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

Jongyul Kim
KAIST

Insu Jang*
University of Michigan

Waleed Reda
KTH Royal Institute of Technology
Université catholique de Louvain

Jaeseong Im
KAIST

Marco Canini
KAUST

Dejan Kostić
KTH Royal Institute of Technology

Youngjin Kwon
KAIST

Simon Peter
The University of Texas at Austin

Emmett Witchel
The University of Texas at Austin
Katana Graph

ABSTRACT

In multi-tenant systems, the CPU overhead of distributed file systems (DFSes) is increasingly a burden to application performance. CPU and memory interference cause degraded and unstable application and storage performance, in particular for operation latency. Recent client-local DFSes for persistent memory (PM) accelerate this trend. DFS offload to SmartNICs is a promising solution to these problems, but it is challenging to fit the complex demands of a DFS onto simple SmartNIC processors located across PCIe.

We present LineFS, a SmartNIC-offloaded, high-performance DFS with support for client-local PM. To fully leverage the SmartNIC architecture, we decompose DFS operations into execution stages that can be offloaded to a parallel datapath execution pipeline on the SmartNIC. LineFS offloads CPU-intensive DFS tasks, like replication, compression, data publication, index and consistency management to a SmartNIC. We implement LineFS on the Mellanox BlueField SmartNIC and compare it to Assise, a state-of-the-art PM DFS. LineFS improves latency in LevelDB up to 80% and throughput in Filebench up to 79%, while providing extended DFS availability during host system failures.

*Work done while at KAIST.

SOSP '21, October 26–29, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8709-5/21/10.
https://doi.org/10.1145/3477132.3483565

CCS CONCEPTS

• **Information systems** → **Distributed storage**; *Storage class memory*; • **Social and professional topics** → **File systems management**; • **Networks** → **Network adapters**; • **Computer systems organization** → **System on a chip**; *Availability*.

KEYWORDS

Distributed file system, SmartNIC offload

ACM Reference Format:
Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. 2021. LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism. In *ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21), October 26–29, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3477132.3483565

1 INTRODUCTION

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 [27, 36, 41]. Due to the stagnation of CPU performance [49], operators wish to dedicate as many client CPU cycles to applications as possible. However, as storage services incorporate persistent memory (PM) [5], CPU contention has increased. Various recent storage system proposals for PM use client-local storage management. For example, the Assise distributed file system (DFS) conducts parallel data eviction when per-process caches fill to capacity [18]. Similarly, Orion periodically creates radix trees to index log-structured file data, performs log garbage collection, and executes client arbitration protocols when data is shared [60]. These processes consume several cores on IO-intensive client nodes (§2.1).

To reduce CPU overhead, offload of disaggregated storage stacks is already commonplace [9, 10]. SmartNICs [2, 3, 8] are popular for this purpose because they can implement the data path for disaggregated storage operations. SmartNICs support both block-level remote access protocols, such as NVMe over fabrics (NVMe-oF) [10], and remote direct memory access (RDMA) [12]. RDMA allows access to remote PM at byte-granularity and recent work has extended it to better support remote file access [61], chain replication [36], and multi-step data access operations via in-network processing [53]. However, none of the existing solutions consider the offload needs of a complete DFS.

In this paper, we present LineFS, a SmartNIC-offloaded, high-performance DFS with support for client-local PM. LineFS offloads processing-intensive DFS tasks, such as replication, data publication, and consistency management. Leveraging spare SmartNIC processing capacity, LineFS reduces network utilization by conducting file- and operation-specific (de)compression. Finally, 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 high-performance 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. We propose a *persist-and-publish* model that separates PM storage operations on the critical-path from operations that can execute asynchronously. To speed up asynchronous publishing operations, while providing strong file system consistency properties, such as linearizability and prefix crash consistency [24, 57], we introduce a *parallel data-path execution pipeline*. We decompose DFS operations into distinct execution stages that operate in a parallel pipeline. The pipeline keeps operations in order for consistency. To avoid pipeline execution stalls, we prefetch data across PCIe and organize host data structures into *pipeline chunks* that can be transmitted in bulk and processed in parallel.

We make the following contributions:

- We present the design of LineFS, a SmartNIC-offloaded, high-performance DFS with support for client-local PM. We describe the challenges of efficient DFS SmartNIC offload and show how to make it efficient by parallelizing DFS functionality that was sequential in previous systems.
- To demonstrate the benefits of LineFS, we apply it to Assise, a state-of-the-art DFS using client-local PM. Our implementation uses the Mellanox BlueField SmartNIC [2]. BlueField allows us full control over data and processing placement across SmartNIC and host, from any machine.
- We compare LineFS performance to Assise (our baseline, replicating upon each `fsync()`), Assise replicating in the background, as well as Assise+Hyperloop (offloading replication as in Hyperloop [36]). In the throughput benchmark, LineFS outperforms all Assise variants by at least 41% and saturates the network. In addition, we execute application benchmarks (LevelDB and Filebench). LineFS shows 80% and 27% better LevelDB latency than Assise for sequential and random insert, respectively. LineFS exhibits 79% higher throughput than Assise in Filebench.
- We showcase the ability of LineFS to maintain a high level of performance even in the presence of an interfering, CPU-intensive workload (streamcluster), whose performance is only mildly impacted by LineFS's presence. In contrast, streamcluster takes longer to execute in the presence of Assise by up to 43%, and LineFS performance is 227% better than that of Assise in this case.
- In addition to demonstrating quantitative benefits of LineFS, we highlight its ability to make qualitative improvements to a DFS by enabling new functionality. In particular, LineFS makes it possible to compress data on the fly while running a sorting benchmark. Doing so results in up to 11% better performance than the no-compression case, all while saving considerable network bandwidth (up to 72%).
- Finally, we demonstrate LineFS's ability to increase system availability by conducting a Varmail experiment with host failure. Even during failure, LineFS enables Varmail to maintain similar throughput levels across the replication chain, as seen without failures.

2 BACKGROUND

Like any operating system service, DFSes consume CPU and memory resources. This consumption was insignificant when networks and storage devices were slow relative to the performance offered by CPUs. Currently, the performance of networks and storage devices is increasing while CPU performance has stagnated in both per-core performance and number of cores. Thus, CPU cycles and memory bandwidth have become precious commodities. In particular, modern PM-optimized DFSes share CPU and memory resources with applications in an effort to reduce PM access latency. Unfortunately, the low-latency access provided by these systems comes with the cost of CPU competition with applications.

We provide an overview of the design of PM-optimized DFSes and the different ways in which they interfere with host CPU and memory performance (§2.1). We then present SmartNIC offload of DFS functionality as a solution to the interference problem, while outlining the unique challenges of DFS offload to SmartNICs (§2.2).

# of proc.	Throughput (GB/s)				CPU utilization			
	25GbE		100GbE		25GbE		100GbE	
	Assise	Ceph	Assise	Ceph	Assise	Ceph	Assise	Ceph
1	0.38	1.23	0.63	1.26	62%	95%	101%	96%
2	0.74	1.34	1.12	1.51	119%	126%	201%	146%
4	1.30	1.40	1.98	1.56	225%	141%	380%	211%
8	1.32	1.41	2.22	1.60	224%	176%	509%	211%

Table 1: CPU utilization of `Assise` and `Ceph` for different numbers of benchmark processes and network speeds. 100% = 1 core.

2.1 Interference in PM-optimized DFSes

As PM gains popularity, new DFS designs are appearing that are optimized to leverage PM's performance. A main design principle of these DFSes is to utilize PM storage that is located on the same machine that executes the applications using it—called *client-local* PM storage. Client-local storage can reduce PM file access latency by orders of magnitude. Assise [18] and Orion [60] are two recent DFSes that use client-local PM storage.

While client-local PM storage can improve file system performance versus client-server designs, it also requires file system management tasks local to the client. These file system tasks compete for memory and compute resources with applications. This kind of concurrent execution causes performance interference and takes away precious resources from application execution. To identify DFS resource consumption, we measure client CPU utilization of a client-local DFS, Assise, and compare it with a client-server DFS, Ceph. Table 1 shows the result. Each client writes a 24 GB file with 4 KB IO size. We can see that both DFSes require a significant amount of client CPU cycles. However, Assise uses up to 60% and 2.4× more cycles than Ceph on 25GbE and 100GbE networks. Assise needs more CPU to perform file system management tasks as the network bandwidth increases. Using a 25 GbE NIC, Assise occupies 2.24 host cores to support 8 clients while Ceph occupies 1.76 cores. Using a 100 GbE NIC, Assise uses 5.09 host cores for 8 clients, while Ceph uses only 2.11.

