Hardware Protocols and Key-Value Storage

David G. Andersen, Michael Kaminsky and the folks who really did the work: Hyeontaek Lim, Anuj Kalia, Dong Zhou
Throughput-Latency on Ethernet

Average latency (μs)

Original Memcached using standard socket I/O; both use UDP

200x+ throughput
Computational Efficiency

Memory Efficiency

Algorithmic Optimization

Architectural Tailoring
In-memory KV stores

Interface: GET, PUT

Requirements:
- Low latency
- High request rate
(See Cuckoo Talk)
• How to get requests (packets) in and out?
• How to design & implement the index and datastore?
• In ways that work with *modern* hardware
  • Multicore, NUMA, 40gbps NICs, etc.
MICA
[NSDI’14]
HERD
[SIGCOMM’14]

Ethernet
Infiniband / RoCE

Intel DPDK
RDMA
MICA Approach

- **MICA**: Redesigning in-memory key-value storage
- Applies new SW architecture and data structures to general-purpose HW in a **holistic** way

![Diagram of server node with components and connections]

1. Parallel data access
2. Request direction
3. Key-value data structures (cache & store)
Modern CPUs have many cores (8, 15, ...)

How to exploit CPU parallelism **efficiently**?
Parallel Data Access Schemes

Concurrent Read Concurrent Write

- Good load distribution
- Limited CPU scalability (e.g., synchronization)
- Cross-NUMA latency

Exclusive Read Exclusive Write

- Good CPU scalability
- Potentially low performance under skewed workloads
In MICA, Exclusive Outperforms Concurrent

Throughput (Mops)

End-to-end performance with kernel bypass I/O
Request Direction

- Sending requests to appropriate CPU cores for better data access locality
- Exclusive access benefits from correct delivery
  - Each request must be sent to corresp. partition’s core
**Request Direction Schemes**

### Flow-based Affinity

**Server node**

- **Client**
- **NIC**
- **CPU**

**Classification using 5-tuple**

- **+ Good locality for flows** (e.g., HTTP over TCP)
- **- Suboptimal for small key-value processing**

### Object-based Affinity

**Server node**

- **Client**
- **NIC**
- **CPU**

**Classification depends on request content**

- **+ Good locality for key access**
- **- Client assist** or special HW support needed for efficiency
Crucial to Use NIC HW for Request Direction

Throughput (Mops)

Using exclusive access for parallel data access
Plus some cool data structures inside

(see Lim et al., NSDI 2014)

Result:
The fastest network-based key-value server that we know of.

2 socket Xeon server can nearly saturate 80Gbps of Ethernet (8x10Gbps).
Protocol changes to let NICs direct requests to the right core

Careful attention to NUMA and locality

OS & Stack bypass to eliminate overhead
RDMA

*Remote Direct Memory Access:* A network feature that allows direct access to the memory of a remote computer.
HERD

1. Improved understanding of RDMA through micro-benchmarking

2. High-performance key-value system:
   - Throughput: 26 Mops \( (2X \text{ higher than others}) \)
   - Latency: 5 µs \( (2X \text{ lower than others}) \)
RDMA intro

Features:

• Ultra-low latency: 1 µs RTT

• Zero copy + CPU bypass

Providers:

InfiniBand, RoCE,…
# RDMA in the datacenter

## 48 port 10 GbE switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>RDMA</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mellanox SX1012</td>
<td>YES</td>
<td>$5,900</td>
</tr>
<tr>
<td>Cisco 5548UP</td>
<td>NO</td>
<td>$8,180</td>
</tr>
<tr>
<td>Juniper EX5440</td>
<td>NO</td>
<td>$7,480</td>
</tr>
</tbody>
</table>
RDMA basics

Verbs

RDMA read:
READ(local_buf, size, remote_addr)

RDMA write:
WRITE(local_buf, size, remote_addr)
Life of a WRITE

1: Request descriptor, PIO
2: Payload, DMA read
3: RDMA write request
6: Completion, DMA write

4: Payload, DMA write
5: RDMA ACK
Recent systems

Pilaf [ATC 2013]

FaRM-KV [NSDI 2014]: an example usage of FaRM

Approach: RDMA reads to access remote data structures

Reason: the allure of CPU bypass
The price of CPU bypass

Key-Value stores have an inherent level of indirection.

An index maps a keys to address. Values are stored separately.

At least 2 RDMA reads required:
\[ \geq 1 \text{ to fetch address} \]
\[ 1 \text{ to fetch value} \]

Not true if value is in index
The price of CPU bypass
The price of CPU bypass

Client

READ #1 (fetch pointer)

Server
The price of CPU bypass
Our approach

<table>
<thead>
<tr>
<th>Goal</th>
<th>Main ideas</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1: Use a single round trip</td>
<td>Request-reply with server CPU involvement + WRITEs faster than READs</td>
</tr>
<tr>
<td>#2. Increase throughput</td>
<td>Low level verbs optimizations</td>
</tr>
<tr>
<td>#3. Improve scalability</td>
<td>Use datagram transport</td>
</tr>
</tbody>
</table>
#1: Use a single round trip
#1: Use a single round trip

<table>
<thead>
<tr>
<th>Operation</th>
<th>Round Trips</th>
<th>Operations at server’s RNIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ-based GET</td>
<td>2+</td>
<td>2+ RDMA reads</td>
</tr>
<tr>
<td>HERD GET</td>
<td>1</td>
<td>2 RDMA writes</td>
</tr>
</tbody>
</table>

- **Lower latency**
- **High throughput**
RDMA WRITEs faster than READs

Setup: Apt Cluster
192 nodes, 56 Gbps IB
High-speed request-reply

Request-reply throughput:

Setup: one-to-one client-server communication

32 byte payloads

- 1 READ
- 2 WRITEs
- 2 READs

Throughput (Mops)
Step 2: Optimize the primitives (details in paper)

Key takeaway: *Naive* uses of other RDMA primitives are slow
But there exist *optimized* uses that are really fast
Evaluation

HERD = Request-Reply + MICA [NSDI 2014]

Compare against emulated versions of Pilaf and FaRM-KV

- No datastore
- Focus on maximum performance achievable
Latency vs throughput

48 byte items, GET intensive workload

Throughput (Mops)

Latency (microseconds)

HERD

95th percentile

5th percentile

Low load, 3.4 µs

26 Mops, 5 µs
Latency vs throughput

48 byte items, GET intensive workload

Latency (microseconds)

Throughput (Mops)

Emulated Pilaf
Emulated FaRM-KV
HERD

95th percentile
5th percentile

12 Mops, 8 µs
Low load, 3.4 µs
26 Mops, 5 µs
Throughput comparison

16 byte keys, 95% GET workload

- Emulated Pilaf
- Emulated FaRM-KV
- HERD

Throughput (Mops)

Throughput comparison

2X higher
Computational Efficiency

Memory Efficiency

MICA and HERD key-value stores

Algorithmic Optimization

Architectural Tailoring

Good Data Structures

Protocols that are locality-friendly

Optimize for the right things (few RTTs!)

This is hard. Can we (semi) automate?