# MadFS: Per-File Virtualization for Userspace Persistent Memory Filesystems

***21st USENIX Conference on File and Storage Technologies***

*Shawn Zhong; Chenhao Ye; Guanzhou Hu; Suyan Qu; Andrea Arpaci-Dusseau; Remzi Arpaci-Dusseau; and Michael Swift*

**Paper Notes**

By JeongHa Lee

## Abstract

Persistent memory (PM) can be accessed directly from userspace without kernel involvement, but most PM filesystems still perform metadata operations in the kernel for security and rely on the kernel for cross-process synchronization.

We present per-file virtualization, where a virtualization layer implements a complete set of file functionalities, including metadata management, crash consistency, and concurrency control, in userspace. We observe that not all file metadata need to be maintained by the kernel and propose embedding insensitive metadata into the file for userspace management. For crash consistency, copy-on-write (CoW) benefits from the embedding of the block mapping since the mapping can be efficiently updated without kernel involvement. For cross-process synchronization, we introduce lockfree optimistic concurrency control (OCC) at user level, which tolerates process crashes and provides better scalability.

Based on per-file virtualization, we implement MadFS, a library PM filesystem that maintains the embedded metadata as a compact log. Experimental results show that on concurrent workloads, MadFS achieves up to 3.6$\times$ the throughput of ext4-DAX. For real-world applications, MadFS provides up to 48% speedup for YCSB on LevelDB and 85% for TPC-C on SQLite compared to NOVA.

## Problem Statement and Research Objectives

- **Metadata management**
  - In kernel filesystems, **metadata is managed exclusively by the kernel** for security reasons.
  - A major challenge of userspace filesystems comes from **untrusted libraries**. Thus, many of them **still rely on the kernel for metadata management**.
- **Crash consistency**: PM only guarantees the atomicity of a single 64-bit store, so filesystems need to build their own constructs for crash consistency. For data crash consistency, CoW is commonly used. However, CoW has two major drawbacks when used with memory-mapped I/O.
  - **Huge pages(2 MB or 1 GB on x86- 64)** have been shown to have significant performance improvements for PM filesystems due to fewer page faults, less TLB shootdown, and shorter page table walk.
    However, **the granularity of CoW is coupled with the page size**. Therefore, writing to a sub-page results in copying the entire page, causing significant write amplification.
  - In addition to huge pages, kernel-level CoW causes expensive TLB shootdowns.
- **Concurrency control**
  - For userspace filesystems, concurrency control is challenging, **especially in cross-process cases**. For example, a process could crash while holding a lock, blocking other processes. As a result, most userspace PM filesystems either **do not support cross-process synchronization** or use **lease-based locking**.

## Proposed Method

### 1. Embedded block map

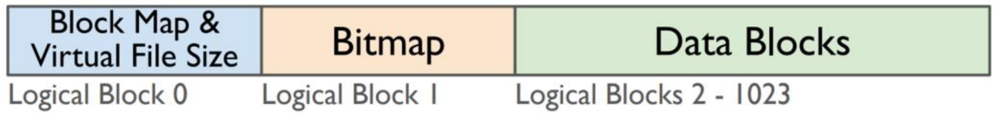

Figure 1: A naive approach for metadata embedding (§4.1)