We identify the following four file system tasks of client-local DFSes that cause high CPU utilization and performance interference:

I1. Data movement and indexing. The file system invokes threads for data movement and organization. For example, Assise's SharedFS creates many threads to apply file system updates to PM and to create indexing structures to optimize later reads.

I2. Replication. For availability, client-local distributed file systems replicate data among client nodes, requiring invocation of DFS services that can accept and persist data per client replica. Replication consumes memory and CPU resources.

I3. Remote storage access requests. Not all file system requests can be served locally, requiring a remote node to serve these requests with low latency. High-priority file system processing interrupts co-running applications, involving high context switching costs when done frequently.

I4. File system consistency. To allow multiple clients access to shared file system state with consistency guarantees, distributed file systems have to coordinate these clients, requiring the invocation of coordination mechanisms on multiple client nodes, often with low latency requirements.

Each of these tasks may execute concurrently, consuming a variable amount of shared compute and memory resources. The interference presents several challenges:

C1. Unpredictable application performance. When CPU and memory resources are consumed by file system management threads, application execution performance is impacted. For example, application threads need to wait for CPUs to become available or are slowed down if not enough memory bandwidth is available.

CPU contention is a particularly grave performance problem for parallel applications that synchronize frequently, such as via barriers. Spontaneous unavailability of CPUs due to file system task execution can create stragglers that disproportionately slow down the entire parallel computation as the application waits for all threads to pass a barrier.

C2. Unpredictable file system access latency and tail-latency. Shared resources also impact file system performance. In particular, file system access latency and tail-latency can increase when latency-critical operations are delayed because CPUs are busy executing application threads.

This problem is exacerbated with PM. Due to the low access latency of PM, critical file system operations often need to finish within microseconds. Even when the CPU scheduler gives priority to critical file system tasks, these latencies can easily be inflated by several orders of magnitude due to context switching and dispatch overheads.

C3. Reduced throughput. A potential solution is to partition available CPU and memory resources among file system services and applications. However, partitions have to be provisioned for peak utilization to support high throughput. CPUs in particular are a scarce resource in today's systems and under-provisioned partitions reduce throughput for either the file system or the applications.

2.2 DFS Offload to SmartNICs

Offload of DFS services to a SmartNIC can be a solution to the aforementioned problems. SmartNICs are versatile compute platforms that sit in the network data-path of applications. Their position makes them well-suited for offload of networked services, including distributed file systems. Indeed,

various SmartNIC platforms [2, 3] provide features that aid in particular the offload of disaggregated storage protocols, e.g., NVMe-oF [10]. However, SmartNIC offload is not a panacea. Several challenges make the offload of distributed file system functionality particularly difficult.

Increased latency for host memory access. SmartNICs are PCIe expansion cards. PCIe is a high-latency interconnect relative to the DDR memory interconnect that CPUs enjoy to access memory. Accessing PM via DDR from a host CPU incurs a latency on the order of 100 ns while accessing PM via PCIe has a latency of several μs—an order of magnitude difference. Thus, accessing PM and related file system state stored in host memory from the SmartNIC has high overhead and such access needs to be minimized or the latency needs to be hidden.

Wimpy execution environment. A typical SmartNIC power envelope is 25W. Compared to host CPUs that have an order of magnitude larger power envelopes, there is no room for powerful memory or processing features. SmartNICs typically opt for a wimpy processor architecture, with many low-frequency cores and a small amount of cache memory.

Mellanox BlueField. We explore DFS offload using a Mellanox BlueField MBF1M332A SmartNIC [2]. This is a 2×25GbE SmartNIC that contains an ARMv8 Cortex-A72 processor with 16 cores, running at 800MHz. Each Cortex-A72 core has a 32KB L1 data cache. Two cores share 1MB of L2 cache and all cores share 6MB of L3 cache, as well as 16GB of memory, which may be DRAM or NVDIMM-N PM. BlueField runs Linux on the Cortex-A72.

BlueField is an off-path SmartNIC [41]. An RDMA switch on the SmartNIC is capable of directly accessing SmartNIC and host memory. The switch can be configured to forward RDMA requests according to various rules. We configure the switch to treat the Cortex-A72 as a discrete host, with its own MAC and IP address. This configuration allows the file system full control, from any machine, over where to transfer data and whether to interact with the SmartNIC or the host.

Treating the SmartNIC as a discrete host running Linux focuses our work on the interconnect and wimpy execution environment challenges described in this section, rather than low-level acceleration. This is intentional. File systems are complex software, handling highly structured data with sophisticated access protocols for consistency and durability. CPU architectures capable of running full OSes, such as ARMv8, are well-suited for file system execution. Lower-level, in-path architectures that are optimized to process individual network packets, such as NPU or FPGA-based SmartNICs, would have unnecessarily complicated our offload design.

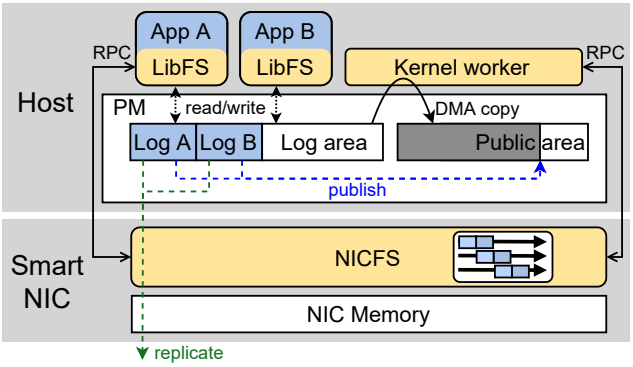


Figure 1: Components and data path of a LineFS node.

3 LINEFS DESIGN

LineFS is a DFS designed for a cluster of nodes that use PM and RDMA. LineFS minimizes host CPU overheads by carefully offloading DFS operations to SmartNICs. LineFS has the following design goals.

- **Minimize host performance interference.** Performance and performance predictability of applications and storage systems suffer from competition for shared resources. CPUs are one of the most constrained resources in modern cloud servers [27, 40–42, 49, 55]. LineFS must minimize competition for CPUs among applications and the DFS. To do so, LineFS offloads DFS functionality to SmartNICs.
- **Minimize slowdown from offload.** LineFS must minimize the performance impact of DFS offload to SmartNICs. To do so, LineFS rethinks the file system data path to exploit fine-grained data parallelism. LineFS executes the data path in the background to allow foreground host computation to continue unaffected.
- **Leverage data-path processing opportunities.** LineFS shall leverage the SmartNIC's data-path processing capability to opportunistically perform semantic-aware data transformations. We take advantage of spare SmartNIC processing capacity to perform data compression, saving network bandwidth for replication.
- **Improved availability.** LineFS shall also leverage the SmartNIC's data-path processing capability to improve DFS availability. While SmartNICs and host CPUs share host power, SmartNICs provide independent execution environments and can continue operating, even if the host operating system has failed.

LineFS architecture. Like Assise [18] and Orion [60], LineFS adopts a *client-local* DFS model that executes DFS functionality on client machines to avoid client-server communication latency for PM access. In addition, LineFS carefully distributes DFS components among host and SmartNICs. LineFS nodes consist of two components: *LibFS* and *NICFS*. LibFSes are linked to application processes (LineFS clients)

running on host cores. NICFS runs on SmartNICs. Figure 1 shows the overall design of a LineFS node.

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 [32] as a consistency mechanism, and chain-replication via RDMA are inherited from Assise [18]. LineFS also uses ZooKeeper [1] as a *cluster manager* to manage DFS node membership, failure detection, and root lease arbitration (§3.4). We initially attempted SmartNIC offload of Assise's per-node SharedFS component but found that, without the design principles presented in this paper, throughput dropped by more than 30× versus the host-based version due to inefficient SmartNIC execution.

3.1 Design Principles

Naive DFS offload to a SmartNIC leads to serious performance degradation due to the wimpy SmartNIC architecture and data movement across PCIe. 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 *persist-and-publish* design, which is inspired by Strata's logging and digesting approach [37], 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 takes advantage of two different types of parallelism: *intra-client* and *inter-client*. Intra-client parallelism leverages the pipeline to publish and replicate each client-private log while keeping the log data *in order*. LineFS processes multiple client logs concurrently by executing the pipeline for each client in parallel. LineFS coordinates the pipeline to linearize shared updates (§3.4).

Pipeline parallelism provides a convenient way to manage the degree of parallelism [46]. To execute each pipeline stage, NICFS assigns a thread for each stage to SmartNIC cores from a single thread pool created at start time. When a LibFS log grows to the size of a LineFS chunk (e.g., 4 MB), LibFS sends an RPC to NICFS to start the pipeline. NICFS monitors the time taken for each stage. If one stage becomes a bottleneck (e.g., its wait queue grows beyond 5 entries), NICFS *dynamically* assigns more threads to process the stage.

