KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC

Bojie Li$^{1,2}$ Zhenyuan Ruan$^{1,3}$ Wencong Xiao$^{1,4}$ Yuanwei Lu$^{1,2}$ Yongqiang Xiong$^1$ Andrew Putnam$^1$ Enhong Chen$^2$
Lintao Zhang$^1$

$^1$Microsoft Research  $^2$USTC  $^3$UCLA  $^4$Beihang University
Key-Value Store in Data Centers

Web Cache

Key  Value
User ID  Recent Tweets
SQL  Query result
Session  Browser cookies
Key-Value Store in Data Centers

Web Cache

- **Key**
  - User ID
  - SQL
  - Session

- **Value**
  - Recent Tweets
  - Query result
  - Browser cookies

Shared Data Structure

- **Key**
  - Graph node
  - Feature
  - Shard

- **Value**
  - List of edges
  - Weight
  - Sequence number

More demanding workload
Design Goals

1. High throughput
2. Low tail latency
3. Write intensive
4. Vector operations
5. Atomic operations

More demanding workload
Key-Value Store Architectures

Software (Kernel TCP/IP)

Bottleneck: Network stack in OS
(~300 Kops per core)
Key-Value Store Architectures

Software (Kernel TCP/IP)

Bottleneck: Network stack in OS
(~300 Kops per core)

e.g. DPDK, mtcp, libvma, two-sided RDMA

Bottlenecks: CPU random memory access
and KV operation computation
(~5 Mops per core)
**Key-Value Store Architectures**

**Software (Kernel TCP/IP)**

Bottleneck: Network stack in OS

(~300 Kops per core)

**Software (Kernel Bypass)**

E.g. DPDK, mtcp, libvma, two-sided RDMA

Bottlenecks: CPU random memory access and KV operation computation

(~5 Mops per core)

**One-sided RDMA**

Communication overhead: multiple round-trips per KV operation (fetch index, data)

Synchronization overhead: write operations
Key-Value Store Architectures

- **Software (Kernel TCP/IP)**
- **Software (Kernel Bypass)**
- **One-sided RDMA**

**KV-Direct FPGA + NIC**

Offload KV processing on CPU to Programmable NIC
A Commodity Server with SmartNIC
Performance Challenges

Reconfigurable NIC

Host DRAM (256 GB)

PCIe Gen3 x16 DMA

FPGA

NIC

40 GbE ToR switch

On-board DRAM (4 GB)

CPU
Performance Challenges

Reconfigurable NIC

On-board DRAM (4 GB)

NIC

FPGA

Header overhead and limited parallelism: Be frugal on memory accesses

PCIe Gen3 x16 DMA

13 GB/s

120 Mops

Host DRAM (256 GB)

40 GbE

ToR switch

CPU
Performance Challenges

- Reconfigurable NIC
- On-board DRAM (4 GB)
- FPGA
- PCIe Gen3 x16 DMA
- 1us delay
- 120 Mops
- Atomic operations have dependency: PCIe latency hiding

- Host DRAM (256 GB)
- 40 GbE
- ToR switch
- CPU
Performance Challenges

Reconfigurable NIC

On-board DRAM (4 GB)

0.2us delay
100 Mops

PCle Gen3 x16 DMA

1us delay
120 Mops

Host DRAM (256 GB)

CPU

NIC

40 GbE

ToR switch

Load dispatch
Performance Challenges

Reconfigurable NIC

On-board DRAM (4 GB)
- 0.2us delay
- 100 Mops

FPGA
- 1us delay
- 120 Mops

NIC

40 GbE
- 60 Mpps

ToR switch

Client-side batching
Vector-type operations

PCIe Gen3 x16 DMA

Host DRAM (256 GB)

CPU

0.2us delay 100 Mops

1us delay 120 Mops
Design Principles

1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server
Be Frugal on Memory Accesses

Hash table: minimal access per GET & PUT
Cuckoo hashing: constant GET, sacrifice PUT.
Log-structured memory: fast PUT, sacrifice GET.
Our choice: Bucketized chaining: Close to 1 per GET, 2 per PUT

