Strata: A Cross Media File System

Youngjin Kwon Henrique Fingler Tyler Hunt
Simon Peter Emmett Witchel Thomas Anderson[1]

The Unversity of Texas at Austin [1]University of Washington

1. Introduction

A set of interacting hardware and software trends stress file systems from below and from above. Below the file system, the market for storage devices has fragmented based on a tradeoff between performance and cost per capacity. SSD technologies are fast evolving and outperform traditional HDDs with much wider IO bandwidth and lower latency, but they have lower capacity. Emerging non-volatile memory (NVM) techniques such as Intel's 3D XPoint and NVDIMM promise revolutionary IO performance, but with high cost-per-capacity. Above the file system, modern applications demand performance and functionality far outside the traditional file system comfort zone. Applications require fast, small, scattered updates on vast datasets and a simple programming model for crash consistency [2].

These storage and application trends are a poor fit for existing file systems that have been designed with only one storage medium in mind. For example, with the arrival of NVM, it no longer makes sense to engineer file systems on old assumptions like the kernel must intermediate every metadata operation or that file operations are inherently asynchronous. Similarly, tying the file system to a single type of physical device, for example to NVM, is expensive for large capacity. To get the best of both capacity and cost, we need new mechanisms to leverage NVM, SSDs and HDDs.

Strata [1] fundamentally rethinks longstanding file system design assumptions. Strata is an integrated file system across different storage media that provides low latency (especially for small writes) and high throughput, transparent data migration across different storage media for low-cost, high-performance capacity, and a user-friendly crash consistency model. To achieve these goals, Strata has a novel split of user and kernel space responsibilities. It stores updates to a write-optimized, user-level log in fast NVM, while asynchronously managing read-optimized, long-term storage in the kernel. Strata continues to provide the POSIX API to run unmodified applications.

2. Strata design

Strata has three design principles.

Log operations to NVM at user-level. Closest to the application, Strata employs a user-level library file system (**LibFS**) that uses a fraction of NVM as a private, operational log. The log holds updates to both user data and file system metadata.

Logging in NVM has excellent latency and crash consistency properties. For performance, LibFS directly accesses fast NVM without using the complicated kernel IO path. By leveraging the byte-addressability of NVM, LibFS performs cache-line granularity IO; LibFS blindly appends small writes (e.g., less than a block size) to the log. A traditional file system must read, modify, and write an entire block. To provide user-friendly crash consistency, LibFS *synchronously* updates the log, obviating the need for any `fsync`-related system calls and the attendant bugs of application-level crash consistency protocols [2]. Upon a crash, the kernel easily restores file system state by replaying log entries.

Asynchronously digest and migrate data in kernel. **KernelFS** organizes multiple storage layers as *shared areas* and exposes them read-only to applications. For low-cost capacity, KernelFS transparently migrates inactive data to larger, cheaper media via an operation called **digest**. A digest first happens from an application-local log to a system-shared area, both stored in NVM, and then from one layer to its next (slow and larger) layer. KernelFS digests large batches of operations from the log, coalescing adjacent writes, as well as identifying and eliminating redundant operations. KernelFS begins digestion by first scanning the submitted log and then computing which operations can be eliminated or coalesced. For example, if KernelFS detects an inode creation followed by deletion of the inode, it skips log entries related to the inode. When migrating data among slower storage layers, KernelFS tailors the data format to the properties of the target medium to improve performance. For example, batching allows kernelFS to accumulate large contiguous file blocks when digesting into the read-optimized shared area. Similarly, batching allows kernelFS to write large amounts of data sequentially to SSDs and HDDs to avoid firmware garbage collection overhead [3].

Because data migrates from faster to slower layers, faster layers contain up-to-date data blocks. Thus, LibFS performs hierarchical searches for reading data; LibFS first searches the log and then NVM, SSD, and HDD system-shared area until it finds all data blocks.

