Replicating Persistent Memory Key-Value Stores with Efficient RDMA Abstraction

Qing Wang, Youyou Lu, Jing Wang, Jiwu Shu

Tsinghua University
Replicated distributed key-value stores (KVSs) support many apps

- Durability ⇒ Storage devices (HDD, SSD)
- High availability ⇒ Data replication
Replicated distributed key-value stores (KVSs) support many apps

- Durability $\Rightarrow$ Storage devices (HDD, SSD)
- High availability $\Rightarrow$ Data replication

How to optimize the latency of replicated KVSs by leveraging modern hardware?
Step 1: Persistent Memory

Using persistent memory (PM) for storage

- Byte-addressable via load/store instructions
- Low latency (~100ns for small I/O)
- High-bandwidth (2GB/s write and 6GB/s read per DIMM)
Step 2: RDMA Network

Using RDMA for network

- Bypass OS kernel: threads interact directly with NICs
- Hardware offloading: e.g., reliability (RC mode), packetization
- High performance: ~2μs RTT, 100-400Gbps
Step 3: One-sided Replication

Using one-sided WRITE for replication

- RDMA provides one-sided RDMA WRITE/READ, bypassing remote CPUs
- Primary pushes replicated objects to backups’ PM via RDMA WRITE
- Eliminate *RPC queueing and CPU execution* of backups in the critical path
- E.g., Mu (OSDI’20, DRAM-based)
However, RDMA WRITE induces write amplification

Each server holds a number of backup logs and receives small RDMA WRITE

A number of backup logs caused by sharding:
Each server acts as backups for many shards

Allocates lots of backup logs, each accommodating RDMA WRITE from a remote thread (primaries)

- FaRM has thousands of backup logs per server
- \#log = (#server - 1) * #(threads per server)

<table>
<thead>
<tr>
<th>Server A</th>
<th>Server B</th>
<th>Server A</th>
<th>Server C</th>
<th>Server D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>Thread 1</td>
<td>Thread 2</td>
<td>Thread 5</td>
<td>Thread 4</td>
</tr>
</tbody>
</table>

RDMA WRITE

backup logs
However, RDMA WRITE induces write amplification

Each server holds a number of backup logs and receives small RDMA WRITE

A number of backup logs caused by sharding:
Each server acts as backups for many shards
Allocate lots of backup logs, each accommodating RDMA WRITE from a remote thread (primaries)
- FaRM has thousands of backup logs per server
- \( \#log = (\#server - 1) \times \#(\text{threads per server}) \)

Small RDMA WRITE caused by small objects:
Small objects are prevalent
- In Meta’s largest KVS ZippyDB, the average object size is 90.8B (FAST’20)
- At Twitter, the average tweet is less than 33 characters (Kangaroo, SOSP’21)

<table>
<thead>
<tr>
<th>Server A</th>
<th>Server B</th>
<th>Server A</th>
<th>Server C</th>
<th>Server D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>Thread 1</td>
<td>Thread 2</td>
<td>Thread 5</td>
<td>Thread 4</td>
</tr>
</tbody>
</table>

RDMA WRITE

backup logs

………
However, RDMA WRITE induces write amplification

PM devices have byte interface with a block-level internal access granularity

- Optane PM: 256B XPL ine; CXL-SSD: Flash Page
- Devices combine adjacent small writes to control device-level write amplification (DLWA)
- Implication: PM devices prefer large writes or sequential small writes
However, RDMA WRITE induces write amplification

PM devices have byte interface with a block-level internal access granularity

- Optane PM: 256B XPLIne; CXL-SSD: Flash Page
- Devices combine adjacent small writes to control device-level write amplification (DLWA)
- Implication: PM devices prefer large writes or sequential small writes

One-sided Replication in KVS: random small writes

(In the PM server, 18 cores perform local sequential PM writes, DDIO disabled)
How to mitigate device-level write amplification?

Using software batching?

- Accumulate small writes within a timeout, then emit the batched writes to remote backup logs via one RDMA WRITE.
- Problem:
  - Induce extra latency, remove benefits of extremely low-latency HW (PM、RDMA)
  - GET operations and sharding reduce the opportunity of batching.
How to mitigate device-level write amplification?

Using software batching?

