CrossFS: A Cross-layered Direct-Access File System

Yujie Ren (Rutgers University), Changwoo Min (Virginia Tech), Sudarsun Kannan (Rutgers University)

1 Introduction

The storage access latency in modern storage devices is transitioning from milliseconds to microseconds [7]. While modern applications strive to increase I/O parallelism, storage software bottlenecks such as system call overheads, coarse-grained concurrency control, and the inability to exploit storage hardware concurrency continues to impact I/O performance. Several kernel-level, user-level, and firmware-level file systems have been designed to benefit from CPU parallelism [9], direct storage access [2, 3, 5], or computational capability embedded in the storage hardware [4]. However, these approaches are designed in isolation and fail to exploit modern, ultra-fast storage hardware.

Kernel-level file systems (Kernel-FS) satisfy *fundamental file system guarantees* such as integrity, consistency, durability, and security. Despite years of research, Kernel-FS designs continue to suffer from **three main bottlenecks**. First, applications must enter and exit the OS for performing I/O, which could increase latency by 1-4μs [3]. Recently found security vulnerabilities have further amplified such costs. Second, even state-of-the-art designs enforce unnecessary serialization (e.g., inode-level read-write lock) when accessing disjoint portions of data in a file leading to high concurrent access overheads. Third, Kernel-FS designs *fail to fully exploit* storage hardware-level capabilities such as device-level compute, thousands of I/O queues, and firmware schedulers, which impacts I/O latency, throughput, and concurrency in I/O-heavy applications.

As an alternative design point, there is an increasing focus towards designing user-level file systems (User-FS) for direct storage access bypassing the OS [1–3, 5, 7]. However, satisfying the fundamental file system guarantees from untrusted user-level is challenging [4]. While these designs have advanced the state of the art, some designs bypass the OS only for data-plane operations (without data sharing) [1, 3, 7]. In contrast, others provide full direct access by either sidestepping or inheriting coarse-grained and suboptimal concurrency control across threads and processes [2, 5], or even compromise correctness. Importantly, most User-FS designs fail to exploit the hardware capabilities of modern storage.

At the other extreme is the exploration of firmware-level file systems (Firmware-FS) that embed the file system into the device firmware for direct-access [4]. The Firmware-FS acts as a central entity to satisfy fundamental file system properties. Although a first important step towards utilizing storage-level computational capability (with 4-8 cores in modern storage), current designs miss out on benefiting from host-level multi-core parallelism. Additionally, these designs inherit inode-centric design for request queuing, concurrency control, and scheduling, leading to poor I/O scalability.

In summary, current User-FS, Kernel-FS, and Firmware-FS designs *lack a synergistic design across the user, the kernel, and the firmware layers*, which is critical for achieving direct storage access and scaling concurrent I/O performance without compromising fundamental file system properties.

To address the aforementioned bottlenecks, we propose

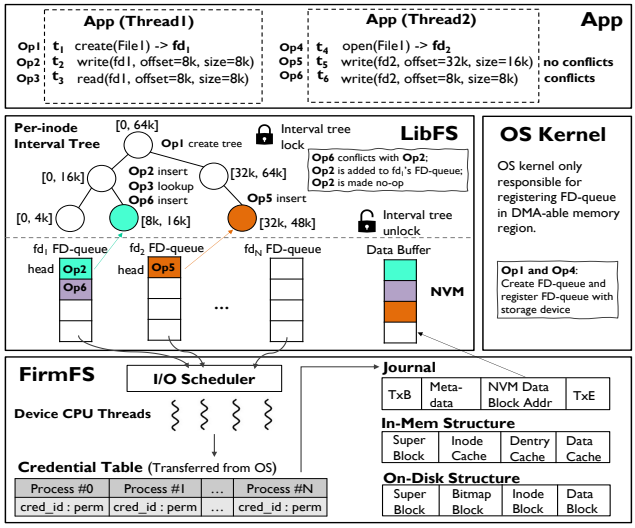


Figure 1: **CrossFS Design Overview.** Host-level LibFS converts POSIX I/O calls to FirmFS commands, and manages FD-queues, interval tree for checking block conflicts and ordering requests. FirmFS implements file system with journaling and scheduler, and concurrently processes requests across FD-queues. The OS component is responsible for the FD-queue setup and updates credential table in FirmFS with host-level permission information. Figure shows example of two instances sharing a file; *Op1* to *Op6* show request execution with global timestamps *t1* to *t6*. *Op6* conflicts with *Op2*, so *Op6* is added to the same FD-queue as *Op2* using an interval tree.