Coordinate concurrent accesses with leases. KernelFS supports leases [4] on files and sections of the file system namespace. Similar to their function in dis-

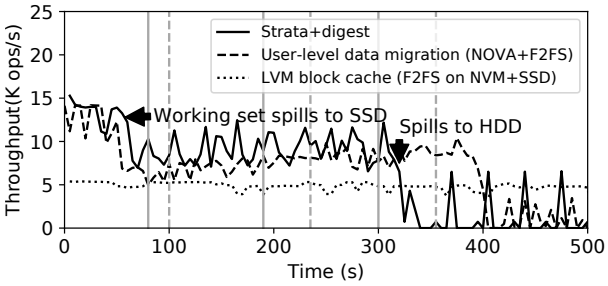

Figure 1. Fileserver throughput on multi-layer storage over time. Vertical lines every 1 million operations.

tributed file systems, leases allow a LibFS exclusive write or shared read accesses. As long as a write lease is held, a thread in a process may write to the leased files or namespace without kernel mediation. A process that holds a write lease gets a revocation notification if another process also wants the write lease. Leases are also revoked when an application is unresponsive and the lease times out. On revocation, LibFS digests all leased data and metadata to the kernel's shared area. As a result, operations to the same namespace made by multiple processes are serialized before or after the lease period.

3. Evaluation

Our prototype of Strata is implemented in 21,255 lines of C code. Our experimental testbed consists of 2 × Intel Xeon E5-2640 CPU, 64 GB DDR3 RAM, 400 GB Intel 750 PCIe-SSD, 1 TB Seagate hard-disk, performed on Ubuntu 16.04 LTS and Linux kernel 4.8.12. We reserve 36 GB of DRAM to emulate NVM. Each application uses 1 GB NVM for the update log. The emulation implements an NVM performance model according to a recent study [6].

File servers require performance and capacity and are thus a good workload to evaluate Strata's data management across multiple storage media. We use a *Fileserver* profile from the Filebench suite, which mimics common behavior of file servers: creating, deleting, appending, and reading a number of files in multiple directories. We configure the Fileserver profile to do 1 MB appends with 1000 files. In this case, the working set starts out operating in NVM, but then grows to incorporate the SSD and HDD. To compare Strata with data migration done by the user-level data migration (UDM), we modify the Fileserver workload to migrate data across multiple storage layers with the same policy as Strata. UDM migrates files instead of blocks, which works well for the Fileserver workload because it mostly performs IO on complete files. UDM uses NOVA [5], F2FS, and EXT4 file systems on the appropriate layers with default mount options. We also compare to Linux's logical volume manager (LVM) cache. LVM can support only two device types: NVM and SSD. We cache SSD blocks in NVM with write-back caching mode, running F2FS on top (out-performing EXT4) in synchronous mode (-o sync) for a fair comparison.

Figure 1 shows the result. Both Strata and UDM start with full throughput on NVM, but UDM demonstrates more jitter. This is because of log garbage collection in NOVA. After 80 seconds, the working set sizes are large enough that data starts migrating to SSD, and it quickly becomes the bottleneck. In this period, digesting is slower than logging so the application stalls on a full log (between spikes in Strata). UDM's throughput drops below that of Strata causing UDM to fall behind (vertical lines). UDM lags because it migrates entire files, rather than individual blocks. Strata's workload grows to include HDD after 300 seconds and throughput drops significantly. This happens 370 seconds in UDM due to its lower throughput. Because the throughput of LVM is low, LVM's working set never grows beyond the size of the SSD during the experiment.

Once the working set spills to SSD, Strata is 2.0× faster than LVM because Strata can leverage byte addressability of NVM using the user-level log, while LVM requires block-level IO in the kernel. LVM also maintains its own persistent meta-data cache which adds IO to every file system operation. Strata's cross-layer approach also outperforms user-level data migration. Strata benefits from combining knowledge of application access patterns and cross-layer block migration. Hence, Strata can maintain frequently accessed meta-data in faster layers to speed up file system data structure traversal.

4. Conclusion

Trends in storage hardware encourage a multi-layer storage topology spanning multiple orders of magnitude in cost and performance. Stata is a cross media file system, providing fast synchronous IO, low-cost capacity, and simple crash consistency.

References

[1] Y. Kwon, H. Fingler, T. Hunt, S. Peter, E. Witchel, and T. Anderson. Strata: A cross media file system. In *SOSP*, 2017.

[2] T. S. Pillai, V. Chidambaram, R. Alagappan, S. Al-Kiswany, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. All file systems are not created equal: On the complexity of crafting crash-consistent applications. In *OSDI*, 2014.

[3] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li. Ripq: Advanced photo caching on flash for facebook. In *FAST*, 2015.

[4] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *SOSP*, 1997.

[5] J. Xu and S. Swanson. Nova: A log-structured file system for hybrid volatile/non-volatile main memories. In *FAST*, 2016.

[6] Y. Zhang and S. Swanson. A study of application performance with non-volatile main memory. In *MSST*, 2015.