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

Replicated distributed key-value stores (KVSs) support many apps
- Durability $\Rightarrow$ Storage devices (HDD, SSD)
- High availability $\Rightarrow$ Data replication
Replicated Distributed Key-Value Stores

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

- Durability ⇒ Storage devices (HDD、SSD)
- High availability ⇒ 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
- \( \#\text{log} = (\#\text{server} - 1) \times \#(\text{threads per server}) \)

Server A
Thread 1

Server A
Thread 2

Server C
Thread 5

Server D
Thread 4

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
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

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)

\[\text{Server A} \quad \text{Server B} \quad \text{Server C} \quad \text{Server D}\]
\[\text{Thread 1} \quad \text{Thread 2} \quad \text{Thread 5} \quad \text{Thread 4}\]

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 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
However, RDMA WRITE induces write amplification

PM devices have byte interface with a block-level internal access granularity
- Optane PM: 256B XPLLine; 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

Increasing address order

Writes from different threads

Rowan Abstraction (Receiver-side)
Our Idea – New RDMA abstraction: Rowan

Rowan (remote write aggregation):

- Receiver-side NICs land remote writes to PM \textit{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 PM and increasing address order]

Watches 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
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

How to handle it?
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 unsighaled)
- 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:
  - 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
- 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

| Shard ID | Primary | Backup |
|----------|---------|--------|
| A, B     | 1       | {2,3}  |
| ...      | ...     | ...    |

![Diagram of Rowan-KV system](image.png)
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

| Shard ID | Primary | Backup |
|----------|---------|--------|
| A, B     | 1       | {2,3}  |
| ...      | ...     | ...    |

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

Server 2
- Configuration Manager
- 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
  - Client sends an RPC to the primary (P)

Configuration Manager

| Shard ID | Primary | Backup |
|----------|---------|--------|
| A, B     | 1       | \{2,3\} |
| ...      | ...     | ...    |

Client

Server 1

DRAM

- Backup log
- Primary log

Hash Index

PM

- 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
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

| Shard ID | Primary | Backup |
|----------|---------|--------|
| A, B     | 1       | {2,3}  |
| ...      | ...     | ...    |

Configuration Manager

Server 1

- Hash Index
- DRAM
- PM
- Backup log
- Primary log

Server 2

- KV
- A
- B
- ...
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

| Shard ID | Primary | Backup |
|----------|---------|--------|
| A, B     | 1       | {2,3}  |
| ...      | ...     | ...    |

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

| Shard ID | Primary | Backup |
|----------|---------|--------|
| A, B     | 1       | {2,3}  |
| ...      | ...     | ...    |

![Diagram showing the workflow and configuration manager](image-url)
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

| Shard ID | Primary | Backup |
|----------|---------|--------|
| A, B     | 1       | {2,3}  |
| ...      | ...     | ...    |

Server 1

Server 2

Configuration Manager

Client

OK
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 Resharding
- Shard-level migration

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

Replicating Persistent Memory Key-Value Stores with Efficient RDMA Abstraction
Qing Wang, Youyou Lu, Jing Wang, and Jiwu Shu∗
Tsinghua University

Abstract
Combining persistent memory (PM) with RDMA is a promising approach to performing replicated distributed key-value stores (KVS) with low latency and high bandwidth. However, the replication mechanisms typically do not work well when applied to PM KVSs. 1) Using RDMA reduces network latency and request backlogs, increasing request latency. 2) Using non-PM RDMA requires many copies of small PM writes, leading to severe device-level write amplification (DLWA) on PM.

In this paper, we propose Reshard, an efficient RDMA abstraction to handle replication writes on PM KVSs; it aggregates continuous in-memory writes from different services, and loads those writes to PM in a sequential (thus low latency) manner. We realize Reshard using a client-side RDMA-based log-structured PM KVS using HPC and RDMA-Weite for replication. Reshard bundles together, e.g., 2-100B writes or several million small writes into one large sequential PM WRITE, thus significantly cutting the replication latency compared with RDMA-enabled replication.

