**Libnvmmio**: Reconstructing SW IO Path with Failure-Atomic Memory-Mapped Interface

Jungsik Choi¹, Jaewan Hong², Youngjin Kwon², Hwansoo Han¹

**USENIX ATC ’20**
SW Overhead Greater than Storage Latency

Latency

ns

μs

ms

Time

SW Overhead

HDD

SSD

TLC 3D NAND SSD

Optane SSD

XL-Flash SSD

NVDIMM-N PM

DCPMM PM
Reconstruct SW IO Path with Libnvmmio

- Libnvmmio
  - Library
  - Run on any POSIX FS (DAX-mmap)
  - Transparent MMIO with logging
  - Make common IO path efficient
    - Handle data ops at user-level
    - Route metadata ops to kernel FS
  - Low-latency & scalable IO
  - Data-atomicity
User-Level IO is Suitable in NVMM system

• Kernel’s IO stacks introduce SW overhead

• User-level IO with `mmap`
  - Access files directly with `load/store`
  - Reduce user/kernel mode switches
  - Avoid complex IO stacks
  - No indexing, no permission checks

• MMIO is the fastest way to access files
Logging is more Efficient than CoW

• CoW (or shadow paging)
  - High write amplification
  - Hugepages make CoW more expensive
  - Frequent TLB-shutdown

• Logging (or journaling)
  - Writing data twice: logs and files
  - Differential logging
  - Checkpointing can be postponed
Redo vs. Undo

- Most logging systems use only one policy (redo or undo)
- They have different pros & cons depending on access type
  - REDO is better for writing, UNDO is better for reading
Hybrid Logging

• Uses **adaptive policy** depending on the access type of a file
  - **Read**-intensive file → **Undo** logging
  - **Write**-intensive file → **Redo** logging

• Maintains per-file read/write counters

• Determines logging policy on each fsync

• Achieves the best case performance of two logging policies
  - Reduce SW overhead and improve logging efficiency
Centralized Logging with Fine-Grained Locks

• Decentralized logging was designed for transactions
  - *e.g.*, per-thread logging, per-transaction logging

• Centralized logging is appropriate for file IO, but not scalable
  - Requires fine-grained locks for scalable file IO
Per-Block Logging

Multi-Level Tree
Per-Block Log
File

Radix Tree
Lock-Free Radix Tree

File Offset

Global 9
Upper 9
Middle 9
Table 9
Offset 12

Index Entry

Log Entry (4KB)

entry
rwlock
offset
len
dest
policy
epoch

Per-Block Log

LGD
LUD
LMD
Table

size

rgd
skip

radix_root
Commit & Checkpoint based on Epoch

• Per-block logs are atomically committed on fsync
• Libnvmmio commits by increasing the global epoch value
  - Committed logs have an epoch smaller than the global epoch
• Background checkpointing
Design Summary

**Libnvmmio** provides low-latency and scalable IO while guaranteeing data-atomicity

- **Low-latency IO**
  - User-level IO with mmap
  - Differential logging
  - Hybrid logging
  - Various log sizes
  - Epoch-based committing
  - Background checkpointing

- **Scalable IO**
  - Per-block logging
  - Lock-free index data structure
Experimental Setup

• Experimental Machines
  - 32GB **NVDIMM-N**, 20 cores and 32GB DRAM
  - 256GB **Optane DC**, 16 cores and 128GB DRAM (in our paper)

• Comparison systems

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>File IO</th>
<th>Data-Atomicity</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext4-DAX</td>
<td>Kernel</td>
<td>X</td>
<td>5.1</td>
</tr>
<tr>
<td>PMFS</td>
<td>Kernel</td>
<td>X</td>
<td>4.13</td>
</tr>
<tr>
<td>NOVA</td>
<td>Kernel</td>
<td>O</td>
<td>5.1</td>
</tr>
<tr>
<td>SplitFS</td>
<td>User</td>
<td>O</td>
<td>4.13</td>
</tr>
<tr>
<td>Libnvmmio*</td>
<td>User</td>
<td>O</td>
<td>5.1</td>
</tr>
</tbody>
</table>
Hybrid Logging

![Graph showing Hybrid Logging performance with R:W Ratio vs. Elapsed Time (sec). The graph compares Undo, Redo, and Hybrid methods, indicating a trend where Hybrid performs better than the others.]
FIO: Different Access Patterns

- A single thread, file size=4GB, block size=4KB, time=60s
FIO: Different Write Sizes

![Graph showing bandwidth (GiB/s) for different write sizes (128B, 1KB, 4KB, 64KB, 1MB) for Ext4-DAX, PMFS, NOVA, and Libnvmmio.]

- **Ext4-DAX**
- **PMFS**
- **NOVA**
- **Libnvmmio**
FIO: Random Write with Multithreads

![Graph showing Bandwidth (GB/s) vs. Number of Threads for Private and Shared files with different file systems and benchmarks.](image-url)
TPC-C on SQLite

• Underlying FS with WAL, and Libnvmmio without WAL
SQLite WAL vs. Libnvmmio

• SQLite WAL
  – Design for block devices
  – Similar to REDO logging
  – Read both WAL and DB file
  – Only one writer at a time
  – Synchronous checkpointing

• Libnvmmio
  – Design for NVMM
  – Hybrid Logging
  – Read DB file (UNDO)
  – Concurrent writes
  – Background checkpointing

• Easily improve performance with Libnvmmio
  – Support any FS, Even FS that does not provide data-atomicity
Conclusion

- It is important to minimize SW overhead in NVMM systems
- Libnvmmio is a simple and practical solution
  - Reconstruct SW IO path
  - Run on any filesystem that provide DAX-mmap
- Low-latency, scalable IO while guaranteeing data-atomicity
  - 2.2x better throughput
  - 13x better scalability
- [https://github.com/chjs/libnvmmio](https://github.com/chjs/libnvmmio)
chjs@skku.edu