CrossFS[1], a cross-layered direct-access file system that provides scalability, high concurrent access throughput, and lower access latency. CrossFS achieves these goals through four main principles. First, CrossFS disaggregate file system components across user-space, firmware, and OS layers to exploit host and device-level compute resources. Second, aligns each file descriptor to independent hardware I/O queue (FD-queue) to provide fine-grained concurrency control at the file-descriptor level granularity as opposed to inodes in current file systems. Third, CrossFS protects the in-transit user data and the file system state by leveraging persistence provided by byte-addressable NVMs. Fourth, CrossFS unifies software and hardware I/O schedulers into a single firmware scheduler to improve throughput (more details in [8]).

CrossFS Layers. CrossFS enables POSIX-compatible applications to benefit from direct storage access. As shown in Figure 1, CrossFS comprises of a user-level library (LibFS), a firmware file system component (FirmFS), and an OS component. The user-level library component (LibFS) provides POSIX compatibility and handles concurrency control and conflict resolution using the host-level CPUs (host-CPUs). The OS component sets up the initial interface between LibFS and FirmFS (e.g., I/O queues) and converts software-level access control to hardware security control. The firmware component (FirmFS) is the heart of the file system, enabling applications to directly access the storage without compromising fundamental file system properties. The FirmFS taps into storage hardware's I/O queues, computational capability, and

[1]Prototype available at *https://github.com/RutgersCSSystems/CrossFS*

I/O scheduling capability for improving I/O performance.

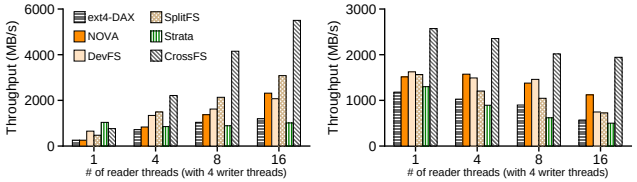
Scalability. File system disaggregation alone is insufficient for achieving I/O scalability, which demands revisiting file system concurrency control, reducing journaling cost, and designing I/O scheduling that matches the concurrency control. We observe that file descriptor (and not inode) is a natural abstraction of access in most concurrent applications, where threads and processes use independent file descriptors to access/update different regions of shared or private files (i.e., RocksDB maintains 3.5K open file descriptors). Hence, for I/O scalability in CrossFS, we introduce file descriptor-based concurrency control, which allows threads or processes to update or access non-conflicting blocks of a file simultaneously.

Concurrency Control via Queue Ordering. In CrossFS, file descriptors are mapped to dedicated hardware I/O queues to exploit storage hardware parallelism and fine-grained concurrency control. All non-conflicting requests (i.e., requests to different blocks) issued using a file descriptor are added to a file descriptor-specific queue (`Op2` and `Op5` in Figure 1). In contrast, conflicting requests are ordered by using a single queue (`Op2` and `Op6` in Figure 1). This provides an opportunity for device-CPUs and FirmFS to dispatch requests concurrently with almost zero synchronization between host and device-CPUs. For conflict resolution and ordering updates to blocks across file descriptors, CrossFS uses a per-inode interval tree, interval tree read-write semaphore (interval tree `rw-lock`), and global timestamps for concurrency control. However, *unlike current file systems that must hold inode-level locks* until request completion, CrossFS only acquires interval tree `rw-lock` for request ordering to FD-queues. In short, CrossFS concurrency design *turns the file synchronization problem into a queue ordering problem*.

CrossFS Challenges. Moving away from an inode-centric to a file descriptor-centric design introduces CrossFS-specific challenges. First, using fewer and wimpier device-CPUs for conflict resolution and concurrency control impacts performance. Second, mapping a file descriptor to an I/O queue (a device-accessible DMA memory buffer) increases the number of queues that CrossFS must manage, potentially leading to data loss after a crash. Finally, overburdening device-CPUs for serving I/O requests across hundreds of file descriptor queues could impact performance, specifically for blocking I/O operations (e.g., `read, fsync`).

