Enhance Strata with Lease

Zeyuan Hu, Jianwei Chen, Junkai Liu

University of Texas at Austin

Background

Strata is a cross-media file system that leverages the strengths of one storage media to compensate for weaknesses of another. Its main design principle is to log operations to NVM at user-level (LibFS) and, digest and migrate data in kernel (KernelFS). In dosing so, Strata provides high performance, low-cost capacity and crash consistency.

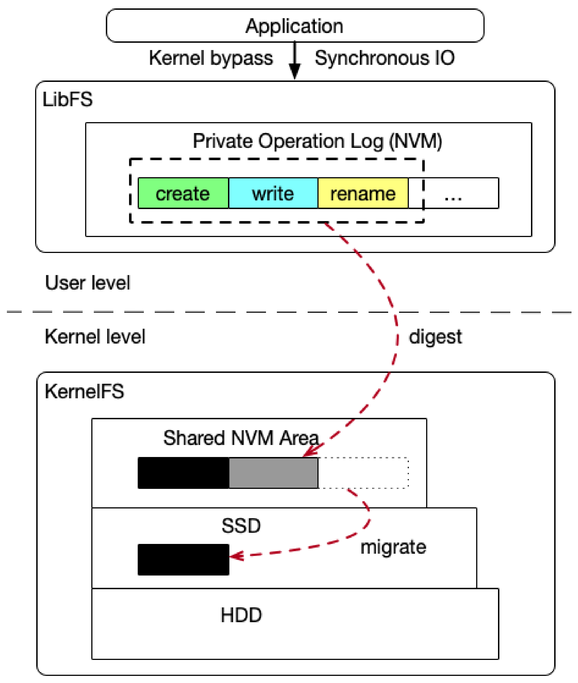

Figure 1: *Strata architecture.*

Problem & Motivation

- Strata implements the file access control for multi-threads within a process. However, for concurrent file access (e.g., read or write of the same file) from multiple processes, there is no mechanism to properly coordinate the access within Strata.
- File lock for each libfs is advisory. We need a mechanism to enforce the file access control.
- File lock is not ideal as one process may crash while holding the lock of the file, which prevents other processes from accessing the file.

Solution

We implemented Lease within Strata that has the following features:

- Support leases on files and directories
- Exclusive writer, shared readers

The architecture of Strata with Lease is shown below.

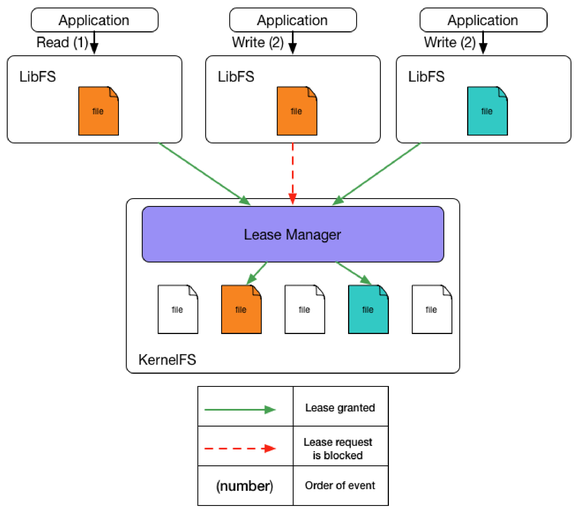

Figure 2: *Strata with Lease architecture.* For any POSIX operation, Libfs will contact lease manager to acquire lease. Lease manager maintains the lease information for a given file path.

As strata uses a transaction mechanism to write application logs, it is expensive to abort the transaction to conform to the correct lease semantics. Therefore, we set a relative long default lease time (2s in our current setting) to make sure most of the read-/write operations would fall within that time frame. To compensate for potential degrade of performance in this setting, we adopt three strategies:

- Allow application voluntarily release of leases after the operation was done
- Allow application to poll the status of the file at a relatively short interval if the lease is denied
- Delegate the complex concurrent process interaction to the Libfs; KernelFS is designed to be simple and fast: deny or accept lease requests.

