Memcached slab manager, in order to achieve high performance with hybrid Memcached running over SATA/NVMe-SSDs.

• Improve the Set/Get latency by up to **16x** over the current hybrid design for RDMA-based Memcached, when all data cannot fit in-memory; up to **3.6x** over default Memcached over IPoIB, when all data fits in memory.

• Improve the server throughput of the hybrid Memcached design by about **2.5x**.
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
(HiPC '19)
Background

- 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.
- There’s a significant need for fast and scalable ‘**Multi-Get**’ support for read-mostly KV store workloads.
- Modern CPU architectures (e.g., Intel Skylake, Cascade Lake) that support **vector registers** which can fit an entire cache-line (512-bits vectors).
- ‘**Single Instruction Multiple Data**’ (SIMD) vector instructions have been leveraged to accelerate lookups over high-performance hash tables.
Motivation

• 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

• Existing RDMA-based KV stores use a hash table in their backend to store and index KV pairs
  – Have not been adapted to leverage the potential of CPU-SIMD
  – Not optimized for bulk read operations like Multi-Get(k1, k2,..,kN)).

• Explore the SIMD-aware backend designs that support read-intensive workloads over high-performance RDMA enhanced KV stores
  – Potential for leveraging SIMD aware hash tables for accelerating KV store?
  – Performance benefits of CPU-SIMD by naively replacing the hash table backend?
  – Eliminate the bottlenecks to optimally exploit both RDMA and CPU-SIMD
Non-SIMD hash table (HT) designs

- CPU-Optimized Key-Value Store Designs (i.e., MemC3)
  - A CLOCK-based LRU-approximating eviction algorithm to maintain cache freshness
  - A cache-optimized optimistic concurrent cuckoo hashing via atomic key-version counters.
  - Each bucket fits into a single cache-line (40 B), load factor: 90-95%

![Figure 1: MemC3 Hash Table Design](image-url)
**SIMD-Aware hash table (HT) designs**

- **Vertical Vectorization for Data Parallel HT Lookups**
  - Single HT probe operation to iterate over multiple input keys
  - Returns a vector of ‘payloads’ (KV object pointer) corresponding to the matching key
  - **N-way** cuckoo HT (BCHT with ‘M’ (=1) slot per bucket and N hash functions)
  - Intel CPU enable 512-bit vectors (AVX-512) ‘N’-way cuckoo HT: Lookup ‘n’ keys in N*n/16 iterations vs N*n total iterations of non-SIMD

![Diagram of 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.](image-url)
SIMD-Aware hash table (HT) designs (Cont’d)

- Stand-Alone HT Performance with Vertical Vectorization
  - Key access patterns: uniformly random (Uniform) and skewed access (Skewed)
  - Input 32-bit column of 1G keys with about 90% selectivity
  - Output 32-bit column with matching payloads
  - Cuckoo-Ver outperforms MemC3 by 2.7x-6.6x (HT size fit into L2 cache)
  - Cuckoo-Ver outperforms MemC3 by 1.63x-2.6x (HT size exceeds L2 cache)

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
Integrating CPU-SIMD into RDMA KV Store

• Extending RDMA KV Store with SIMD HT
  – Request Phase
    • Each key in MGet is mapped to a specific Memcached server, requests are batched by server
  – Server Data Access
    • a) Pre-Processing: incoming request of ‘W’ keys is parsed to extract the individual keys, compute 32-bit hash value
    • b) HT Lookup Phase (KV Pair Search): HT is probed to locate the payload, read from backend memory slabs, verified against the client-supplied key string to ensure a full match
  – Post-Processing
    • Prepares and posts responses to the client, updating the server’s metadata to maintain cache freshness (e.g. LRU update)
Multi-Get Performance Analysis
- Only 13% gain in the total server performance
- SIMD-aware ‘HT Look-up’ phase gains about 3.2x, it is bottlenecked by ‘Key Match’ phase
- ‘Post-Processing’ phase is the most dominant
Challenges of designing a fully functional ‘RDMA+SIMD’ accelerated KV store

• Naively replacing the HT backend in existing non-SIMD RDMA-based KV store designs with a SIMD-aware HT cannot get optimal server scalability and performance

• Server-Side Bottlenecks
  – Improvement for the ‘HT Lookup’ phase diminished by ‘Pre-Processing’ and ‘Key Match’ phase.
  – Need a KV store-friendly SIMD-aware HT backend.