Host Delegation. To overcome the challenge of fewer (and wimpier) device-CPUs, CrossFS utilizes the cross-layered design and delegates the responsibility of request ordering to host-CPUs. The host-CPUs order data updates to files they have access to, whereas FirmFS is ultimately responsible for updating and maintaining metadata integrity, consistency, and security with POSIX-level guarantees.

Crash-Consistency and Scheduler. To handle crash consistency and protect data loss across tens and possibly hundreds of FD-queues, CrossFS uses byte-addressable, persistent NVMs as DMA'able and append-only FD-queues from which FirmFS can directly fetch requests or pool responses. CrossFS also designs low-cost data journaling for crash-consistency of firmware file system state. Finally, for efficient scheduling of device-CPUs, CrossFS smashes traditional two-level I/O schedulers spread across the host-OS and the firmware into one FirmFS scheduler equipped with configurable policies that enhance file descriptor-based concurrency.



(a) Aggregated Read Throughput (b) Aggregated Write Throughput

Figure 2: **Microbenchmark.** Throughput of concurrent readers and 4 writers randomly accessing a 12GB file. For CrossFS and DevFS, 4 device-CPUs are used.

2 Evaluation

We evaluate CrossFS on a 64-core dual-socket server with 32GB DRAM and 512GB (4x128GB) Optane DC NVM for storage and FD-queues. To emulate the proposed cross-layered file system, similar to prior work [4], we implement FirmFS in Linux kernel as a device driver with dedicated `kthreads`. We reserve and use 2GB of DRAM for maintaining FirmFS in-memory state. To emulate PCIe latency, we add a 900ns software delay [6] between the time a request is added to the host's FD-queue and the time a request is marked ready for FirmFS processing.

In Figure 2a and Figure 2b, we vary the number of concurrent readers in the x-axis setting the concurrent writer count to four threads. For this multithreaded micro-benchmark, we use LibFS-level interval tree updates. We compare CrossFS against ext4-DAX and NOVA (Kernel-FS), DevFS (Firmware-FS), SplitFS and Strata (User-FS), all using Intel Optane DC persistent memory for storage. As discussed earlier, all these designs suffer from high overheads of inode-level `rw-lock`, which impacts I/O scaling. In contrast, CrossFS's cross-layered design with file descriptor concurrency allows concurrent read and write across FD-queues, achieving up to 3.64x read throughput gains and 3.38x write throughput gains over state-of-the-art file systems. Real-world applications RocksDB and Redis show up to 2.32x and 2.35x gains (see [8]).

References

[1] Adrian M. Caulfield, Todor I. Mollov, Louis Alex Eisner, Arup De, Joel Coburn, and Steven Swanson. Providing Safe, User Space Access to Fast, Solid State Disks. *SIGARCH Comput. Archit. News*.

[2] Mingkai Dong, Heng Bu, Jifei Yi, Benchao Dong, and Haibo Chen. Performance and Protection in the ZoFS User-Space NVM File System. SOSP '19.

[3] Rohan Kadekodi, Se Kwon Lee, Sanidhya Kashyap, Taesoo Kim, Aasheesh Kolli, and Vijay Chidambaram. SplitFS: Reducing Software Overhead in File Systems for Persistent Memory. SOSP '19.

[4] Sudarsun Kannan, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Yuangang Wang, Jun Xu, and Gopinath Palani. Designing a True Direct-access File System with DevFS. FAST'18.

[5] Youngjin Kwon, Henrique Fingler, Tyler Hunt, Simon Peter, Emmett Witchel, and Thomas Anderson. Strata: A Cross Media File System. SOSP '17.

[6] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. Understanding PCIe Performance for End Host Networking. SIGCOMM'18.

[7] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The Operating System is the Control Plane. OSDI'14.

[8] Yujie Ren, Changwoo Min, and Sudarsun Kannan. CrossFS: A Cross-layered Direct-Access File System. OSDI'20.

[9] Jian Xu and Steven Swanson. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. FAST'16.