LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism

Jongyul Kim[1] Insu Jang[2] Waleed Reda[3,4] Jaeseong Im[1] Marco Canini[5]
Dejan Kostić[3] Youngjin Kwon[1] Simon Peter[6] Emmett Witchel[6,7]
[1]*KAIST* [2]*University of Michigan* [3]*KTH Royal Institute of Technology* [4]*Université catholique de Louvain*
[5]*KAUST* [6]*The University of Texas at Austin* [7]*Katana Graph*

I. Introduction and Challenges

In multi-tenant systems, the CPU overhead of distributed storage services sharing the machine with the applications using them is increasingly a burden to application performance. Due to the stagnation of CPU performance, operators wish to dedicate as many client CPU cycles to applications as possible. However, as storage services incorporate persistent memory (PM) [1], CPU contention has increased. To exploit PM performance, recent PM-optimized distributed file system (DFS) proposals deploy their applications on the machine where PM is installed (client-local PM). In this type of DFS [2], [3], client-local storage management such as parallel data cache eviction, indexing, and log garbage collection consumes several cores on IO-intensive client nodes.

To reduce CPU overhead, offload of disaggregated storage stacks is already commonplace. SmartNICs are popular for this purpose because they provide additional processing power and can implement the data path for disaggregated storage operations. SmartNICs support remote direct memory access (RDMA) [4] that allows access to local and remote PM at byte-granularity. However, none of the existing solutions consider the offload needs of a complete DFS.

We present LineFS, a SmartNIC-offloaded and high-performance DFS with support for client-local PM. LineFS offloads processing-intensive DFS tasks, such as replication, data publication (§ II), and consistency management. LineFS reduces file system fail-over time by providing a fast failure detector and SmartNIC-based recovery mechanism, leveraging the SmartNIC as an isolated failure domain.

Offloading a DFS to a SmartNIC is challenging. File systems are complex, handling highly structured data with sophisticated access protocols for consistency and durability. Wimpy SmartNIC architecture and the PCIe interconnect that separates SmartNIC and host PM implies that a naive offload of DFS operations will be much slower than using host CPUs. To make offload worthwhile, LineFS must hide execution and data access latencies by exploiting opportunities for parallelization, batching, and asynchronous operation.

II. LineFS Design

The main design goals of LineFS include minimizing host performance interference and slowdown from offload. Like Assise [2] and Orion [3], LineFS adopts a *client-local* DFS model that executes DFS functionality on client machines to avoid client-server communication latency for PM access. LineFS nodes consist of two components: *LibFS* and *NICFS*.

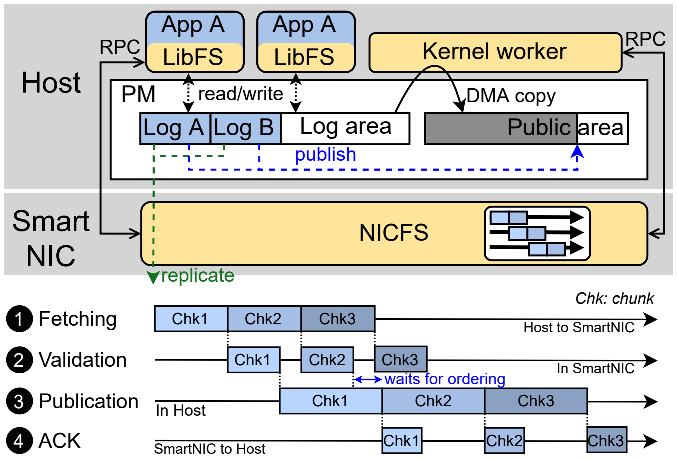

Fig. 1: Components and data path of a LineFS node (top). Publishing pipeline (bottom).

LibFSes are linked to application processes (LineFS clients) running on host cores. NICFS runs on SmartNICs (Figure 1).

Beyond per-node offload, LineFS follows Assise's design closely. Ideas, such as user-level PM IO with per-process LibFSes and update logs, leases [5] as a consistency mechanism, and chain-replication via RDMA are inherited from Assise [2]. LineFS also runs a *cluster manager* to manage DFS node membership, failure detection, and root lease arbitration.

To avoid the overhead for offloading, LineFS follows two design principles: persist-and-publish and pipeline parallelism.

Persist-and-publish. LineFS assigns a fraction of PM as a per-client PM log (cf. Figure 1). LibFSes *persist* data and metadata updates to their private PM logs on the host. The logs are asynchronously *published* to a host-local *public PM* and replicated to remote PM by NICFS. This design enables LineFS to clearly separate PM-latency critical operations from those that can be deferred. After LibFS makes data and metadata durable in the host PM log using fast host cores, NICFS publishes and replicates the updates in the background with SmartNIC cores, saving the host cores from performing file system management tasks.

Pipeline parallelism. LineFS exploits pipeline parallelism to publish and replicate the log. LineFS organizes DFS operations into distinct *execution stages* to construct an execution pipeline. LineFS defines a group of log entries as a *LineFS chunk*, processing each chunk in parallel through the pipeline. LineFS leverages the pipeline to publish and replicate each client-private log while keeping the log data *in order* (*intra-*

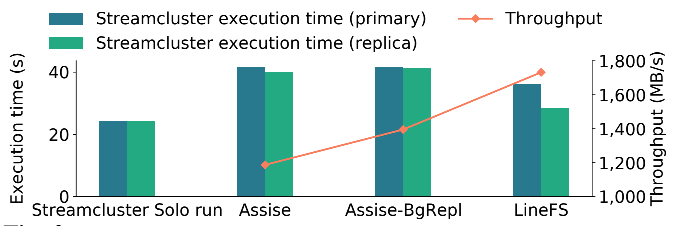

Fig. 2: Impact of DFS co-execution on `streamcluster` execution time (left Y-axis) and DFS throughput (right Y-axis).

client parallelism). At the same time, it processes multiple client logs concurrently by executing the pipeline for each client in parallel (*inter-client* parallelism).