(a) 10B GET.
(b) 10B PUT.
Be frugal on memory accesses

Slab allocator for variable-sized KVs

- 512B
- 256B
- 128B
- 64B
- 32B
Be frugal on memory accesses

Random memory access to merge free slabs

512B

256B

128B

64B

32B

New free slab  Adjacent slab to check
Be Frugal on Memory Accesses

Solution: lazy slab merging on CPU

Worst case: 0.07 amortized DMA operations per (de)allocation.
Design Principles

1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server
Memory dependency

Client: $K1 += a$

NIC: $K1 += b$

Mem: $K1$ unlocked

PCle RTT: $\sim1$ us
Reservation Station

Cache most frequently accessed KVs

Client → NIC

K1 += a
K1 += b
K1 cached
Execute in cache

NIC → Mem
Pipeline stall

K1 += a
K1 += b
K2 += c

Stalled due to K1
Out-of-order execution

Client

NIC

Mem

K1 += a
K1 += b
K2 += c

OOO execution

Reordered response
Out-of-order Execution

(a) Atomics.
(b) Long-tail workload.

Throughput: 191x single-key atomics, 20x long-tail workload

We hope future RDMA NICs could adopt out-of-order execution for atomic operations!
Design Principles

1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server
How to Use On-board DRAM?

Reconfigurable NIC

NIC

FPGA

On-board DRAM (4 GB)

100 Mops

40 GbE

ToR switch

PCle Gen3 x16 DMA

120 Mops

Host DRAM (256 GB)

CPU
How to Use On-board DRAM?

Cache?

- Up to 100 Mops
  - On-board DRAM (100 Mops)
    - Host Memory (120 Mops)

Load balance?

- Up to 122 Mops
  - Dispatcher
  - 25 Mops/GB
  - 0.5 Mops/GB
  - On-board DRAM (4 GB, 100 Mops)
  - Host Memory (256 GB, 120 Mops)
Load Dispatch = Cache + Load Balance

Make full use of both on-board and host DRAM by adjusting the cache-able portion
Design Principles

1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server
Vector-Type Operations

Example: key[0] += a, key[1] += b

**Approach 1: Each element as a key**

Client

Server

atomic_add(key_0, a)
atomic_add(key_1, b)

**Approach 2: Compute at client**

Client

Server

get(key)
put(key, new_vector)
Vector-Type Operations

Example: key[0] += a, key[1] += b

Approach 1: Each element as a key

Client -> Server

atomic_add(key_0, a)
atomic_add(key_1, b)

Approach 2: Compute at client

Client -> Server

Compute new vector

Client

get(key)

Server

put(key, new_vector)

Our approach: Vector operations

Client -> Server

vector_atomic_add(key, [a,b])

Actually the “atomic add” function can be user-defined.
Client-side Network Batching

Amortize packet header overhead

(a) Throughput.

3.6x throughput
(50 Mops -> 180 Mops)

(b) Latency.

+1 us latency
(2.2 us -> 3.3 us)
KV-Direct System Performance

(a) Uniform.
KV-Direct System Performance

(a) Uniform.

(b) Long-tail.
KV-Direct System Performance

- Min latency
- Avg latency
- Max latency

**Batching** vs. **Non-batching**
Little Impact on CPU Performance

<table>
<thead>
<tr>
<th>CPU performance</th>
<th>Random memory access</th>
<th>Sequential memory access</th>
</tr>
</thead>
<tbody>
<tr>
<td>KV-Direct NIC</td>
<td>14.4 GB/s</td>
<td>60.3 GB/s</td>
</tr>
<tr>
<td>idle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KV-Direct NIC</td>
<td>14.4 GB/s</td>
<td>55.8 GB/s</td>
</tr>
<tr>
<td>busy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PCIe Gen3 x16 DMA
13 GB/s
120 Mops

Run other tasks on CPU

Host DRAM
(64 GB KVS, 192 GB other)

100 GB/s
600 Mops
Scalability with Multiple NICs

The graph shows the throughput (MOPS) in relation to the number of NICs. The throughput increases linearly with the number of NICs for both GET and PUT operations.
Scalability with Multiple NICs