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 [24, 26, 57].

3.2 Low-latency PM IO via LibFS

To provide low latency PM IO, LibFS persists data to an operational log in host PM. Operational logging provides a compact way for persisting (meta-)data in PM [37, 59]. LineFS assigns a fraction of PM to each LibFS as a persistent write log. LibFS intercepts the application's POSIX file system calls and writes file data and metadata to the PM log. For example, on a `create()` system call, LibFS writes updated inodes and directories to the PM log. A log is efficient for PM because the sequential performance of PM is high and log appends are sequential.

Reading is a two-step process: 1) LibFS searches in its client-private log (the data is not yet published) and 2) if the data is not found in the log, it searches public PM. The read path is performed in the host CPU without involving NICFS. Writing requires only asynchronous communication with NICFS to publish and replicate the client-private logs (§3.3).

3.3 Pipelining DFS operations

NICFS runs two different pipelines: the publishing pipeline and the replication pipeline. The publishing pipeline (§3.3.1) consists of four stages: *fetching*, *validation*, *publication*, and *acknowledgment*. The replication pipeline (§3.3.2) also consists of four stages: *fetching*, *validation*, *transfer*, and *acknowledgment*. Both pipelines have to fetch log data to NICFS and validate it. To avoid redundant data movement, the publishing and replication pipelines share the first two stages—i.e., they operate on the same data. After the first two, they run their own pipeline stages.

3.3.1 Publishing Client Logs. NICFS publishes client-private log entries to public PM in the background. After publishing log entries, LibFS reclaims them to make room for further updates. Publishing log entries involves memory-intensive data movement from the client-private log to public PM.

Rationale. LineFS offloads publication of the log to a SmartNIC for four reasons: (1) to reduce occupancy and load on the host CPU for data movement; (2) to reduce the overhead of scheduling and context switches due to DFS services; (3) to reduce head-of-line blocking on log writes when logs are full and host CPU resources are scarce; (4) to enable ancillary tasks, such as compression, without CPU contention.

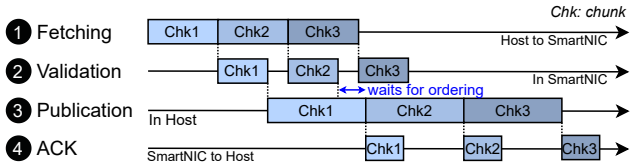


Figure 2: Publishing with pipeline parallelism.

Challenges. There are two challenges for efficient offload to the SmartNIC: (1) High PCIe latency; (2) Permission checks (§3.4) and validation (e.g., prevent directory cycles in the DFS namespace) consume enough computational bandwidth to saturate the relatively wimpy SmartNIC processors.

The dual challenges of communication latency and computational load suggests that overlapping these latencies can help reduce their effect on end-to-end system performance.

Approach. The persist-and-publish model allows NICFS to publish the client-private log in the background, while application execution continues. 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 start publishing the chunk.

As shown in Figure 2, the publishing pipeline consists of four stages (with the primary resource for the stage in parentheses): fetching (PCIe), validation (computation), publication (PCIe and computation), and acknowledgment (PCIe latency). 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 thread* in the host operating system to initiate asynchronous host DMA [38] to publish the LineFS chunk. Instead of copying PM with host cores, the DMA copy still avoids CPU utilization.

Data-path processing opportunities. NICFS can add additional pipeline stages to utilize any spare SmartNIC processing cycles. For example, we add a stage to perform semantic-aware compression called *coalescing* [37, 45]. Coalescing reduces the amount of published data by skipping unnecessary log entries, thereby reducing write amplification and improving the lifetime of PM. The stage scans a fetched chunk to find a temporarily durable write pattern (e.g., creating and then deleting the same file). If the stage detects the pattern, it removes the redundant log entries before building a copy list for the kernel worker. To find a coalescing opportunity, the stage scans log entries, which is also needed for validation. Hence, we can execute the validation and coalescing stages together in the same core to exploit CPU cache locality.

3.3.2 Replication. LineFS chain-replicates [51] the client-private log to a number of replicas using RDMA, providing availability and strong consistency among replicas. Like other DFSes, `fsync()` guarantees durability and replication of file data and metadata. Along a replication chain, each replica persists a primary's client-private log to its local log. When the primary receives ACKs from all replicas, `fsync()` returns. Thereby, the primary and all replicas have the same view of updates.

With strong consistency, replication latency directly impacts DFS write performance [36]. In turn, two factors primarily affect replication latency: (1) Delays in network (RDMA) request processing and (2) CPU contention between DFS replication operation and co-running applications. Host-based approaches to providing low replication latency avoid these factors by pre-posting network operations to RDMA send-receive queue pairs (QPs) and polling for completion in a busy loop. To guarantee sustainable throughput and latency, pre-posting and polling must not be delayed and thus they must run on isolated CPUs.

Busy polling on the host CPU is not a viable option for multi-tenant systems because it must reserve many host CPUs just for replication processing. Unfortunately, blocking for replication introduces context switch and dispatch overheads that delay network request processing, while interference from co-running applications may cause scheduling policies and cache effects to further delay replication processing. PM-optimized DFSes exhibit less than 10μs latency for small replicated updates [18], intensifying the impact of any interference.

Rationale. We have to find another way to provide consistent DFS replication performance, even when replica CPUs are highly utilized by co-tenant applications. At the same time, DFS replication should not interfere with co-running applications on these replicas. Offload to a SmartNIC can provide us with both benefits. We can busy poll for network events and process DFS replication operations on the SmartNIC with low latency and without interference with co-running applications on the host. LineFS offloads replication for these reasons.

Challenges. Naive SmartNIC offload, where every RDMA connection has an independent polling thread, would overload the SmartNIC CPUs. It does not scale to many connections. At the same time, replication processing needs to be parallelized and scale to many SmartNIC CPUs to achieve low latency and high throughput on wimpy SmartNIC architecture. We describe solutions to each challenge.

Approach (scalable, low latency RDMA request processing). To realize low latency RDMA request processing, yet scale to many connections, NICFS partitions DFS requests into two types: low latency and high throughput. They use

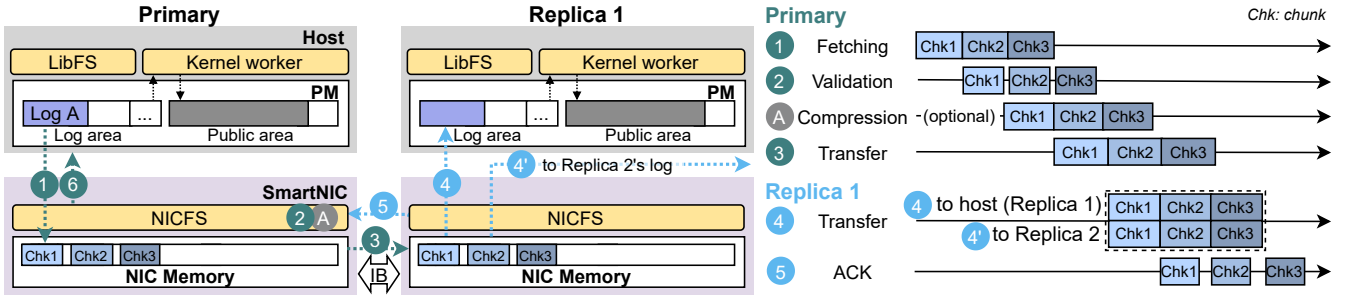


Figure 3: LineFS IO path and replication pipeline with 3 nodes (2 shown).

different network ports, so LibFS performs separate RPC requests according to the use. For the low latency connection, NICFS dedicates a thread for busy polling, pinned to a SmartNIC core. For the high throughput connections, NICFS maintains a worker thread pool invoked when an event occurs (e.g., replication acknowledgments). NICFS uses the low latency connection for latency-sensitive operations (e.g., `fsync()` notification and lease operations) and the high throughput connection for data-intensive operations, like replication and publication. NICFS also multiplexes RDMA operations from multiple LineFS clients to reduce the number of QPs; for RDMA scalability, having a small number of QPs is necessary to avoid NIC cache thrashing [23] and further reduces the busy loop thread count.

Approach (replication). To further reduce replication latency, LineFS uses the SmartNIC to *asynchronously* and *proactively* replicate log entries (at the granularity of chunks) before LibFS calls `fsync()`. On `fsync()`, LineFS synchronously replicates any remaining log entries. Like the publish operation, NICFS uses pipeline parallelism to accelerate asynchronous replication. The replication pipeline consists of four stages: fetch, validation, transfer, and acknowledgment (ACK). Recall that the first two stages are identical to the publishing pipeline and the two pipelines share these stages for efficiency.

