FusionFS: Fusing I/O Operations using CISC$_{Ops}$ in Firmware File Systems

Jian Zhang[*] *Yujie Ren*[*] *Sudarsun Kannan*
Rutgers University

1 Introduction

Modern high bandwidth and low-latency storage technologies such as NVMe SSDs and 3D-Xpoint have significantly accelerated I/O performance leading to better application performance. Yet, the combination of software and hardware I/O overheads that include system calls, data movement, and communication cost in the application and the OS, and the storage hardware latency (e.g., PCIe) continue to be an Achilles heel in fully exploiting storage hardware capabilities.

A recent focus is to reduce software indirections by moving filesystems to userspace and avoiding system calls and kernel traps for data and metadata updates [3]. Although effective, the dominating I/O overheads such as data and metadata movement cost, host and device communication cost (e.g., PCIe latency), and indirect costs like polling or interrupts remain. Henceforth, we refer to the combination of above overheads, which includes system calls, as dominating I/O overheads.

Another design point to reduce I/O overheads is the reincarnation of near-storage processing. Vendors are introducing computational storage devices (**CSD**) that embed in-storage processors that range from ARM cores to FPGAs.

More broadly, these techniques can be classified into systems that focus on (1) in-storage compute offloading and (2) in-storage filesystems and key-value stores designed to accelerate I/O and storage management. First, in-storage compute offloading systems (which includes a majority of current CSD solutions) such as PolarDB focus on data processing by rewriting application logic to offload computation. While beneficial, these systems either lack storage management or delegate management to the host file system. The former leads to a lack of data and metadata integrity, crash consistency, durability, or managing in-storage resources across tenants.

In contrast, in-storage management designs like CrossFS [6], DevFS [4], and Insider [7] offload filesystems and key-value stores inside the storage firmware for direct-I/O, bypassing the OS. Unfortunately, these designs lack near-storage processing capability leading to substantial data movement and failing to manage in-storage resources such as device compute and memory or handle multi-tenancy.

We envision an ideal near-storage design that co-designs and combines storage management and data processing by rethinking I/O abstractions to reduce dominant I/O overheads, such as system calls, data and metadata movement, and host to device communication latency. Importantly, the design must ensure (storage) correctness, handle crash consistency, and achieve fairness across tenants.

Therefore, We propose **FusionFS**, a near-storage file system design to exploit device compute and memory resources for reducing dominant I/O overheads and improving application performance. FusionFS achieves these goals through four main principles. First, FusionFS co-designs in-storage management and data processing to eliminate dominating I/O overheads. Second, designs abstractions to reduce host and device interactions. Third, FusionFS exploits in-storage compute for fine-grained crash consistency and faster recovery. Fourth, FusionFS manages in-storage resources for fairness and performance efficiency across tenants (Full version of FusionFS appeared at FAST 2022 and the code is available at [8]).

[*]The authors contribute equally to this paper.

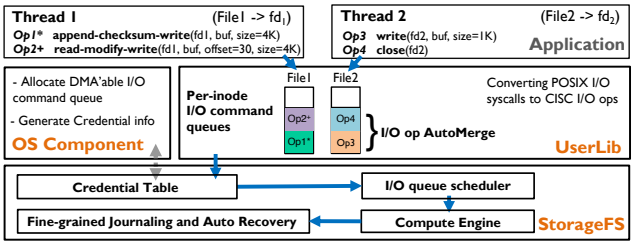


Figure 1: **FusionFS High-level Design.** Figure shows the high-level design of FusionFS with the UserLib, the StorageFS, and the OS components. For thread1, Op1 and Op2 show a CISC$_{Op}$ with data processing, whereas Op3 and Op4 show simple I/O. StorageFS shows the in-device structure with durability, permission, and scheduling components.