Technical Highlights

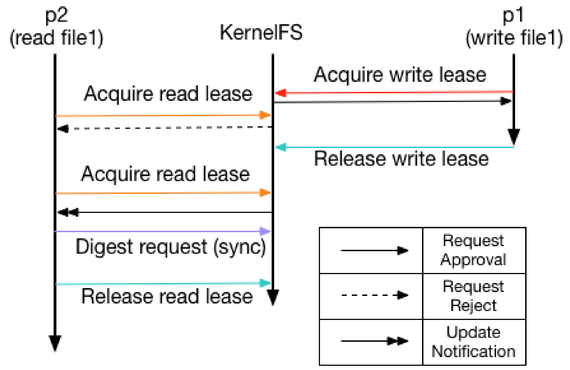

Figure 3: *Read operation initiates digest request for write-intensive workload.* The design favors write-intensive workload as read operation will request digestion before performing read in the concurrent access scenario. For the read-intensive workload, the system can be switched to ask write operation request digestion before releasing its lease.

/mlfs/file1	{ expiration time; state: UNK }
/mlfs/file2	{ expiration time; state: DEL }
/mlfs/file3	{ expiration time; state: CRT }
...	...

Figure 4: *Lease Manager Data Structure.* Lease manager maintains a hash table with file paths as keys. In the entry, the lease manager tracks the expiration time for the lease assigned to each file path. In addition, lease manager also maintains the state of file. States include `UNK, DEL, CRT`, which are used to keep track of file state changes that might be unknown to individual process. For example, a concurrent process accesses a file with `DEL` status will get an update notification, which indicates that the file is deleted by other processes. Libfs will perform digestion and notify application about the change with an error. During digestion, the file states of lease manager will be reset.

Performance

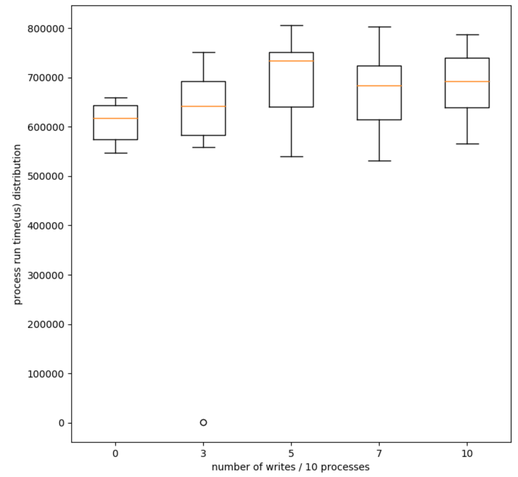

Figure 5: *Process runtime for read-intensive workload vs. write-intensive workload.* Each process read or write 100MB to the 0.45GB simulated NVM. The figure shows that the write-intensive performance does not degrade much compared with the read-intensive performance. This is because the synchronous digestion is done selectively during read operation to maintain file consistency.

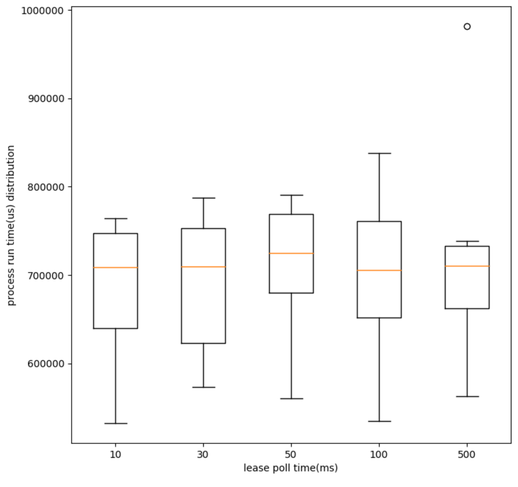

Figure 6: *Process runtime as `poll` time increases.*