Figure 3 shows LineFS's replication IO path and pipeline. At the primary's NICFS, a pipeline chunk is first fetched (1) and validated (2). Then, NICFS transfers the chunk to the next replica's NICFS (3). After receiving the chunk, the replica asynchronously copies the chunk to its local PM log (4), and, *in parallel*, transfers the chunk to the next replica (4'). Finally, because the last replica does not have to conduct any further data replication, the penultimate replica (replica 1 in Figure 3) can directly transfer the chunk to the last replica's host PM log in step 4', saving a SmartNIC memory copy. Each replica sends an ACK to the primary, after copying the pipeline chunk to the local PM log (5). These steps happen proactively in the background and LibFS does not need to be informed.

By design, NICFS parallelizes the replication pipeline; e.g., replicating Chk1 (3), while validating Chk2 (2), and fetching Chk3 (1) in parallel. Similarly, replica NICFSes transfer each pipeline chunk to the next SmartNICs' memory (4') in the replication chain (except for the last replica) and, in parallel, copy the chunk to their host-local PM log (4). The cost of copying is hidden by overlapping it with transferring the chunk to the next replica (4 and 4' happen concurrently in Figure 3).

On `fsync()`, the primary's NICFS fetches any client-private log entries that are not yet replicated and synchronously replicates them using the replication pipeline. Unlike asynchronous replication, synchronous replication uses the low latency RDMA connection to transfer the log entries quickly. When done, the primary NICFS acknowledges successful replication to LibFS (6) and `fsync()` can return.

Data-path processing opportunities. Optionally, NICFS can configure a compression stage (A) before transferring data in the replication pipeline to save network bandwidth. The compression stage consumes a large amount of SmartNIC's CPU resources, so NICFS monitors the wait queue length at each stage of the pipeline. If the compression stage becomes a bottleneck, NICFS opportunistically disables the stage.

3.4 Shared File Management

LineFS linearizes concurrent accesses to shared state using leases [32]. Leases provide single-writer, multiple-reader access to files and directories. LineFS follows Assise's lease management design, but offloads lease arbitration to NICFS. LibFSes acquire leases from the NICFS lease manager on the SmartNIC, instead of a host-based SharedFS. Like in Assise, lease management is initially rooted in the cluster manager and then delegated to NICFS instances, upon LibFS request. Once a lease is granted, LibFS can access the file/directory associated with the lease without further synchronization, until the lease expires or is revoked. NICFS accepts published log entries only if LibFS holds the correct leases for file or directory updates (as checked in the validation stage).

To provide crash consistency, the DFS needs to record granted leases, requiring persistence and replication. Replication and persistence have high overhead compared to lease arbitration and thus increase the latency of lease operations. To reduce latency on the critical path, LineFS provides filesystem access concurrently with recording leases. When granting a lease, NICFS updates lease state only in SmartNIC memory and then carries out host PM persistence and replication asynchronously in the background, while the application continues DFS operation using the granted lease. This does not affect crash consistency, as NICFS waits until all leases are persisted and replicated when `fsync()` is called.

3.5 Extended NICFS Availability

Software crashes are among the most common failures of data center servers [21, 28, 33]. Among these, the most interesting software failure from a NICFS perspective is the failure of the host OS. In this case, the host, including DFS clients on the host, ceases operation and cannot function as a DFS primary anymore. However, if the host is a replica for other primary nodes, its service can be offloaded entirely to NICFS and kept available, even if the host has crashed. Isolated NICFS operation allows LineFS to mask host fail-over time, improving DFS availability.

To detect host failure, NICFS consistently monitors the host kernel worker. If NICFS detects that the kernel worker is not responding, it deems the host down and switches to isolated NICFS operation. Isolated NICFS operation continues to carry out replication and publication services via RDMA across PCIe, keeping the DFS node available while the host operating system is undergoing maintenance.

Switching in and out of isolated NICFS operation is seamless. Client log and public areas in host PM must remain in place across host OS crashes. This is already a requirement for DFS recovery after a host crash [18]. Any interrupted kernel worker publication operation is restarted by NICFS. This is done without any data loss, as publication is idempotent. When the kernel worker becomes available again, NICFS can switch out of isolated operation and submit future copy requests to the worker.

3.6 Discussion

We discuss how LineFS provides access control, fail-over, and recovery, while offloading to NICFS.

Access control. To open a file, LibFS must send an open request to NICFS. NICFS checks file permissions and, if access is granted, sends an RPC request to the kernel worker running in the host kernel. On receiving the RPC, the kernel worker allows LibFS to perform `mmap()` of file, directory, and file index pages in public PM. LibFS must not directly modify the pages, so the kernel worker maps the pages read-only. Instead, all updates go to the per-process update log, where NICFS can validate them upon publication.

NICFS fail-over. The cluster manager sends heartbeat messages to each alive NICFS every second. When the cluster manager detects a NICFS failure, it immediately expires all leases assigned to the failed NICFS and ensures a live NICFS replica takes over the lease management. When a NICFS fails in our current prototype, the host kernel worker is informed by the cluster manager. The kernel worker, in turn, informs all local LibFSes via a signal to return appropriate error codes on file system access.

NICFS fail-over on a SmartNIC crash is a possible alternative. If the host has a redundant (potentially ordinary) RDMA NIC, LineFS can perform a fail-over to a replica NICFS. The replica NICFS becomes the new primary NICFS for the host and takes over replication and lease management. LibFSes re-acquire leases from the replica NICFS (lease state is replicated) and continue operation, while the local NICFS recovers. We leave this for future work.

Recovery. To track file system progress during NICFS downtime, the cluster manager maintains an epoch number. The epoch number increments on node failure and recovery. Once the epoch number is changed, the cluster manager notifies all alive NICFSes and each NICFS persists the epoch number to PM. All NICFSes have a replicated history bitmap that records what inodes have been updated during each epoch.

Once a failed NICFS restarts, it registers with the cluster manager and starts recovery. Upon recovery, NICFS reads the persisted epoch number and requests the history bitmap from an online replica. To synchronize with the current file system progress, NICFS fetches from the replica all inodes that have been recorded between its persisted epoch and the current epoch. Local update logs that touch recovered inodes are invalidated. As future work, the recovery process could be optimized to make it more fine-grained.

4 IMPLEMENTATION

We implement LineFS in x86 Linux hosts and ARM-based Mellanox BlueField SmartNICs. LineFS is written in C with 25,827 lines of code (LoC) for LibFS and 22,538 LoC for NICFS. LineFS uses Intel Optane DC persistent memory modules in App-Direct mode, which expose PM as physical memory. The App-Direct mode allows LineFS to map PM directly, so LineFS directly persists data and metadata via the PMDK library [11], without involving the OS.

LibFS is implemented as a shared library, dynamically linked into each client process' address space. LineFS does not require any application modification. LibFS intercepts filesystem POSIX system calls to persist data and metadata to the client-private log. Currently, LibFS supports 21 system calls. LibFS `mmap()`s the client-private log in contiguous

virtual memory (by default 512 MB) and registers it as an RDMA region to communicate with NICFS.

NICFS is a process running on the SmartNIC's Linux OS. NICFS includes RDMA communication and the file system as independent layers. Using the RDMA layer, the file system layer communicates with local LibFSes and NICFSes in other nodes. NICFS uses one-sided RDMA to fetch client log headers and entries. The file system layer caches inodes, directory, and file indexes (e.g., extent tree) in SmartNIC's DRAM to avoid frequent access to host PM via PCIe. The kernel worker is implemented as a Linux kernel module. The kernel worker publishes client-private logs to the public PM area using the Intel I/OAT DMA engine [38].

Fast read (LibFS). To efficiently search file data in the update log, LibFS has an in-memory hash table that locates it. The hash table does not need to be persisted. LibFS accesses file blocks in public PM after passing a permission check (§3.6). LibFS searches the file data in public PM via per-file extent trees [45]. LibFS does not cache extent trees in DRAM because 1) it has better performance [48], and 2) it does not require maintaining consistency between public PM and DRAM cache.

Replication flow control (NICFS). The memory capacity of our SmartNIC is small (16 GB). When NICFS replicates multiple update logs simultaneously, the SmartNIC may run out of memory. To avoid running out of memory, NICFS consistently monitors memory consumption. When SmartNIC memory consumption is higher than a high watermark (e.g., 70%), NICFS temporarily stops further update log replication, draining the current replication pipeline. Once memory utilization drops below a low watermark (e.g., 30%), NICFS resumes replication.

Asynchronous DMA (kernel worker). To perform asynchronous DMA, we use the Linux I/OAT DMA kernel driver from the kernel worker. NICFS batches memory copy requests by building a memory copy list (preserving order) and then sends an RPC to the kernel worker. The kernel worker issues DMA requests in the order of the memory copy list and sleeps until it receives completion notifications from the kernel I/OAT driver.