Overview. We design FusionFS as a disaggregated I/O stack with direct-I/O that splits the file system into the host-level user-level library (UserLib) and in-storage (StorageFS) components that work in tandem to offload I/O and computation without compromising correctness, crash consistency, recovery, security, and resource fairness (see Figure 1). Applications either issue traditional POSIX I/O requests or CISC$_{Ops}$ to a DMA'able inode-queue, which are then processed by StorageFS by checking the permission for each I/O request and scheduling them for processing, in addition to fine-grained durability and recovery.

CISC$_{Ops}$. Exploiting the in-storage capabilities requires reducing frequent interaction between the host and device to reduce software and hardware overheads. Therefore, we take inspiration from seminal RISC (reduced instruction set computers) and CISC (complex instruction set computers) processor design and apply it for I/O to either offload simple RISC-styled operations (e.g., read, write, open) or CISC-styled CISC$_{Ops}$. CISC$_{Ops}$ extends the NVMe interface to aggregate I/O and data processing sequences to significantly reduce dominant overheads (system calls, data movement, and device and host communication costs). While seemingly simple, unlike traditional vector I/O (e.g., *writev*, *readv*), CISC$_{Ops}$ composes heterogeneous I/O and data processing operations, which raises new challenges. We discuss the principles of composing CISCOps followed by challenges of realizing CISC$_{Ops}$.

Principle 1: CISC$_{Ops}$ for Identical and Non-identical I/O Operations: We observe that I/O operations are executed in sequence or pairs in several applications. For example, Figure 2(a) shows a widely-used NoSQL database and webserver sequence that opens, writes, syncs, and closes the file when inserting values (i.e., open()->write()->sync()->close()) or when reading data [1]. The figure also shows overheads for each operation, which includes data movement and system call costs. We observe several such sequences contributing to I/O overheads [8]. In contrast, CISC$_{Ops}$ aggregates and offloads such sequences to StorageFS reducing I/O overheads.

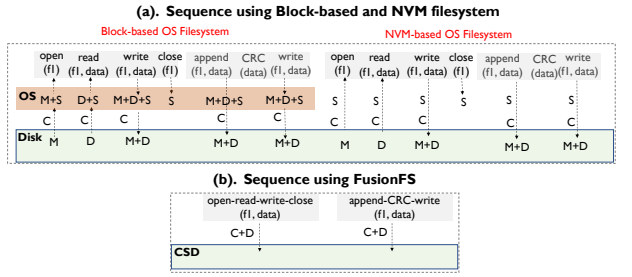


Figure 2: **Comparison of I/O Overheads.** (a) and (b) compare data movement (D), communication cost (C), and system call (S) using traditional storage and envisioned $CISC_{Ops}$ design that bypasses OS);

Principle 2: $CISC_{Ops}$ for I/O and Data Processing Operations. For I/O and data processing inside CSDs, unlike prior approaches that require significant application changes [2], we focus on organically supporting I/O and their related pre and post-processing to reduce I/O overheads. Specifically, we observe that applications (e.g., NoSQL key-value stores) frequently fetch I/O data to perform operations like checksum generation (CRC) to prevent the propagation of corrupted data by adding CRC for integrity check, compression/decompression, encryption, search, sort, and ML operation pairs (e.g., XOR, multiplication). For example, with CRC, after each file system append() system call, the CRC is computed and appended to the actual data. $CISC_{Ops}$ provides a capability to combine these operations into *append-CRC-write* $CISC_{Ops}$ and offload to StorageFS, thereby requiring just one data movement without system call cost (see Figure 2).

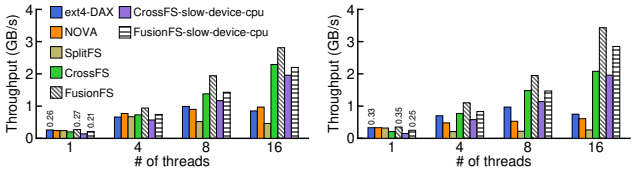
Application Support. With the explicit approach, applications can use $CISC_{Ops}$ pre-constructed by UserLib or construct custom $CISC_{Ops}$. Each $CISC_{Op}$ is a vector of commands in an extended NVMe format for supporting multiple operations and added to inode-queue for processing. The number of elements in the $CISC_{Op}$ is configurable, and by default, can pack *32* operations to fit in a DMA-able page.