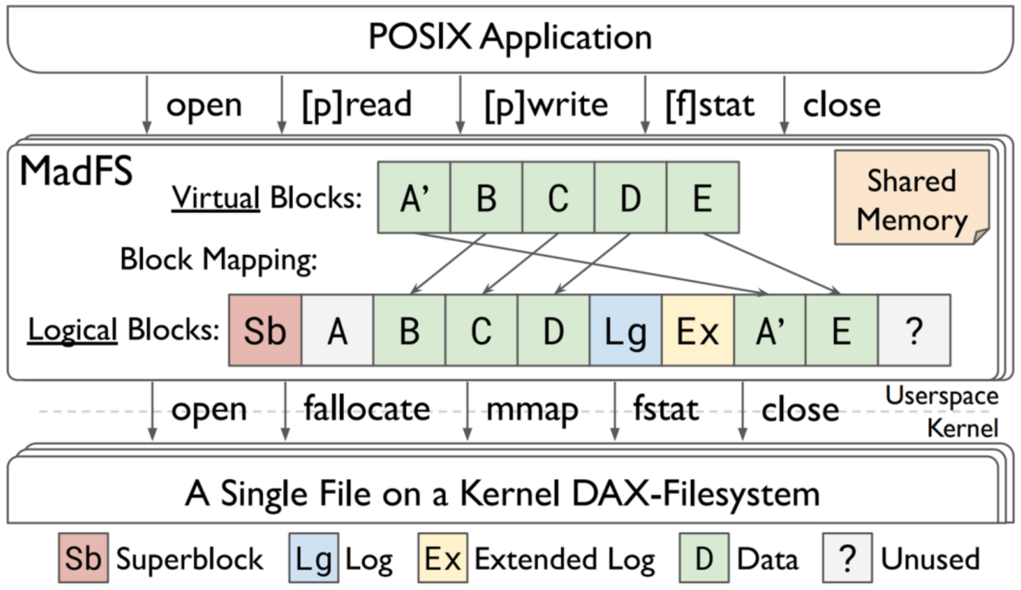

Figure 2: The architecture of MadFS. The application sees *virtual* blocks, which are mapped to the *logical* blocks backed by the kernel filesystem.

- **Superblock** is the first block, which contains a magic number that identifies MadFS files and a pointer to the first log block.
- **Log blocks** consist of an array of fixed-size log entries, each corresponding to a metadata update. Each log block also carries a pointer to the next one, forming a linked list.
- **Extended log blocks** store extended log entries, which contain additional information about a metadata update that does not fit into the fixed-size log entry.
- **Data blocks** contain the user data. Each virtual block seen by the application is backed by a logical data block.
- **Unused blocks** are blocks that are not referenced by the block map. They appear due to pre-allocation from the kernel filesystem and garbage collection.

# Proposed Method

## 2. Compact log-structured metadata

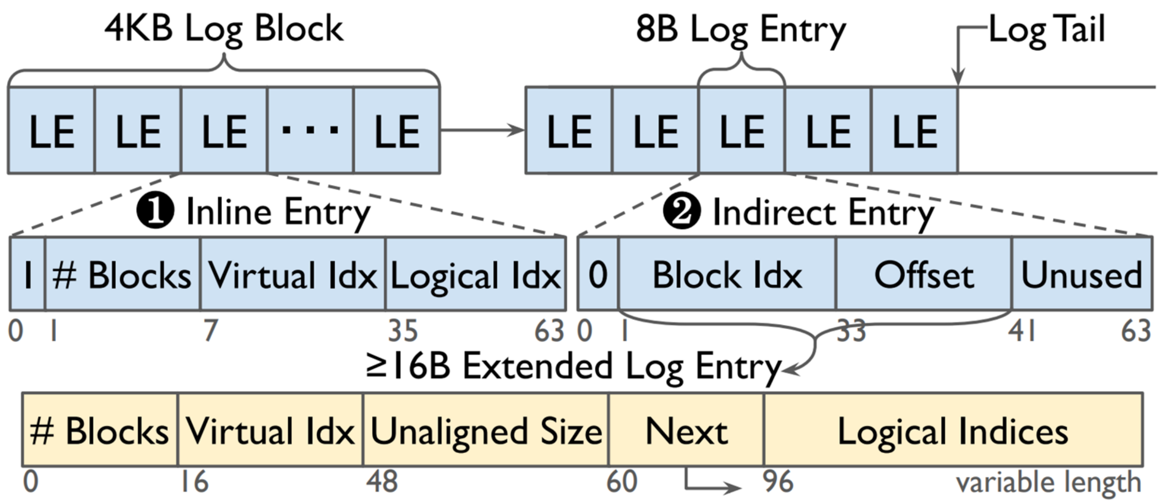

Figure 3: Layout of the log-structured metadata (§4.3). A metadata update is described as either an *inline* log entry or an *indirect* log entry pointing to an *extended* log entry.

❶ An inline log entry is used to represent updates of less than or equal to 64 blocks, which is the maximum number of contiguous logical blocks that the allocator can provide.

❷ An indirect log entry is for more complex updates. It carries a pointer to a variable-length extended log entry that contains a virtual block range and an array of logical block indices.