5 EVALUATION

We evaluate LineFS to validate our design principles when offloading DFSes to SmartNICs. We compare LineFS with Assise [18], a state-of-the-art DFS that supports client-local PM access. Our evaluation answers the following questions:

- What latency and throughput can LineFS achieve on an idle and on a busy cluster with co-running applications? How does it compare to Assise? How do the various parallel pipelines contribute to LineFS performance? (§5.2)
- How do various levels of DFS offload contribute to performance isolation? (§5.2.4, §5.3)
- What data-path processing opportunities exist? (§5.4)
- By how much can LineFS improve DFS availability (§5.5)?

5.1 Experimental Setup

Our evaluation testbed consists of 3× dual-socket Intel Xeon Gold 5220R servers at 2.2 GHz with 48 cores (no hyperthreading), 96 GB DDR4-2666 DRAM, and 768 GB PM (6× 128 GB Intel Optane DC persistent memory modules). All nodes run Ubuntu 18.04 with Linux kernel version 5.3.

SmartNIC. We deploy a Mellanox BlueField MBF1M332A in each node. The SmartNIC has 16× ARMv8 A72 cores with 6 MB shared L3 cache, 16 GB DRAM, and 25Gbps network bandwidth with RDMA. The SmartNICs are connected to a 100 GbE switch and we use RoCE [13] for RDMA. The measured memory bandwidth of our SmartNIC is 10 GB/s and network goodput (measured by our file benchmark) is 2.2 GB/s.

System configuration. We choose Assise as a baseline system because it is a client-local DFS like LineFS. Orion also supports the client-local model, but its source code is not available. We do not compare LineFS with client-server DFSes, such as Ceph, because Assise demonstrated an order of magnitude faster performance than those DFSes. We configure Assise with three settings: `Assise` is vanilla Assise in "pessimistic" [18] mode, guaranteeing persistence on `fsync()`, just like LineFS. `Assise-BgRepl` additionally replicates in the background, before `fsync()` is called, akin to LineFS replication. `Assise-BgRepl` uses 3 threads for background replication (which we confirmed maximizes performance) and the same 4MB chunk size, but does not implement pipeline parallelism. `Assise+hyperloop` adapts the replication protocol from Hyperloop [36], a system that offloads replication to an RDMA NIC. The source code of Hyperloop is unavailable, so we implemented the proposed design. Hyperloop manipulates RDMA request destination addresses on the remote NIC via RDMA verbs. Modern RDMA NICs (ConnectX-4 and later—our BlueField uses a ConnectX-5) disable this feature for security. To get around this problem, we run `Assise+Hyperloop` with an IO trace generated in advance, which specifies the modified destination addresses when replicating data, avoiding remote request manipulation. Our implementation is validated by the Hyperloop authors and we reproduced the results presented in the Hyperloop paper. Because our Hyperloop evaluation requires traces, we compare LineFS with `Assise+Hyperloop` for microbenchmarks only. To evaluate the effectiveness of LineFS's pipeline parallelism, we configure LineFS without it (`LineFS-NotParallel`), so it replicates the client-private log *sequentially* in the background. We configure the PM log

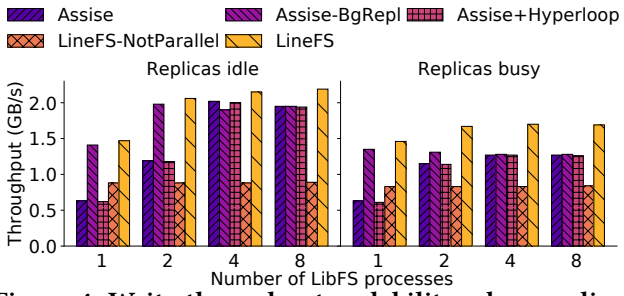


Figure 4: Write throughput scalability when replicas are idle and busy.

size to 512 MB for both Assise and LineFS. Both DFSes use 3 nodes; primary, replica-1, and replica-2.

Multi-tenancy. It is common practice to run IO-intensive and CPU-intensive jobs together in the cloud [29, 34, 43]. To evaluate the resulting interference with DFS execution, we use `streamcluster` from the PARSEC [20] suite v3.0 to mimic CPU-intensive jobs. To stress the host cores, we set the number of threads for `streamcluster` equal to the number of host cores. When running `streamcluster` with Assise or LineFS, we adjust the scheduling priority to evaluate the impact of resource contention.

Test suite. We test LineFS with xfstests [17] and CrashMonkey [47]. LineFS successfully passes all 75 general xfstest cases and all CrashMonkey tests. Also, LineFS passes all 201 LevelDB unit tests and is able to run all Filebench profiles.

5.2 Microbenchmarks

5.2.1 Write Throughput Scalability. Our throughput benchmark writes file data to a 12 GB file with 16 KB IO size sequentially and calls `fsync` at the end. Figure 4 shows the scalability of throughput by increasing the number of DFS clients. Each client has its own file of the same size. We measure throughput when the replicas' CPU is idle (no co-runner) and busy.

Replicas idle. `Assise` shows the worst throughput with a single client (645 MB/s). `Assise` synchronously replicates updates only upon `fsync()` and within the thread context of the caller. `Assise` throughput is thus heavily dependent on the number of client contexts. `Assise-BgRepl` improves the performance by up to 124% due to replicating in the background. `LineFS` performs up to 133% and 4% better than `Assise` and `Assise-BgRepl`, respectively. `LineFS`'s pipeline parallelism hides PM persistence and replication latencies, achieving similar throughput to `Assise-BgRepl`, even with a more complex data-path. `LineFS-NotParallel` performs at least 60% worse than `LineFS`, demonstrating the importance of pipeline parallelism for good offload performance.

Sequential read		Random read	
`Assise`	`LineFS`	`Assise`	`LineFS`
3,147	3,134	2,960	2,946

Table 2: Read throughput (MB/s) of `Assise` and `LineFS`

When increasing the number of clients, both `LineFS` and `Assise` become increasingly resource-bottlenecked. The maximum goodput of our microbenchmark is 2.2 GB/s. `LineFS` saturates the maximum goodput with only two clients due to background parallel replication, but `Assise` needs four clients. We stop at eight clients because both systems already saturate network bandwidth.

Replicas busy. In this benchmark, we are interested in how DFS performance is affected by host resource contention due to background co-tenant activity. Hence, when co-running with applications, we give the DFS higher scheduling priority to eliminate scheduling policy as a source of performance degradation. We run `streamcluster` as co-runner in the replicas, leaving the primary for the microbenchmark. With a single client, all DFSes achieve similar performance to the idle case because they have a higher priority than `streamcluster` and the contention is not high. As we increase the number of clients, all DFSes experience increased host contention and none can saturate the network bandwidth. LineFS experiences the least contention due to offload and outperforms the other DFSes by 33% at scale. Note that `Assise+Hyperloop` still requires periodic host participation for publication and posting of RDMA verbs for replication offload, reducing its performance. Even LineFS still requires a kernel worker for publication, causing performance degradation versus the idle case. We analyze the kernel worker cost in §5.2.4, showing that skipping kernel worker publication in replicas allows `LineFS` to saturate the network bandwidth, and confirming that *resource contention* in replicas is a bottleneck, *even when DFSes are prioritized.*

In all cases, `LineFS-NotParallel` does not scale because it does not effectively utilize SmartNIC resources. `LineFS-NotParallel` uses 4.1 ARM cores on average, whereas `LineFS` uses 5.8 cores. This result confirms that *pipeline parallelism is essential when offloading a DFS to SmartNICs.*

5.2.2 Read Throughput. Table 2 presents the read throughput of `LineFS` and `Assise`. The microbenchmark runs a single client reading a 12 GB file with 16 KB IO size locally. For reads, `LineFS` does not offload any operations to SmartNICs. The entire read data-path is performed in the host CPU. Therefore, `LineFS` and `Assise` show similar performance in both sequential and random read cases.

5.2.3 Pipeline Performance Analysis. We break down the time taken by each pipeline stage when publishing and replicating a 4 MB chunk. Figure 5 shows this result. Publish and

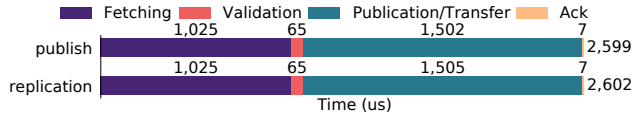


Figure 5: Publish and replication pipeline latency breakdown.

Figure 6: Performance impact of `LineFS` and `Assise` co-execution on `streamcluster` execution time (left Y-axis) and DFS throughput (right Y-axis).