Together, these principles not only avoid overhead but also maintain consistency when offloading. Client logs are a natural way to persist file system updates in order. When publishing and replicating, pipeline parallelism allows LineFS to process data in client log order, providing linearizability and prefix crash consistency [6].

Two pipelines, the publishing pipeline (Figure 1) and the replication pipeline, have to fetch log data to NICFS and validate it. They share the first two stages (*fetching*, *validation*) and after the two, they run their own pipeline stages.

The *persist-and-publish* model allows NICFS to publish the client-private log in the background, while application execution continues. After publishing log entries, LibFS reclaims them to make room for further updates. To amortize PCIe transfer overheads, LineFS batches consecutive updates into *LineFS chunks*. As soon as LibFS has accumulated a single LineFS chunk of updates, it sends an asynchronous RPC request to NICFS to *fetch* the chunk. NICFS fetches a LineFS chunk to the SmartNIC's memory and *validates* it. After passing the validation, NICFS *publishes* the LineFS chunk and *acknowledges* it to LibFS.

Publishing the chunk via PCIe causes excessive latency, stalling the pipeline. Instead, LineFS uses a *kernel worker* in the host operating system to initiate asynchronous host DMA [7] to publish the chunk. Instead of copying PM with host cores, the DMA copy still avoids CPU utilization.

LineFS replicates primary's client-private log to replicas using RDMA, providing availability and strong consistency among replicas. Like other DFSes, `fsync()` guarantees durability and replication of file data and metadata. LineFS uses the SmartNIC to *asynchronously* and *proactively* replicate log entries before LibFS calls `fsync()`. On `fsync()`, LineFS synchronously replicates any remaining log entries. Similar to the publishing pipeline, LineFS parallelizes replication with 4-stages pipeline (*fetching*, *validation*, *transfer*, *acknowledgment*). Additionally, LineFS improves latency by dedicating a SmartNIC core for RDMA processing with busy-polling.

III. Evaluation

Our evaluation testbed consists of 3× Intel Xeon Gold 5218 servers at 2.3 GHz with 16 cores, 96 GB DDR4-2666 DRAM, 6× 128 GB Intel Optane DC persistent memory modules, and Mellanox BlueField SmartNIC (16 ARMv8 A72 cores, 16 GB DRAM, and 25Gbps network bandwidth with RDMA). All nodes run Ubuntu 18.04 with Linux kernel version 5.3.

We compare LineFS with Assise [2], a state-of-the-art DFS that supports client-local PM access. `Assise-BgRepl` additionally replicates in the background before `fsync()` is called with multiple threads and the same 4MB chunk size, akin to LineFS replication but without pipelining. We configure the PM log size to 512 MB for both Assise and LineFS. Both DFSes use 3 nodes; primary, replica-1, and replica-2. We run two throughput benchmarks as DFS clients. Each single-threaded client writes file data to a 12 GB file with 16 KB IO size sequentially and calls `fsync` at the end. We run `streamcluster` from PARSEC 3.0 [8] in all nodes to mimic CPU-intensive jobs running together with DFSes. To stress the host cores, we set the number of threads for `streamcluster` equal to the number of host cores.

Figure 2 presents the `streamcluster` execution time and the microbenchmark throughput. When running `streamcluster` with `Assise`, `Assise` degrades the performance of `streamcluster` by 72% in the primary and 66% in replicas due to resource contention. The primary, where the DFS clients run, uses more host CPU and memory resources, causing a more severe slowdown than on the replicas. `Assise-BgRepl` improves `Assise`'s throughput by 18%, limited by contention with `streamcluster`. `LineFS` shows the best throughput (46% better than `Assise`) due to the pipeline parallelism and the background replication. At the same time, it incurs minimal slowdown of `streamcluster` (49% and 19% slowdown compared to the solo run for primary and replica, respectively) by means of the SmartNIC offloading. This result confirms that *LineFS's design minimizes host performance interference while providing good performance.*

IV. Conclusion

LineFS proposes the persist-and-publish model and pipeline parallelism to offload a PM-optimized DFS to SmartNICs. We implement LineFS on the BlueField SmartNIC and compare it to Assise, a state-of-the-art PM DFS. LineFS achieves 46% better I/O throughput while improving host application performance by up to 40% when host CPU resource is scarce. This work has been published in SOSP 2021 [9].

References

[1] Intel Optane memory. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html

[2] T. E. Anderson, M. Canini, J. Kim, D. Kostić, Y. Kwon, S. Peter, W. Reda, H. N. Schuh, and E. Witchel, "Assise: Performance and availability via client-local NVM in a distributed file system," in *OSDI'20*.

[3] J. Yang, J. Izraelevitz, and S. Swanson, "Orion: A distributed file system for non-volatile main memory and rdma-capable networks," in *FAST'19*.

[4] Rdma consortium. [Online]. Available: http://www.rdmaconsortium.org

[5] C. Gray and D. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," in *SOSP'89*.

[6] Y. Wang, M. Kapritsos, Z. Ren, P. Mahajan, J. Kirubanandam, L. Alvisi, and M. Dahlin, "Robustness in the salus scalable block store," in *NSDI'13*.

[7] T. Le, J. Stern, and S. Briscoe. Fast memcpy with SPDK and intel I/OAT DMA engine. [Online]. Available: https://software.intel.com/en-us/articles/fast-memcpy-using-spdk-and-ioat-dma-engine

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *PACT'08*.

[9] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism," in *SOSP'21*. [Online]. Available: https://doi.org/10.1145/3477132.3483565