Fine-grained Crash-Consistency and Fast Recovery. Beyond supporting journaling for basic file system operations inside StorageFS, FusionFS must support crash consistency and recovery for I/O and data processing operations in a $CISC_{Ops}$. Challenges arise in terms of granularity and the benefits of exploiting in-storage compute to accelerate recovery. To address the challenges, we explore macro-transactions (MacroTx) and micro-transactions (MicroTx). MacroTx uses an all-or-nothing approach that only commits and recovers an entire $CISC_{Op}$ including the data processing state, whereas MicroTx supports crash consistency of partially committed $CISC_{Ops}$. Further, to reap the benefits of MicroTx, we go a step beyond current filesystems and use in-storage compute to support operational logging and automatic recovery by finishing partially completed $CISC_{Ops}$.

2 Evaluation

To understand the benefits and implications of reducing dominating I/O overheads like kernel/userspace crossing, data movement cost, and communication cost between host and device, we study the performance of state-of-the-art file systems designs with two benchmark workloads: (1) *append-CRC-write* and (2) *read-modify-write*. These operations are commonly used in key-value stores, databases, and several other applications [8].

Due to the lack of programmable CSD, we carefully emulate our FusionFS on a 64-core dual-socket server with 96GB DRAM and 256GB Optane DC NVM for storage. For StorageFS processing, we reserve 4 CPUs. We also study the impact of varying CPU speeds using fast (2.7GHz and default) and slow (1.2GHz) CPUs resembling ARM-based CSDs. For device-RAM, we reserve 2 GB memory managed by StorageFS. We study the impact of device-RAM bandwidth using a 64-core CloudLab machine. For PCIe latency, we add 900ns [5] delay before a request is processed.



(a) Append-CRC-Write (b) Read-Modify-Write

Figure 3: **Microbenchmark.** Shows aggregated throughput. CrossFS and FusionFS use 4 device cores.

In Figure 3, we vary the number of benchmark threads in the x-axis, and the y-axis shows the throughput, and the threads use separate files. We compare ext4-DAX and NOVA (kernel file systems), SplitFS (hybrid user-level file system), CrossFS (in-storage file system), and FusionFS. Additionally, to understand the impact of slower device-CPUs, we also evaluate in-storage StorageFS to use 1.2GHz device-CPUs (*CrossFS-slow-device-cpu* and *FusionFS-slow-device-cpu*).

First, kernel-level ext4-DAX and NOVA provides direct access without data copies to page cache but incurs significant system call and data copy cost. Next, hybrid user-level SplitFS memory-maps storage to userspace and replaces reads/writes with loads/store operations. SplitFS reduces system calls, but metadata updates require frequent OS interaction. Further, CrossFS, an emulated firmware-level file system design, reduces system calls and only metadata movement between filesystem and storage, resulting in higher performance.

In contrast, FusionFS's $CISC_{Ops}$ design avoids system calls, reduces a data copy between the application and the OS, and the PCIe latency, all leading to up to 4.58X gains over state-of-the-art file systems. Further, real-world applications like Snappy and LevelDB show up to 1.63X and 2.1X gains (not shown due to space constraints, please see [8]).

References

[1] Google LevelDB . http://tinyurl.com/osqd7c8.

[2] Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, Zhenjun Liu, Feng Zhu, and Tong Zhang. POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database. In *FAST '20*.

[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. In *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. In *FAST '18*.

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

[6] Yujie Ren, Changwoo Min, and Sudarsun Kannan. Crossfs: A cross-layered direct-access file system. In *OSDI '20*.

[7] Zhenyuan Ruan, Tong He, and Jason Cong. INSIDER: Designing In-Storage Computing System for Emerging High-Performance Drivei. In *ATC '19*.

[8] Jian Zhang, Yujie Ren, and Sudarsun Kannan. FusionFS: Fusing I/O operations using CISCOps in firmware file systems. In *FAST '22*.