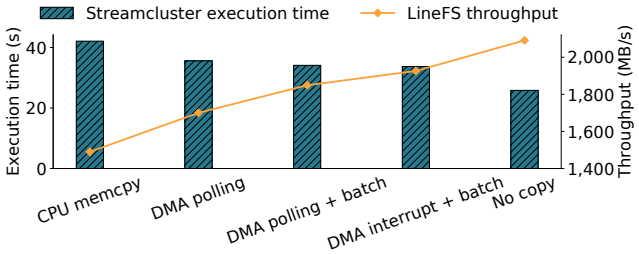


Figure 7: Impact of different copying methods on `streamcluster` execution time (left Y-axis) and LineFS throughput (right Y-axis).

	Replicas idle			Replicas busy		
	Avg.	99th	99.9th	Avg.	99th	99.9th
`Assise`	76	101	126	323	7,115	8,331
`Assise+hyperloop`	60	68	4,716	61	78	4,125
`LineFS`	149	187	205	149	188	204

Table 3: Latency (μs) when idle and busy.

replication pipelines share the fetching and validation stages, so their latencies are identical. As expected, fetching and publication/transfer make up the largest portion of processing latency due to having to cross high-latency interconnects. As pipeline stage execution overlaps, this benchmark shows the per-stage latency that is hidden by pipeline parallelism.

5.2.4 Performance Interference. To compare the performance interference of `LineFS` and `Assise`, we run `streamcluster` in all nodes, including the primary. This configuration represents aggressive consolidation by running an IO and CPU-intensive workload together. We run the throughput microbenchmark (two DFS clients, each running a single thread) and `streamcluster` with 48 threads (using all cores on the node) under the same priority. Figure 6 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`, while incurring minimal slowdown of `streamcluster` (49% and 19% slowdown compared to the solo run for primary and replica, respectively). This result confirms that *LineFS's design minimizes host performance interference while providing good performance.*

Kernel worker interference. As we have seen in §5.2.1, kernel worker publication is a major source of interference with co-running applications. To understand the contention, we implement several publication methods and evaluate the performance of `streamcluster` (co-running host application) and LineFS according to these methods. Figure 7 shows the execution time of `streamcluster` while four LineFS clients run the throughput microbenchmark with the same scheduling priority as the `streamcluster` process. With `DMA polling`, the kernel worker issues DMA requests to copy memory and waits for completion in a busy loop, which is a standard implementation used in Intel SPDK [15]. `DMA polling + batch` is an optimization of `DMA polling`, where NICFS batches memory copy requests to the kernel worker. `DMA interrupt + batch` means the kernel worker blocks until a DMA completion interrupt arrives, causing less contention than polling-based methods. `No copy` skips publication altogether.

As expected, `streamcluster` performs worse as LineFS uses heavier-weight host publication methods. `streamcluster` performance of `No copy` is identical to running `streamcluster` alone. When the kernel worker uses `DMA interrupt + batch`, `streamcluster` performance decreases by 23% compared to `No copy`, whereas `CPU memcpy` makes the performance drop by 61.5%. LineFS throughput shows how LineFS performs under contention with `streamcluster`. Under contention, `DMA interrupt + batch` increases LineFS throughput by 40.3% and 5.9% compared to `CPU memcpy` and `DMA polling + batch`, respectively. Hence, we use `DMA interrupt + batch` in all other benchmarks (including §5.2.1).

5.2.5 Latency. We measure the latency of a microbenchmark that writes 16 KB of data followed by `fsync()` in a 12 GB file, so replication occurs at each write. This scenario is not ideal for LineFS because, unlike throughput, LineFS cannot perform replication in the background. Table 3 reports

the statistics of our latency measurements. When replicas are idle, LineFS's latency is higher than that of Assise due to the following reasons. 1) The hardware data-path of LineFS has multiple high-latency steps (cf. §5.2.3). LineFS fetches IO data from host PM to SmartNIC memory via RDMA. After running the pipeline stages, LineFS makes another RDMA request to ConnectX transport hardware through the SoC-internal PCIe interconnect. 2) Due to low SmartNIC CPU clock speed, we measured that the SmartNIC's L3 cache and DRAM access latency are more than 2× slower than on our host hardware. In contrast, Assise can make direct RDMA requests from fast host CPUs.

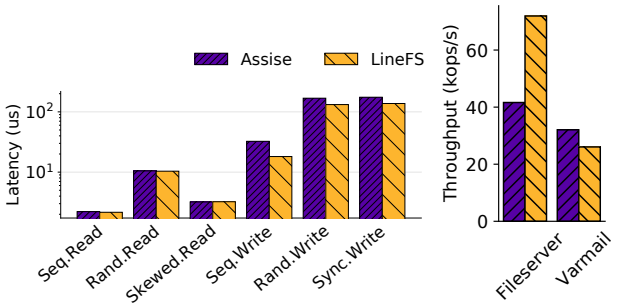
However, when replicas are busy, LineFS's latency is similar to when replicas are idle. In this case, LineFS shows up to 41× better latency than Assise. This result confirms our claim that *LineFS minimizes interference of DFS operations with host applications*. Assise+Hyperloop also shows good average and 99^{th} percentile latency for both cases. However, Hyperloop requires periodic posting of RDMA verbs by the host CPU. If posting is delayed (e.g., due to contention), Hyperloop increases latency. Due to this effect, the 99.9^{th} percentile latency of Hyperloop is 23× worse than LineFS.

5.3 Application benchmarks

LevelDB. We run a LevelDB performance benchmark using db_bench, distributed with LevelDB's source code. The benchmark includes sequential and random object insert and read, skewed (1% of frequently accessed objects) read, and synchronous insert (an insert followed by fsync()). We set a key size of 16 B and a value size of 1 KB (default testing configuration of LevelDB). We run the LevelDB performance benchmark when replicas are busy.

Figure 8a shows the average LevelDB latency (μs) results (y-axis is in log scale). As expected, LineFS and Assise perform similarly for read operations. LineFS has 80% better latency in sequential insert and 27% better latency in random insert than Assise. The insert performance shows similar trends to the write microbenchmark. In the case of synchronous insert, LineFS obtains 27% better latency (lower improvement than sequential insert) because of the workload's frequent fsync() calls.

Filebench. We run the Fileserver and Varmail workloads in the Filebench suite [54]. Fileserver executes file operations on files of 128 KB average size. Varmail operates on smaller files of 16 KB average size (emulating reading small mailbox files). We set a working set of 10K files for both benchmarks. The write to read ratio of Fileserver and Varmail is 2:1 and 1:1, respectively. Finally, Varmail frequently calls fsync() for its persistence semantics (emulating write-ahead logging when updating mailbox files), whereas Fileserver does not call fsync() (relaxed crash consistency).



(a) LevelDB (lower is better, log scale) **(b) Filebench (higher is better)**

Figure 8: LevelDB latency and Filebench throughput.

Figure 8b shows the average throughput of Filebench when replicas are busy. In Fileserver, LineFS gets 79% better throughput than Assise because the Fileserver workload performs writes frequently. LineFS's write performance is better than Assise's as shown in the microbenchmark results (§5.2). Also, Fileserver does not call fsync(), so LineFS is able to run all pipeline stages in the background. The Varmail result shows different trends. Mail server IO characteristics do not favor LineFS. The Varmail workload performs small file creation and updates followed by fsync(), so LineFS cannot efficiently leverage pipeline parallelism. In addition, each time Varmail opens a file, LibFS requests a permission check to NICFS via PCIe, whereas Assise performs the permission check on the host. We identify that open() takes up 9.1% of all file system calls during the Varmail experiment, providing a second major reason that LineFS is 21% slower than Assise.

5.4 Data-path Processing

We investigate the opportunity for additional data-path processing using spare SmartNIC resources and its impact on DFS performance. In particular, we evaluate how effectively LineFS's data compression saves network bandwidth and its impact on replication performance. Our use case is batch processing. Distributed batch processing applications often write intermediate data to a DFS (cf. Hadoop [52] on HDFS). The intermediate data is replicated, causing high network bandwidth utilization. LineFS compresses the data in the primary to save network bandwidth when replicating data.

To evaluate this scenario, we run a parallel batch processing benchmark, Tencent Sort [35]. Tencent Sort performs parallel sorting, consisting of a range partitioning phase and a merge-sort phase. In the partitioning phase, each worker process partitions the input data into non-overlapping ranges, and stores the partitioned data to temporary files. In the merge-sort phase, each sort worker process reads the temporary files assigned to the process, merges data from the files,

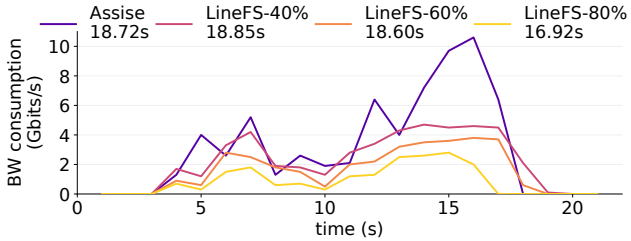


Figure 9: Network bandwidth consumption of `Assise` and `LineFS` while running Tencent Sort. `LineFS-x%` indicates LineFS with an input data set of x% compression ratio. The numbers in the legend indicate execution time of Tencent Sort.