- Accumulate small writes within a timeout, then emit the batched writes to remote backup logs via one RDMA WRITE
- Problem:
  - **Induce extra latency**, remove benefits of extremely low-latency HW (PM, RDMA)
  - GET operations and sharding reduce the opportunity of batching

Can we mitigate DLWA without inducing any software delay?
Our Idea – New RDMA abstraction: Rowan

Rowan (remote write aggregation):
- Receiver-side NICs land remote writes to PM **sequentially**, and return ACKs
- Receiver-side NICs decide destination addresses
  - Do not need per-remote-thread log area for RDMA WRITE

![Diagram of Rowan Abstraction (Receiver-side)]
Our Idea – New RDMA abstraction: Rowan

Rowan (remote write aggregation):

- Receiver-side NICs land remote writes to PM **sequentially**, and return ACKs
- Receiver-side NICs decide destination addresses
  - Do not need per-remote-thread log area for RDMA WRITE
- Benefits
  - Low latency: one-sided, no delay at sender/receiver
  - Low DLWA: sequential small writes
  - High throughput: NIC ASIC executes data path

![Diagram showing Rowan Abstraction (Receiver-side) with increasing address order and writes from different threads]
Our Idea – New RDMA abstraction: Rowan

Rowan (remote write aggregation):
- Receiver-side NICs land remote writes to PM sequentially, and return ACKs
- Receiver-side NICs decide destination addresses
  - Do not need per-remote-thread log area for RDMA WRITE
- Benefits
  - Low latency: one-sided, no delay at sender/receiver
  - Low DLWA: sequential small writes
  - High throughput: NIC ASIC executes data path

Simple RDMA abstraction, but how to implement it using commodity RDMA NICs?
Observations

Observation 1:
- RDMA SEND in RC mode is **one-sided on the data path**
  - Control path: receiver’s CPU prepares receive buffers via RDMA RECV
  - Data path: receiver’s NIC performs **all tasks**: DMA data, and return **hardware ACKs**

Observation 2:
- In a receive queue (RQ), receive buffers are consumed in order
  - the receiver-side NIC pops the first buffer in the associated RQ and lands data to it
Rowan – Basic Architecture

Rowan Basic Architecture

- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
Rowan – Basic Architecture

Rowan Basic Architecture

- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ
Rowan – Basic Architecture

Rowan Basic Architecture

- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ
- Data path: NIC
  - 1) Pops the first buffer in SRQ and DMAs data to it
  - 2) Returns an ACK and generates a CQ entry

Diagram:
- Senders
- RDMA NIC
- pop and DMA
- generate CE
- addr: 0x040
- addr: 0x080
- addr: 0x120
- poll
- push 64B PM bufs
- Shared RQ
- Receiver
- Control Thread
Rowan – Basic Architecture

Rowan Basic Architecture

- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ
- Data path: NIC
  - 1) Pops the first buffer in SRQ and DMAs data to it
  - 2) Returns an ACK and generates a CQ entry

writes from different senders can be combined into the same PM internal block
Rowan – Basic Architecture

Rowan Basic Architecture

- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ
- Data path: NIC
  - 1) Pops the first buffer in SRQ and DMAs data to it
  - 2) Returns an ACK and generates a CQ entry

Writes from different senders can be combined into the same PM internal block
Rowan – Handling Variable-sized Writes

Leveraging Multi-Packet (MP) RQ
- A new type of RQ, supported by CX-4/5/6 NICs
- Each receive buffer can accommodate **multiple** SEND
- Define a stride (e.g., 64B in the right figure)
  - Each message has a stride-aligned start address
Rowan – Handling Variable-sized Writes

Leveraging Multi-Packet (MP) RQ

- A new type of RQ, supported by CX-4/5/6 NICs
- Each receive buffer can accommodate multiple SEND
- Define a stride (e.g., 64B in the right figure)
  - Each message has a stride-aligned start address

Rowan supports variable-sized writes, while combining small writes to mitigate DLWA
Avoid control thread become bottleneck

- Data path: > 50Mops/s
- Two tasks of control thread:
  - 1. Push PM buffers to MP SRQ
  - 2. Poll CQ (RDMA RECV cannot be unsignaled)
Avoid control thread become bottleneck

- Data path: > 50Mops/s
- Two tasks of control thread:
  - 1. Push PM buffers to MP SRQ
  - 2. Poll CQ (RDMA RECV cannot be unsignaled)
- Low overhead RDMA RECV
  - Large recv buffer (e.g., 4MB) using MP features
  - Post a batch of RDMA RECV at a time
Avoid control thread become bottleneck

- Data path: > 50Mops/s
- Two tasks of control thread:
  - Push PM buffers to MP SRQ
  - Poll CQ (RDMA RECV cannot be unsighaled)
- Low overhead RDMA RECV
  - Large recv buffer (e.g., 4MB) using MP features
  - Post a batch of RDMA RECV at a time
- Eliminate CQ polling
  - Like eRPC@NSDI’19
  - Ring-structure CQ and NIC can overwrite CQ entries
  - Flag: IBV_EXP_CQ_IGNORE_OVERRUN
Rowan-KV

