RIO: Order-Preserving and CPU-Efficient Remote Storage Access

Xiaojian Liao, Zhe Yang, Jiwu Shu

Tsinghua University
Agenda

• Background and Motivation
  • RIO Design and Implementation
  • Evaluation
• Conclusion
Hardware and software trend

Hardware performance boosts, software overhead increases

**CPU (software)**

- ~ 50 MB/s
- ~ 500 MB/s
- ~ 5 GB/s
- ~ 10 GB/s

**Storage**

- HDD
- SATA SSD
- NVMe SSD (PCIe 4.0)
- NVMe/CXL SSD (PCIe 5.0)

**Network**

- 50 Gbps
- 100 Gbps
- 200 Gbps
Hardware and software trend

- Commodity RDMA NICs already offer a byte/memory interface
- Research SSDs offer a byte/memory interface to aid the design of system software

[1] 2B-SSD: The Case for Dual, Byte- and Block-Addressable Solid-State Drives, ISCA’18
[2] FlatFlash: Exploiting the Byte-Accessibility of SSDs within a Unified Memory-Storage Hierarchy, ASPLOS’19
[3] Crash Consistent Non-Volatile Memory Express, SOSP’21
[4] Hello bytes, bye blocks: PCIe storage meets compute express link for memory expansion, Hotstorage’22
System software design: storage order

- The system software design this paper focuses on: storage order
- **What** is storage order: the persistence order of a set of data blocks
- **Why** does storage order matter: storage reliability (crash consistency)
- **How** is storage order enforced: almost a synchronous fashion

![Diagram showing system software, SSDs, and concurrency](image-url)
The overhead of keeping storage order

- Measured tool: FIO. Workloads: append writes + fsync
- Network: Mellanox CX-6, RDMA. Storage: Samsung PM981 flash SSD, Intel 905P Optane SSD
- Compared systems: Linux NVMe over Fabrics, HORAE [OSDI’20][1]

[1] Write Dependency Disentanglement with HORAE, OSDI’20
Overhead analysis

• Linux’s approach to storage order

- Block layer
- NIC driver
- Network transfer
- SSD driver
- PCIe transfer
- FLUSH

Low concurrency

• HORAE’s approach to storage order

- Store ordering metadata
- Block layer
- NIC driver
- Network transfer
- SSD driver
- PCIe transfer

Low concurrency

> 4μs in NVMe over RDMA

Try to minimize or avoid synchronous processing!
Agenda

• Background and Motivation

• RIO Design and Implementation

• Evaluation

• Conclusion
RIO’s design insight

- Key observation: the layered design of modern I/O stack is similar to the pipeline.
  - Each layer performs a single functionality, and can process multiple requests concurrently (by the multi-queue interface).
  - Ordered write requests on non-overlapping LBAs can be parallelized.

The storage order should not stall the concurrent requests
RIO’s design overview

- Key idea: I/O pipeline for ordered write requests
  - Asynchronous processing: do not block, enables higher concurrency
  - Track storage order: enforce necessary synchronization, handle crashes

- RIO’s approach: speculative, optimistic, higher concurrency, recovery needed

- Linux’s approach: sequential, pessimistic, lower concurrency, no recovery
RIO’s I/O path

- W1 must be durable before W2
Tracking storage order in RIO

- Embed ordering attr. (describe the storage order) in each request
- Store ordering attr. to SSDs via MMIOs powered by the NVMe PMR feature
The motivation of RIO’s I/O scheduling

- Ordered write requests in RIO can be scheduled and merged
- Request merging reduces the overall CPU overhead of remote storage access, although merging itself requires some CPU cycles

Graphs showing CPU utilization with and without merging for Flash SSD and Optane SSD.
RIO’s I/O scheduling

- Separate ordered requests from the orderless via the ORDER queue
- Merge consecutive ordered requests in the ORDER queue without sacrificing the storage order
- Introduce the stream notion (a sequence of ordered write requests) for better scalability
- Align each stream to each NIC QP to exploit the in-order delivery of the network
Reorganizing journaling with RIO

- Concurrent JD, JM and JC, no barrier needed
- Per-file journal, each journal uses a dedicated stream

stream 1 (file B)  
JD  JD  JM  JC

stream 0 (file A)  
JD  JD  JM  JC

/dev/vda1

/dev/nvme0n1  Target 0
/dev/nvme1n1  Target 1
/dev/nvme2n1  Target 2
/dev/nvme3n1  Target 3
RIO’s Crash Recovery

• More details in the paper
  • Basic cases: out-of-place updates
  • Other cases: in-place updates
  • Data consistency and version consistency support

• Recovery overhead: 180 ms in the worst case (4 SSDs, 3 servers, 200Gbps RDMA)
Agenda

• Background and Motivation
• RIO Design and Implementation
• Evaluation
• Conclusion
Evaluation setup

- 3 Servers: 1 Initiator server and 2 target servers
- CPUs: each server 2 Intel Xeon Gold 5220, 18 cores, 2.2 GHz
- SSDs: Intel 905P(Optane), Intel P4800X(Optane), 2 * Samsung PM981(Flash)
- Network: NVIDIA ConnectX-6, 200 Gbps, RDMA
- OS: Ubuntu 18.04 LTS
- Compared Systems: Linux NVMe over RDMA from NVIDIA, an NVMe-oF version of HORAE[OSDI’20], RIO based on the same codebase of Linux NVMe over RDMA
Microbenchmark: ordered writes

Overall performance

- Workloads: multiple threads concurrently append 4 KB data blocks with storage ordering guarantee
- CPU efficiency: throughput / CPU utilization, normalized to the orderless Linux.

RIO ≈ orderless Linux
Evaluation: file systems

- Workloads: FIO 4KB append writes with fsync
- HORAEFS: the original HORAE + per-file journal; RIOFS: RIO + Ext4 + per-file journal

RIO achieves higher throughput, lower and more stable latency.
Evaluation: Varmail & RocksDB

- Varmail: Filebench default settings, create, unlink and fsync intensive
- RockDB: compaction intensive, 16B keys, 1024B values

RIO achieves higher throughput with less CPU cores
Agenda

- Background and Motivation
- RIO Design and Implementation
- Evaluation

- Conclusion
Conclusion

• RIO: Order-Preserving and CPU-Efficient Remote Storage Access
  • Problem: Storage order overhead. Root cause: synchronization.
  • Solution: RIO’s I/O pipeline = asynchronous processing + tracking storage order + recovery.
  • Result: higher CPU and I/O efficiency compared to existing systems.

• Takeaways:
  • Asynchronous processing (even in a synchronous interface) is the key to unleash the performance of fast storage and network hardware.
  • The byte interface is well suited for storing the temporary yet persistent metadata or control information of the storage systems.
Thank You!

RIO: Order-Preserving and CPU-Efficient Remote Storage Access

Xiaojian Liao, Zhe Yang, Jiwu Shu

Tsinghua University

liaoxiaojian@tsinghua.edu.cn
http://storage.cs.tsinghua.edu.cn/~lxj