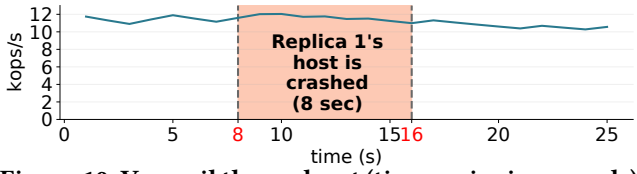


Figure 10: Varmail throughput (time series in seconds). Replica-1's host (VM) fails at time 8s and recovers at time 16s.

and sorts them. Each sort worker writes its final output to a file.

We implement Tencent Sort for evaluating `Assise` and `LineFS`. We configure four partitioning and four sorting processes in the primary node. The data set consists of 80M records. We use the Quicksort algorithm for the merge and sort phase and radix-based range partitioning.

We evaluate network bandwidth consumption and benchmark performance according to different compression ratios. To control compression ratios, we modify the benchmark input generation tool [14] to adjust the ratio of zero values in input files. We create three input sets to have 40%, 60% and 80% compression ratios respectively. For compression, NICFS runs the Lempel-Ziv-Welch algorithm [7]. In our evaluation, the compression throughput of a single SmartNIC core is around 200 MB/s, which is much lower than the network bandwidth. NICFS performs parallel compression to avoid a pipeline bottleneck in the compression stage, running 16 threads to execute the compression stage in our evaluation. To stress network bandwidth, we also run `iperf3` [16] as a background task that contends for network bandwidth.

Figure 9 shows the network bandwidth consumption of `Assise` and `LineFS` with different input sets. As expected, LineFS reduces network bandwidth by 29%, 49%, and 72% compared to `Assise` in `LineFS-40%`, `LineFS-60%`, and `LineFS-80%`, respectively. We measure the time taken by the merge-sort phases. When the compression ratio is low, `LineFS` performance is comparable to `Assise`. When the compression ratio is 80%, `LineFS` (16.69 seconds) performs 10.6% better than `Assise` (18.72 seconds), while saving 72% of network bandwidth.

5.5 Availability

We evaluate LineFS's extended availability in the face of replica failures with a failure experiment using 3 nodes (one primary, two replicas). We run Varmail in the Filebench suite on the primary node and instrument Varmail to report its throughput every second.[1] While running Varmail, we crash replica-1 by injecting a failure that reboots the host (running in a VM).

Figure 10 shows the Varmail throughput over time. At 8 seconds, NICFS in replica-1 detects the failure because the kernel worker is not responsive to heartbeat messages. NICFS changes its copy mode to use PCIe for publishing replica-1's log. Replica-1's NICFS also continues replicating data to replica-2 while replica-1's host system is down. Varmail replicates data at a few hundred MB/s and we do not observe a performance drop due to PCIe memory copy during the host failure window. At time 16s, the host system recovers. The host kernel worker is stateless and can restart copying without any further recovery. At time 16s, NICFS detects that the kernel worker has recovered and starts sending RPC requests to resume host-based publication.

6 RELATED WORK

Distributed file systems. Disk-based client-server file systems used in large scale clouds and high-performance computing systems, such as Ceph [58], LustreFS [6], GPFS [4], GFS [30], and HDFS [52], focus on providing scalability and high availability for thousands of storage nodes. These file systems are inherently slow with fast PM storage due to their complex design and client-server architecture, motivating PM-optimized DFSes leveraging fast RDMA.

Various PM-optimized DFSes follow the client-server model, but optimize communication primitives and metadata placement. Octopus [44] couples PM with high-performance RDMA, introducing collect-dispatch transactions to provide lower-latency access to data and metadata. Clover [56] manages remote storage via RDMA operations and locates metadata with applications. While these DFSes optimize communication overheads, the cost of accessing remote storage via the network remains high.

Assise [18] and Orion [60] leverage the client-local model. Orion places data in local PM for "internal clients" to reduce network accesses, but communicates metadata updates with remote metadata servers. Assise places data and metadata in the client node, eliminating network latency to access

[1]Note that printing Varmail's throughput every second reduces Varmail throughput compared to §5.3.

remote storage and metadata servers. Assise uses RDMA to replicate data and metadata. These DFSes provide superior performance for PM than the client-server model. However, the client-local model has to share host resources between DFS management operations and co-running applications.

SmartNIC offload. Many approaches offload host tasks to SmartNICs. Floem [50] and ClickNP [39] offload network functions like rate limiter and firewall to SmartNICs. AccelNet [27] offloads processing TCP and SDN stack. For FPGA-based SmartNICs, Tonic [19] enables programmable transport protocols and hXDP [22] efficiently executes eBPF programs. iPipe [41] proposes the actor programming model for offloading applications like KV store, distributed transaction systems, and real-time analytics to SoC-based SmartNICs. E3 [42] and λ-NIC [25] offload microservices to SoC-based SmartNICs. FairNIC [31] and PANIC [40] address fairness and performance isolation problems among competing offload tasks in SmartNICs. These approaches share the same vision with LineFS of reducing host CPU burden by offloading tasks and leveraging resources in SmartNICs to accelerate the offloaded tasks, but they do not address unique challenges of DFS offload, such as handling highly structured data with sophisticated access protocols for consistency and durability across multiple SmartNICs.

Hyperloop [36] offloads chain replication to RDMA NICs. Hyperloop uses the RDMA WAIT verb and remote manipulation of RDMA requests for replication without involving the host CPU. Hyperloop provides stable tail latency even when a replica's host CPU is busy. Hyperloop focuses on replicated transactions, but offloading a full-fledged DFS has to address more issues: shared file coordination, metadata consistency, crash consistency, and scalability by efficiently utilizing SmartNICs. LineFS proposes general design ideas when offloading DFS operations to SmartNICs.

7 CONCLUSION

LineFS proposes the persist-and-publish model and pipeline parallelism to offload a PM-optimized DFS to SmartNICs. LineFS offloads CPU-intensive DFS tasks, like replication, compression, data publication, index and consistency management to a SmartNIC. We implement LineFS on the Mellanox BlueField SmartNIC and compare to Assise, a state-of-the-art PM DFS. LineFS improves latency in LevelDB up to 80% and throughput in Filebench up to 79%, while providing extended DFS availability during host system failures.

LineFS is available at https://github.com/casys-kaist/LineFS.

Acknowledgments. We thank the anonymous reviewers and our shepherd, Manya Ghobadi, for their comments and feedback. This work is supported by an Institute of Information & communications Technology Planning & Evaluation (IITP) grant, funded by the Korean government (MSIT) (No. 2018-0-00503, Research on next generation memory-centric computing system architecture). This work is also supported by a fellowship from the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC), funded by the European Commission (EACEA) (FPA 2012-0030), and, in part, by ERC grant 770889, NSF grant CNS-1900457, Huawei, and the Texas Systems Research Consortium.

REFERENCES

[1] 2021. *Apache ZooKeeper.* https://zookeeper.apache.org
[2] 2021. *BlueField SmartNIC Ethernet.* https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet
[3] 2021. *Broadcom Stingray.* https://www.broadcom.com/blog/at-a-glance--the-broadcom-stingray-ps1100r-delivers-breakthrough-performance-and-efficiency-for-nvme-of-storage-target-applications
[4] 2021. *General Parallel File System.* https://www.ibm.com/docs/en/gpfs
[5] 2021. *Intel Optane Memory.* http://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html
[6] 2021. *Lustre File System.* http://www.lustre.org
[7] 2021. *LZ4 compression algorithm.* https://en.wikipedia.org/wiki/LZ4_(compression_algorithm)
[8] 2021. *Marvell's Data Processing Units.* https://www.marvell.com/products/data-processing-units.html
[9] 2021. *Network-attached Storage.* https://en.wikipedia.org/wiki/Network-attached_storage
[10] 2021. *NVM Express over Fabrics 1.1.* https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1-2019.10.22-Ratified.pdf
[11] 2021. *pmem.io — Persistent Memory Programming.* https://pmem.io
[12] 2021. *RDMA Consortium.* http://www.rdmaconsortium.org
[13] 2021. *RDMA over Converged Ethernet (RoCE).* https://www.roceinitiative.org
[14] 2021. *Sort Benchmark Home Page.* http://sortbenchmark.org
[15] 2021. *Storage Performance Development Kit.* http://www.spdk.io
[16] 2021. *Tool for active measurements of the maximum achievable bandwidth on IP networks.* http://software.es.net/iperf
[17] 2021. *Xfstests.* https://git.kernel.org/pub/scm/fs/xfs/xfstests-dev.git
[18] Thomas E. Anderson, Marco Canini, Jongyul Kim, Dejan Kostić, Youngjin Kwon, Simon Peter, Waleed Reda, Henry N. Schuh, and Emmett Witchel. 2020. Assise: Performance and Availability via Client-local NVM in a Distributed File System. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20).* USENIX Association, 1011–1027. https://www.usenix.org/conference/osdi20/presentation/anderson
[19] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. 2020. Enabling Programmable Transport Protocols in High-Speed NICs. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20).* USENIX Association, Santa Clara, CA, 93–109. https://www.usenix.org/conference/nsdi20/presentation/arashloo
[20] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques* (Toronto, Ontario, Canada) *(PACT 08).* Association for Computing Machinery, New York, NY, USA, 72–81. https://doi.org/10.1145/1454115.1454128