# Proposed Method

## 3. Lock-free optimistic concurrency control

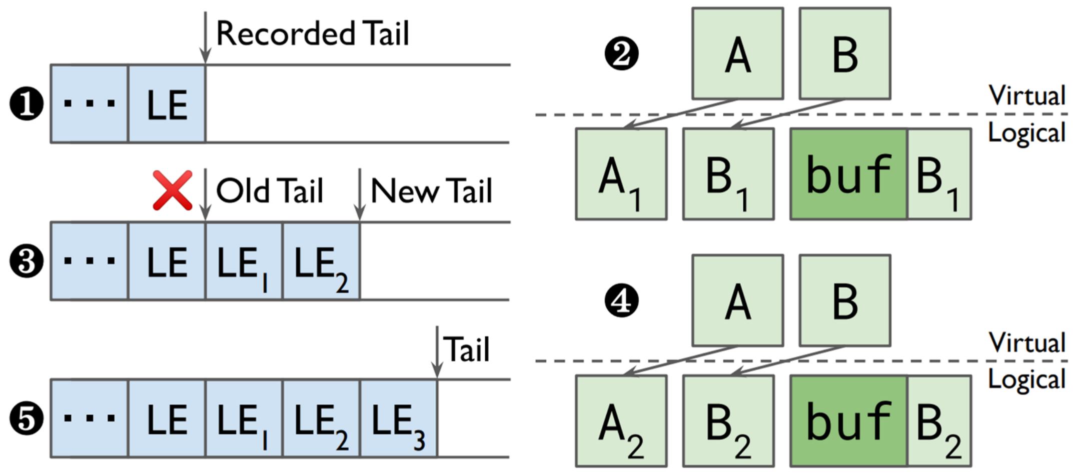

Figure 4: Concurrent writers example (§4.4). Each 🟦 represents an 8-byte log entry. Extended log entries are omitted. Each 🟩 represents a 4 KB data block.

1. The block table is updated and record the current log tail.
2. The writer does a CoW and generates a log entry to commit.
3. When the thread tries to commit to the recorded log tail via CAS, it finds that the tail has been moved.
4. Suppose $LE_1$ remaps block A to $A_2$ and $LE_2$ remaps B to $B_2$. Although both log entries overlap with the current write, the thread only needs to recopy the unaligned part of $B_2$. There is no need to recopy block A since it will be completely overwritten.
5. The current thread successfully commits the log entry to the latest tail at $LE_3$ .
6. The block table will be updated to reflect the changes in the block map.