1.22 billion KV op/s
357 watts power
## KVS Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>Tput (Mops) (GET / PUT)</th>
<th>Power (Kops/W)</th>
<th>Comment</th>
<th>Latency (us) (GET / PUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>1.5 / 1.5</td>
<td>5 / 5</td>
<td>TCP/IP</td>
<td>50 / 50</td>
</tr>
<tr>
<td>MemC3</td>
<td>4.3 / 4.3</td>
<td>14 / 14</td>
<td>TCP/IP</td>
<td>50 / 50</td>
</tr>
<tr>
<td>RAMCloud</td>
<td>6 / 1</td>
<td>20 / 3.3</td>
<td>Kernel bypass</td>
<td>5 / 14</td>
</tr>
<tr>
<td>MICA (12 NICs)</td>
<td>137 / 135</td>
<td>342 / 337</td>
<td>Kernel bypass</td>
<td>81 / 81</td>
</tr>
<tr>
<td>FARM</td>
<td>6 / 3</td>
<td>30 (261) / 15</td>
<td>One-side RDMA</td>
<td>4.5 / 10</td>
</tr>
<tr>
<td>DrTM-KV</td>
<td>115 / 14</td>
<td>500 (3972) / 60</td>
<td>One-side RDMA</td>
<td>3.4 / 6.3</td>
</tr>
<tr>
<td>HERD</td>
<td>35 / 25</td>
<td>490 / 300</td>
<td>Two-side RDMA</td>
<td>4 / 4</td>
</tr>
<tr>
<td>FPGA-Xilinx</td>
<td>14 / 14</td>
<td>106 / 106</td>
<td>FPGA</td>
<td>3.5 / 4.5</td>
</tr>
<tr>
<td>Mega-KV</td>
<td>166 / 80</td>
<td>330 / 160</td>
<td>GPU</td>
<td>280 / 280</td>
</tr>
<tr>
<td>KV-Direct (1 NIC)</td>
<td>180 / 114</td>
<td>1487 (5454) / 942 (3454)</td>
<td>Programmable NIC</td>
<td>4.3 / 5.4</td>
</tr>
<tr>
<td>KV-Direct (10 NICs)</td>
<td>1220 / 610</td>
<td>3417 (4518) / 1708 (2259)</td>
<td>Programmable NIC</td>
<td>4.3 / 5.4</td>
</tr>
</tbody>
</table>

* Number in parenthesis indicates power efficiency based on power consumption of NIC only, for server-bypass systems.
## KVS Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>Tput (Mops) (GET / PUT)</th>
<th>Power (Kops/W)</th>
<th>Comment</th>
<th>Latency (us) (GET / PUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>1.5 / 1.5</td>
<td>5 / 5</td>
<td>TCP/IP</td>
<td>50 / 50</td>
</tr>
<tr>
<td>MemC3</td>
<td>4.3 / 4.3</td>
<td>14 / 14</td>
<td>TCP/IP</td>
<td>50 / 50</td>
</tr>
<tr>
<td>RAMCloud</td>
<td>6 / 1</td>
<td>20 / 3.3</td>
<td>Kernel bypass</td>
<td>5 / 14</td>
</tr>
<tr>
<td>MICA (12 NICs)</td>
<td>137 / 135</td>
<td>342 / 337</td>
<td>Kernel bypass</td>
<td>81 / 81</td>
</tr>
<tr>
<td>FARM</td>
<td>6 / 3</td>
<td>30 (261) / 15</td>
<td>One-side RDMA</td>
<td>4.5 / 10</td>
</tr>
<tr>
<td>DrTM-KV</td>
<td>115 / 14</td>
<td>500 (3972) / 60</td>
<td>One-side RDMA</td>
<td>3.4 / 6.3</td>
</tr>
<tr>
<td>HERD</td>
<td>35 / 25</td>
<td>490 / 300</td>
<td>Two-side RDMA</td>
<td>4 / 4</td>
</tr>
<tr>
<td>FPGA-Xilinx</td>
<td>14 / 14</td>
<td>106 / 106</td>
<td>FPGA</td>
<td>3.5 / 4.5</td>
</tr>
<tr>
<td>Mega-KV</td>
<td>166 / 80</td>
<td>330 / 160</td>
<td>GPU</td>
<td>280 / 280</td>
</tr>
<tr>
<td>KV-Direct (1 NIC)</td>
<td>180 / 114</td>
<td>1487 (5454) / 942 (3454)</td>
<td>Programmable NIC</td>
<td>4.3 / 5.4</td>
</tr>
<tr>
<td>KV-Direct (10 NICs)</td>
<td>1220 / 610</td>
<td>3417 (4518) / 1708 (2259)</td>
<td>Programmable NIC</td>
<td>4.3 / 5.4</td>
</tr>
</tbody>
</table>