- Log-structured data layout
- Primary-backup replication

<table>
<thead>
<tr>
<th>Shard ID</th>
<th>Primary</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>1</td>
<td>{2,3}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Server 1

Server 2

Configuration Manager
Rowan-KV

- Log-structured data layout
- Primary-backup replication
- Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
Rowan-KV

- Log-structured data layout
- Primary-backup replication
- Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- Workflow of a PUT operation
  - Client sends an RPC to the primary (P)

<table>
<thead>
<tr>
<th>Shard ID</th>
<th>Primary</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>1</td>
<td>{2,3}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Configuration Manager

Server 1

Server 2

DRAM

Hash Index

Backup log

Primary log

Server 2
Log-structured data layout

Primary-backup replication

Three components per server
- A single backup log managed by one Rowan instance
- Per-thread primary logs
- Per-shard DRAM hash indexes

Workflow of a PUT operation
- 1 Client sends an RPC to the primary (P)
- 2 P appends an entry E to the local primary log

---

Client

Configuration Manager

<table>
<thead>
<tr>
<th>Shard ID</th>
<th>Primary</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>1</td>
<td>{2,3}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Workflow of a PUT operation
- Client sends an RPC to the primary (P)
- P appends an entry E to the local primary log
Log-structured data layout

Primary-backup replication

Three components per server
- A single backup log managed by one Rowan instance
- Per-thread primary logs
- Per-shard DRAM hash indexes

Workflow of a PUT operation
- 1. Client sends an RPC to the primary (P)
- 2. P appends an entry E to the local primary log
- 3. P writes E to backup logs of all backups via Rowan

<table>
<thead>
<tr>
<th>Shard ID</th>
<th>Primary</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>1</td>
<td>{2,3}</td>
</tr>
</tbody>
</table>

Server 1
- Configuration Manager
- DRAM
- Hash Index
- Backup log
- Primary log

Server 2
- DRAM
- Hash Index
- Backup log
- Primary log

Server 2
Rowan-KV

- Log-structured data layout
- Primary-backup replication
- Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- Workflow of a PUT operation
  - 1 Client sends an RPC to the primary (P)
  - 2 P appends an entry E to the local primary log
  - 3 P writes E to backup logs of all backups via Rowan
  - 4 P waits for hardware ACKs from backups’ NICs
Rowan-KV

- **Log-structured data layout**
- **Primary-backup replication**
- **Three components per server**
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- **Workflow of a PUT operation**
  - 1 Client sends an RPC to the primary (P)
  - 2 P appends an entry E to the local primary log
  - 3 P writes E to backup logs of all backups via Rowan
  - 4 P waits for hardware ACKs from backups’ NICs
  - 5 P updates index, pointing to E in primary log

**Configuration Manager**

<table>
<thead>
<tr>
<th>Shard ID</th>
<th>Primary</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>1</td>
<td>{2,3}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Server 1**

**Server 2**
Rowan-KV

- Log-structured data layout
- Primary-backup replication
- Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- Workflow of a PUT operation
  - 1 Client sends an RPC to the primary (P)
  - 2 P appends an entry E to the local primary log
  - 3 P writes E to backup logs of all backups via Rowan
  - 4 P waits for hardware ACKs from backups’ NICs
  - 5 P updates index, pointing to E in primary log
  - 6 P returns a response
Rowan-KV

- Log-structured data layout
- Primary-backup replication
- Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- Workflow of a PUT operation
  1. Client sends an RPC to the primary \( P \)
  2. \( P \) appends an entry \( E \) to the local primary log
  3. \( P \) writes \( E \) to backup logs of all backups via Rowan
  4. \( P \) waits for hardware ACKs from backups’ NICs
  5. \( P \) updates index, pointing to \( E \) in primary log
  6. \( P \) returns a response

1) Low latency: One-sided replication
2) Low DLWA: Log-structured & Rowan merges replication writes into a single backup log
More Design Details : Check Our Paper

Diet and Garbage Collection
- Reserve dedicated threads, RAMCloud-style GC

Failover
- FaRM's reconfiguration-style approach

Dynamic Redsharding
- Shard-level migration

Fast Remote Persistence with disabled DDIO
- Prefetching, Reducing PCIe Txns
Experimental Setup

Hardware Platform
- 6 machines as servers
- Intel Xeon Gold 6240M CPU (18 physical/36 logical cores)
- 3 \times 256GB Optane DIMMs (6GB/s writes, 18 GB/s reads)
- 100Gbps Mellanox ConnectX-5 NIC

Software Setting
- 24 cores for worker threads; 5/6/1 cores for digest/GC/control
- Replication factor: 3
- Each server holds 48 shards
- Disable DDIO and send 1B RDMA READ for persistency of RDMA WRITE or Rowan
Performance of Rowan