• Lack of End-to-End SIMD-Aware Designs
  – End-to-end latency for MGet is dominated by response communication times (‘Post-Processing’ phase)
  – Need SIMD-aware RDMA protocols for MGets
SCOR-KV: Design Details

- SIMD-aware Client-Centric and Optimistic RDMA-aware design for Key-Value Stores
- Overview of proposed end-to-end for MGet pipeline for SCOR-KV
  - KV-friendly **SIMD-aware hybrid** hash table for the ‘Server Data Access’ phase
  - RDMA-optimized ‘**optimistic lookup**’-aware response processing engine
  - **Client-centric** SIMD-aware request offload engine

![Diagram](attachment:image.png)

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

**Figure 7: Proposed End-to-End Flow for MGet in SCOR-KV**
(A). SIMD-Aware Hybrid Hash Table with Optimistic Lookups

- Need to make the key-matching phase ‘SIMD-aware’ and overcome the need to match variable key-lengths.
- Hybrid and partitioned SIMD-aware KV friendly HT design
  - Partition the (key-hash, KV-obj-ptr) among Key-Sig HT (SIMD-HT)’ and ‘Partial-Key Table (PKey-Table)’

Figure 8: SCOR-KV Hybrid Table Design (+HybridHT): KV-friendly SIMD-aware HT layout with PKey-Table
• (A). SIMD-Aware Hybrid Hash Table with Optimistic Lookups
  – SIMD-Aware Optimistic Lookups
    • Two steps to enable data-parallel key lookups and matches
    • HT Lookups is ‘Optimistic’: match ‘partial keys’
    • Vector gather instructions eliminates server’s non-SIMD overheads during the ‘HT Lookup’ phase
  – Enabling SIMD-aware Concurrency
    • Employs key-version counters, to enable lock-free single-writer/multiple-reader concurrency
SCOR-KV: Design Details (Cont’d)

- (B). RDMA-optimized Response Engine (+PostOptm)
  - To minimize the round-trip communication times involved per MGet in ‘Post-Processing’ phase
  - Latency-optimized ‘Hybrid Server-Reply/-Bypass’ RDMA protocol
    - Offload the server-side ‘Post-Response’ phase to the client
    - All responses are aggregated into a pre-allocated buffer and only the buffer address is communicated to the client
  - Full comparison of variable-length key
    - Send the ‘key’ in-line with the corresponding ‘value’ in the response
    - Client invoke ‘Post-Process’ engine parse and ensure full key matches
(C). Client-Centric Request Offloading (+PreOptm)

- Client performs ‘n’ hash/signature computations locally
- Packs them into a contiguous buffer with aligned ‘n’ 8-byte partial keys
- Posts an RDMA-Write-with-Immediate (RWImmm) directly into a pre-leased CPU-aligned server buffers at the server
- Server launches the ‘Server Data Access’ phase via the SIMD Aware MGet offload engine
Experimental Setup

• Cluster: OSU-RI2-Skylake
  – Intel Skylake dual fourteen-core processors (28-cores), 128GB DRAM
  – Mellanox InfiniBand EDR interconnects (100 Gbps)

• Benchmark
  – “memslap” MGet benchmark, 28 streaming client threads per node, employ FB’s Memcached workload generator to mimic KV store’s skewed data access pattern

• Configuration
  – A single KV store server running full-subscription
  – HT size of 32MB with 90% load factor: $2^{20}$ buckets with 4slots/bucket for MemC3, $2^{24}$ hash buckets with 32-bit keys/payloads Cuckoo-Ver (AVX-512) and SCOR-KV
Performance with Varying KV Pair / MGet Sizes

- SCOR-KV can improve the overall MGet latency by about 2.6x for small KV pair sizes and about 12% for larger sizes (a. MGet size ‘n=32’)
- Improve the overall MGet latencies from 1.6x to 3.6x as the MGet size ‘n’ increases, server-side throughput up by about 8.56x (b. KV pair size: (16B, 64B))

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

• SCOR-KV improve server get throughput by 3.74x–8.69x
  – +PostOptm: improve 5.09x overall, +HybridHT: improve 2.6x/1.3x for KV Pair Search, +PreOptm: reduce 90% preprocessing server-side

• More beneficial for skewed KV workloads than uniform pattern

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

- YCSB benchmark: Benchmarking Cloud Serving Systems
- Mixed KV store workloads (95:5 read/write mix with MGet 16 keys and Set operations)
- 2.2x improvement for MGet over non-SIMD MemC3 and SIMD based Cuckoo-Ver
- Maintain the RDMA KV store’s Set performance, no overhead of hybrid and partitioned HT layout
Conclusion

• Propose SCOR-KV, that optimally exploits ‘RDMA+SIMD’ to accelerate read heavy MGet operations in KV stores on emerging multi-core CPU architectures

• Propose 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

• SCOR-KV can achieve up to 3.7-8.6x improvement in server-side Get throughput

• SCOR-KV improves end-to-end MGet latencies for read-heavy YCSB workloads by about 2.2x over RDMA-based Memcached server with non-SIMD cache-optimized MemC3 backend