* Number in parenthesis indicates power efficiency based on power consumption of NIC only, for server-bypass systems.
## KVS Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>Tput (Mops) (GET / PUT)</th>
<th>Power (Kops/W)</th>
<th>Comment</th>
<th>Latency (us) (GET / PUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>1.5 / 1.5</td>
<td>5 / 5</td>
<td>TCP/IP</td>
<td>50 / 50</td>
</tr>
<tr>
<td>MemC3</td>
<td>4.3 / 4.3</td>
<td>14 / 14</td>
<td>TCP/IP</td>
<td>50 / 50</td>
</tr>
<tr>
<td>RAMCloud</td>
<td>6 / 1</td>
<td>20 / 3.3</td>
<td>Kernel bypass</td>
<td>5 / 14</td>
</tr>
<tr>
<td>MICA (12 NICs)</td>
<td>137 / 135</td>
<td>342 / 337</td>
<td>Kernel bypass</td>
<td>81 / 81</td>
</tr>
<tr>
<td>FARM</td>
<td>6 / 3</td>
<td>30 (261) / 15</td>
<td>One-side RDMA</td>
<td>4.5 / 10</td>
</tr>
<tr>
<td>DrTM-KV</td>
<td>115 / 14</td>
<td>500 (3972) / 60</td>
<td>One-side RDMA</td>
<td>3.4 / 6.3</td>
</tr>
<tr>
<td>HERD</td>
<td>35 / 25</td>
<td>490 / 300</td>
<td>Two-side RDMA</td>
<td>4 / 4</td>
</tr>
<tr>
<td>FPGA-Xilinx</td>
<td>14 / 14</td>
<td>106 / 106</td>
<td>FPGA</td>
<td>3.5 / 4.5</td>
</tr>
<tr>
<td>Mega-KV</td>
<td>166 / 80</td>
<td>330 / 160</td>
<td>GPU</td>
<td>280 / 280</td>
</tr>
<tr>
<td>KV-Direct (1 NIC)</td>
<td>180 / 114</td>
<td>1487 (5454) / 942 (3454)</td>
<td>Programmable NIC</td>
<td>4.3 / 5.4</td>
</tr>
<tr>
<td>KV-Direct (10 NICs)</td>
<td>1220 / 610</td>
<td>3417 (4518) / 1708 (2259)</td>
<td>Programmable NIC</td>
<td>4.3 / 5.4</td>
</tr>
</tbody>
</table>

* Number in parenthesis indicates power efficiency based on power consumption of NIC only, for server-bypass systems.
Conclusion

Accelerate systems with Programmable NIC
Thank you! Questions!

Go beyond the memory wall & reach a fully programmable world
Backup
What application?

Back-of-envelope calculations show potential performance gains when KV-Direct is applied in end-to-end applications. In PageRank, because each edge traversal can be implemented with one KV operation, KV-Direct supports 1.2 billion TEPS on a server with 10 programmable NICs. In comparison, GRAM (Ming Wu on SoCC’15) supports 250M TEPS per server, bounded by interleaved computation and random memory access.
Are the optimizations general?

The discussion section of the paper discusses NIC hardware with different capacity. First, the goal of KV-Direct is to leverage existing hardware in data centers instead of designing a specialized hardware to achieve maximal KVS performance. Even if future NICs have faster or larger on-board memory, under long-tail workload, our load dispatch design still shows performance gain. The hash table and slab allocator design is generally applicable to cases where we need to be frugal on memory accesses. The out-of-order execution engine can be applied to all kinds of applications in need of latency hiding.
Throughput is similar to state-of-art

With a single KV-Direct NIC, the throughput is equivalent to 20 to 30 CPU cores. These CPU cores can run other CPU intensive or memory intensive workload, because the host memory bandwidth is much larger than the PCIe bandwidth of a single KV-Direct NIC. So we basically save tens of CPU cores per programmable NIC. With ten programmable NICs, the throughput can grow almost linearly.
Consistency among multiple NICs

Each NIC behaves as if it is an independent KV-Direct server. Each NIC serves a disjoint partition of key space and reserves a disjoint region of host memory. The clients distribute load to each NIC according to the hash of keys, similar to the design of other distributed key-value stores. Surely, the multiple NICs suffer load imbalance problem in long-tail workload, but the load imbalance is not significant with a small number of partitions. The NetCache system in this session can also mitigate the load imbalance problem.
What is the non-batch throughput?

We use client-side batching because our programmable NIC has limited network bandwidth. The network bandwidth is only 5 GB/s, while the DRAM and PCIe bandwidth are both above 10 GB/s. So we batch multiple KV operations in a single network packet to amortize the packet header overhead. If we have a higher bandwidth network, we will no longer need network batching.