- Remote threads concurrently perform PM writes to a PM server via one Rowan instance
- In the PM server, 18 cores perform local sequential PM writes
Performance of Rowan

- Remote threads concurrently perform PM writes to a PM server via one Rowan instance
- In the PM server, 18 cores perform local sequential PM writes
Performance of Rowan

- Remote threads concurrently perform PM writes to a PM server via one Rowan instance
- In the PM server, 18 cores perform local sequential PM writes

Rowan can largely eliminate device-level write amplification (DLWA), and thus has higher (1.85X) throughput than RDMA WRITE
Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 servers, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes

![Graph showing latency and throughput comparison between Rowan and RPC](image)

(a) Throughout vs. Latency
Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 servers, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes

Under write-intensive workloads, compared with RPC and RDMA WRITE, Rowan boosts KVS’s throughput (by 1.2X and 1.4X) & reduces PUT latency (by 1.8X and 2.1X)
Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 severs, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes

![Graph showing PUT and GET latency for Rowan, RPC, RDMA write, and Batched RDMA write]

Software batching suffers the highest (50% more) PUT latency

(a) Throughout vs. Latency
Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 servers, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes

![Graphs and figures]

(a) Throughout vs. Latency

(b) DLWA
Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 severs, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes

![Graphs showing latency and throughput comparisons between Rowan, RPC, RDMA write, and batched RDMA write. Rowan largely eliminates DLWA, like RPC.](image)

Put Latency

GET Latency

(b) DLWA

Rowan

RPC

RDMA write

Batching

Request BW

Media BW
Performance Comparison with Other KVSs

- Clover [ATC’20]: one-sided READ/WRITE for replication
- HermesKV [ASPLOS’20]: broadcast replication protocol via RPC
- 6 Servers

![Chart](chart.png)

- Throughput (Mops/s)
- (a) ZippyDB Obj
- (b) 4KB Obj
Performance Comparison with Other KVSs

- Clover [ATC’20]: one-sided READ/WRITE for replication
- HermesKV [ASPLOS’20]: broadcast replication protocol via RPC
- 6 Servers

![Comparison Chart]

Under write-intensive workloads (i.e., 50% PUT), Rowan-KV outperforms Clover and HermesKV significantly (24.5X and 1.98X) when objects are small.
Performance Comparison with Other KVSs

- Clover [ATC’20]: one-sided READ/WRITE for replication
- HermesKV [ASPLOS’20]: broadcast replication protocol via RPC
- 6 Servers

Throughput (Mops/s)

(a) ZippyDB Obj

(b) 4KB Obj

Rowan-KV still has performance advantages when objects are large (e.g., 4KB)

Under write-intensive workloads (i.e., 50% PUT), Rowan-KV outperforms Clover and HermesKV significantly (24.5X and 1.98X) when objects are small.
Conclusion

- One-sided replication can achieve extreme low latency
  - Remove software latency of backups (RPC queueing, CPU execution) from the critical path
Conclusion

- One-sided replication can achieve extreme low latency
  - Remove software latency of backups (RPC queueing, CPU execution) from the critical path

- RDMA WRITE for replication induces severe device-level write amplification on PM
  - Pre-allocate many logs for remote threads
  - Small objects in workloads vs. block-level internal access granularity in PM devices
Conclusion

- One-sided replication can achieve extreme low latency
  - Remove software latency of backups (RPC queueing, CPU execution) from the critical path

- RDMA WRITE for replication induces severe device-level write amplification on PM
  - Pre-allocate many logs for remote threads
  - Small objects in workloads vs. block-level internal access granularity in PM devices

- We propose Rowan, a one-sided RDMA abstraction
  - Translating concurrent remote small writes into a single write stream
  - Rowan-based KVS achieves high performance, while largely eliminating DLWA
Conclusion

- One-sided replication can achieve extreme low latency
  - Remove software latency of backups (RPC queueing, CPU execution) from the critical path

- RDMA WRITE for replication induces severe device-level write amplification on PM
  - Pre-allocate many logs for remote threads
  - Small objects in workloads vs. block-level internal access granularity in PM devices

- We propose Rowan, a one-sided RDMA abstraction
  - Translating concurrent remote small writes into a single write stream
  - Rowan-based KVS achieves high performance, while largely eliminating DLWA

- Takeaway
  - For one-sided writes, receiver-side NIC is good at managing storage/memory devices
    1) It can coordinate requests from different senders
    2) It can allocate addresses according to features of storage/memory devices
Thanks & QA

Replicating Persistent Memory Key-Value Stores with Efficient RDMA Abstraction