Yet, in the context of PMs, RDMA-enabled replication approach does not work well: it induces severe device-level write amplification (DLWA) on PM. Specifically, a KVS is typically finely sharded for load balancing and fast recovery, generating a large number of small PM writes. Further, under heavy write loads, numerous concurrent replication writes from many services would exceed PM’s limited PM write bandwidth, shortens PM lifetime, and harms PM’s persistence efficiency. In this paper, we propose Reshard, an efficient RDMA abstraction to handle replication writes on PM KVSs. Reshard can aggregate millions of concurrent in-memory writes from different services, and loads those writes to PM in sequential, thus reducing the replication latency compared with RDMA-enabled replication.

In our experiments, with 128B RDMA packets, Reshard achieves 1.39×, 1.77×, and 2.11× speedups boosting throughput by 1.22×, latency by 1.77×, and 2.11×, respectively, while largely eliminating DLWA.

Introduction
Replicated distributed key-value stores (KVS) support many applications by providing scalability and high availability [7, 34]. The recent commercialization of persistent memory (PM) (e.g., Intel’s Optane DIMMs) enables local storage with extremely low latency (e.g., 10fs when persisting small data [7]). When building replicated distributed KVSs, we need to consider the multiple facets of network communication and request processing.

PM’s small memory capacity and high bandwidth can provide high performance, but low latency and high bandwidth may lead to severe device-level write amplification (DLWA) on PM [1, 2, 4]. DLWA waste PM’s persistence efficiency. In this paper, we propose Reshard, an efficient RDMA abstraction to handle replication writes on PM KVSs. Reshard can aggregate millions of concurrent in-memory writes from different services, and loads those writes to PM in sequential, thus reducing the replication latency compared with RDMA-enabled replication.

1 Introduction
Replicated distributed key-value stores (KVS) support many applications by providing scalability and high availability [7, 34]. The recent commercialization of persistent memory (PM) (e.g., Intel’s Optane DIMMs) enables local storage with extremely low latency (e.g., 10fs when persisting small data [7]). When building replicated distributed KVSs, we need to consider the multiple facets of network communication and request processing.

PM’s small memory capacity and high bandwidth can provide high performance, but low latency and high bandwidth may lead to severe device-level write amplification (DLWA) on PM [1, 2, 4]. DLWA waste PM’s persistence efficiency. In this paper, we propose Reshard, an efficient RDMA abstraction to handle replication writes on PM KVSs. Reshard can aggregate millions of concurrent in-memory writes from different services, and loads those writes to PM in sequential, thus reducing the replication latency compared with RDMA-enabled replication.

In our experiments, with 128B RDMA packets, Reshard achieves 1.39×, 1.77×, and 2.11× speedups boosting throughput by 1.22×, latency by 1.77×, and 2.11×, respectively, while largely eliminating DLWA.

In this paper, we propose Reshard, an efficient RDMA abstraction to handle replication writes on PM KVSs. Reshard can aggregate millions of concurrent in-memory writes from different services, and loads those writes to PM in sequential, thus reducing the replication latency compared with RDMA-enabled replication.

In our experiments, with 128B RDMA packets, Reshard achieves 1.39×, 1.77×, and 2.11× speedups boosting throughput by 1.22×, latency by 1.77×, and 2.11×, respectively, while largely eliminating DLWA.
Experimental Setup

Hardware Platform

- 6 machines as servers
- Intel Xeon Gold 6240M CPU (18 physical/36 logical cores)
- 3 × 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](image)

(a) Throughput vs. Latency
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

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

Software batching suffers the highest (50% more) PUT 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

(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

(a) Throughout vs. Latency

(b) DLWA

Rowan largely eliminates DLWA, like RPC
Performance Comparison with Other KVSs

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

(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

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

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

Contact Information: wq1997@tsinghua.edu.cn