# Evaluation and Results

1. Single-thread performance of MadFS

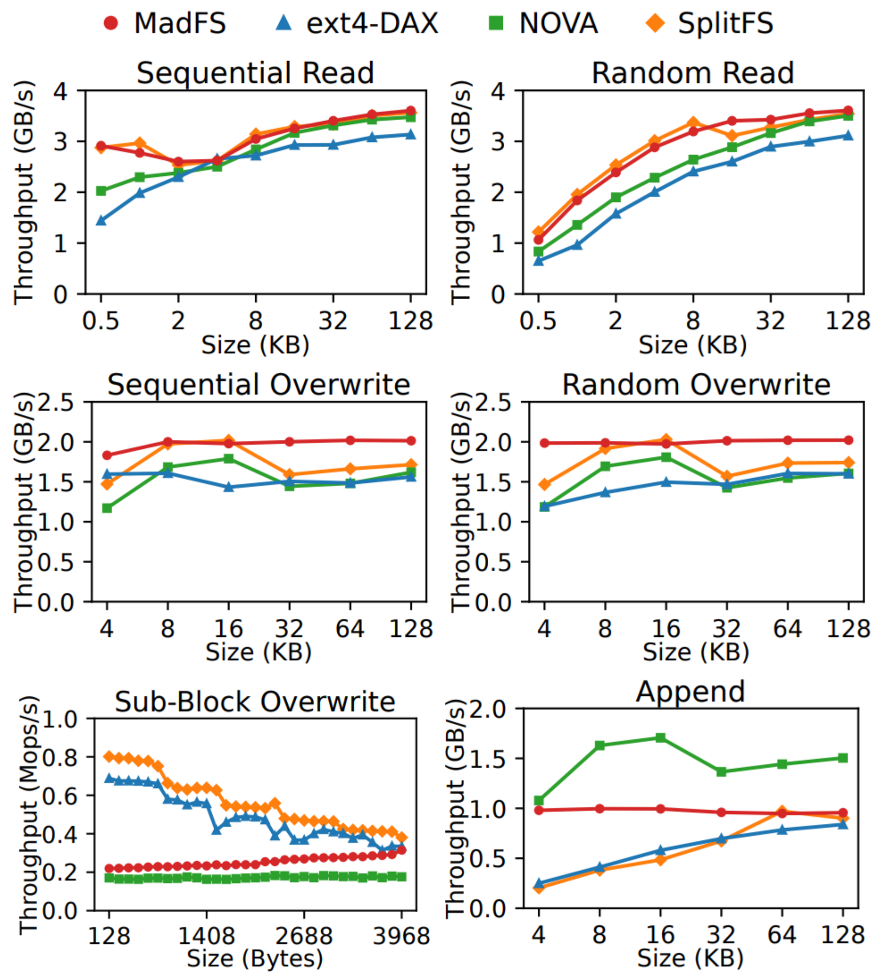

Figure 6: Single-threaded performance. Note for sub-block overwrite, we report throughput in Mops/s instead of GB/s.

2. Multi-thread performance of MadFS

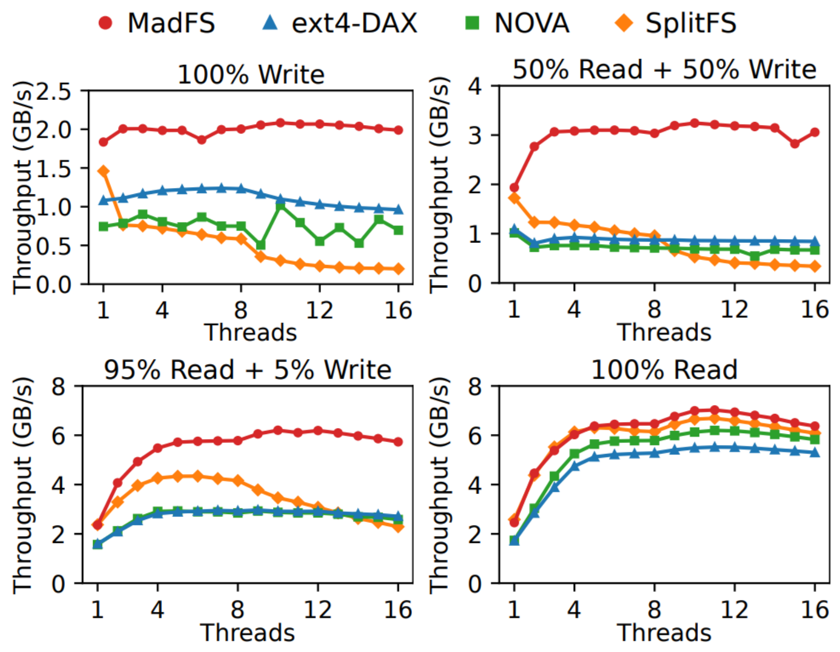

Figure 8: Councurrent 4 KB read/write with uniform offset.

3. File open overhead in MadFS

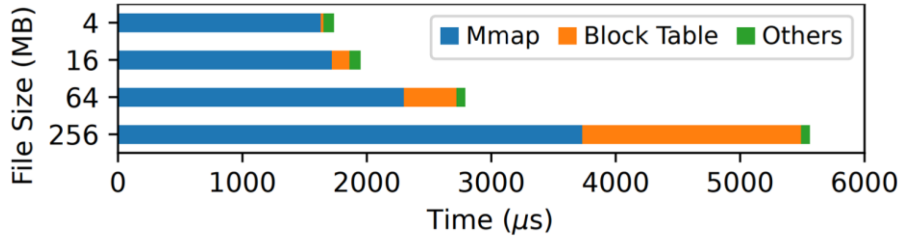

Figure 11: Open latency breakdown. The file size is logical.

# Notes

1 : https://en.wikipedia.org/wiki/Filesystem_in_Userspace

- Filesystems for Persistent Memory
  - **Kernel filesystems**: Mature Linux filesystems such as ext4 and XFS introduce **direct access (DAX) mode**, which **bypasses the page cache** and allows applications to directly access file data stored on PM via memory-mapped I/O.
  - **Userspace filesystems**: With ultra-fast hardware, software overhead becomes non-trivial. Thus, many PM filesystems have proposed to **bypass the kernel**.
- Filesystem in Userspace (FUSE) [1]
  - A software interface for Unix and Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code.
  - This is achieved by running file system code in user space while the FUSE module provides only a bridge to the actual kernel interfaces.