[21] Robert Birke, Ioana Giurgiu, Lydia Y. Chen, Dorothea Wiesmann, and Ton Engbersen. 2014. Failure Analysis of Virtual and Physical Machines: Patterns, Causes and Characteristics. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 14)*. IEEE Computer Society, Washington, DC, USA, 1–12. https://doi.org/10.1109/DSN.2014.18

[22] Marco Spaziani Brunella, Giacomo Belocchi, Marco Bonola, Salvatore Pontarelli, Giuseppe Siracusano, Giuseppe Bianchi, Aniello Cammarano, Alessandro Palumbo, Luca Petrucci, and Roberto Bifulco. 2020. hXDP: Efficient Software Packet Processing on FPGA NICs. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 973–990. https://www.usenix.org/conference/osdi20/presentation/brunella

[23] Youmin Chen, Youyou Lu, and Jiwu Shu. 2019. Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing. In *Proceedings of the Fourteenth EuroSys Conference 2019* (Dresden, Germany) *(EuroSys 19)*. Association for Computing Machinery, New York, NY, USA, Article 19, 14 pages. https://doi.org/10.1145/3302424.3303968

[24] Vijay Chidambaram, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2013. Optimistic Crash Consistency. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP 13)*. Association for Computing Machinery, New York, NY, USA, 228–243. https://doi.org/10.1145/2517349.2522726

[25] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. 2020. lambda-NIC: Interactive Serverless Compute on Programmable SmartNICs. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS 20)*. 67–77. https://doi.org/10.1109/ICDCS47774.2020.00029

[26] James Cipar, Greg Ganger, Kimberly Keeton, Charles B. Morrey, Craig A.N. Soules, and Alistair Veitch. 2012. LazyBase: Trading Freshness for Performance in a Scalable Database. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys 12)*. Association for Computing Machinery, New York, NY, USA, 169–182. https://doi.org/10.1145/2168836.2168854

[27] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

[28] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. 2010. Availability in Globally Distributed Storage Systems. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. USENIX Association, Vancouver, BC. https://www.usenix.org/conference/osdi10/availability-globally-distributed-storage-systems

[29] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating Interference at Microsecond Timescales. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 281–297. https://www.usenix.org/conference/osdi20/presentation/fried

[30] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA) *(SOSP 03)*. ACM, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450

[31] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C. Snoeren. 2020. SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM 20)*. Association for Computing Machinery, New York, NY, USA, 681–693. https://doi.org/10.1145/3387514.3405895

[32] C. Gray and D. Cheriton. 1989. Leases: An Efficient Fault-tolerant Mechanism for Distributed File Cache Consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles (SOSP 89)*. ACM, New York, NY, USA, 202–210. https://doi.org/10.1145/74850.74870

[33] John L. Hennessy and David A. Patterson. 2017. *Computer Architecture, Sixth Edition: A Quantitative Approach* (6th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[34] Calin Iorgulescu, Reza Azimi, Youngjin Kwon, Sameh Elnikety, Manoj Syamala, Vivek Narasayya, Herodotos Herodotou, Paulo Tomita, Alex Chen, Jack Zhang, and Junhua Wang. 2018. PerfIso: Performance Isolation for Commercial Latency-Sensitive Services. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 519–532. https://www.usenix.org/conference/atc18/presentation/iorgulescu

[35] Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao, Mark R Nutter, and Jeremy D Schaub. 2016. *Tencent Sort.* Technical Report. Tencent Corporation. http://sortbenchmark.org/TencentSort2016.pdf.

[36] Daehyeok Kim, Amirsaman Memaripour, Anirudh Badam, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Shachar Raindel, Steven Swanson, Vyas Sekar, and Srinivasan Seshan. 2018. Hyperloop: Group-Based NIC-Offloading to Accelerate Replicated Transactions in Multi-Tenant Storage Systems. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM 18)*. Association for Computing Machinery, New York, NY, USA, 297–312. https://doi.org/10.1145/3230543.3230572

[37] Youngjin Kwon, Henrique Fingler, Tyler Hunt, Simon Peter, Emmett Witchel, and Thomas Anderson. 2017. Strata: A Cross Media File System. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP 17)*. Association for Computing Machinery, New York, NY, USA, 460–477. https://doi.org/10.1145/3132747.3132770

[38] Thai Le, Jonathan Stern, and Stephen Briscoe. 2017. *Fast memcpy with SPDK and Intel I/OAT DMA Engine.* https://software.intel.com/en-us/articles/fast-memcpy-using-spdk-and-ioat-dma-engine

[39] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM 16)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/2934872.2934897

[40] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. 2020. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 243–259. https://www.usenix.org/conference/osdi20/presentation/lin

[41] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading Distributed Applications onto SmartNICs Using IPipe. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM 19)*. Association for Computing Machinery, New York, NY, USA, 318–333. https://doi.org/10.1145/3341302.3342079

[42] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 363–378. https://www.usenix.org/conference/atc19/presentation/liu-ming

[43] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA 15)*. Association for Computing Machinery, New York, NY, USA, 450–462. https://doi.org/10.1145/2749469.2749475

[44] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an RDMA-enabled Distributed Persistent Memory File System. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 773–785. https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu

[45] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. 2007. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux Symposium*, Vol. 2. Ottawa, ON, Canada.

[46] Hongyu Miao, Heejin Park, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S. McKinley, and Felix Xiaozhu Lin. 2017. StreamBox: Modern Stream Processing on a Multicore Machine. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 617–629. https://www.usenix.org/conference/atc17/technical-sessions/presentation/miao

[47] Jayashree Mohan, Ashlie Martinez, Soujanya Ponnapalli, Pandian Raju, and Vijay Chidambaram. 2018. Finding Crash-Consistency Bugs with Bounded Black-Box Crash Testing. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 33–50. https://www.usenix.org/conference/osdi18/presentation/mohan

[48] Ian Neal, Gefei Zuo, Eric Shiple, Tanvir Ahmed Khan, Youngjin Kwon, Simon Peter, and Baris Kasikci. 2021. Rethinking File Mapping for Persistent Memory. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, 97–111. https://www.usenix.org/conference/fast21/presentation/neal

[49] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 361–378. https://www.usenix.org/conference/nsdi19/presentation/ousterhout

[50] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. Floem: A Programming System for NIC-Accelerated Network Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 663–679. https://www.usenix.org/conference/osdi18/presentation/phothilimthana

[51] Robbert Van Renesse and Fred B. Schneider. 2004. Chain Replication for Supporting High Throughput and Availability. In *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*. USENIX Association, San Francisco, CA. https://www.usenix.org/conference/osdi-04/chain-replication-supporting-high-throughput-and-availability

[52] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST 10)*. 1–10. https://doi.org/10.1109/MSST.2010.5496972

[53] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: Smart Remote Memory. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys 20)*. Association for Computing Machinery, New York, NY, USA, Article 29, 16 pages. https://doi.org/10.1145/3342195.3387519

[54] Vasily Tarasov, Erez Zadok, and Spencer Shepler. 2016. Filebench: A Flexible Framework for File System Benchmarking. *USENIX ;login:* 41, 1 (2016).

[55] Shelby Thomas, Rob McGuinness, Geoffrey M. Voelker, and George Porter. 2018. Dark Packets and the End of Network Scaling. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems* (Ithaca, New York) *(ANCS 18)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3230718.3230727

[56] Shin-Yeh Tsai, Yizhou Shan, and Yiying Zhang. 2020. Disaggregating Persistent Memory and Controlling Them Remotely: An Exploration of Passive Disaggregated Key-Value Stores. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 33–48. https://www.usenix.org/conference/atc20/presentation/tsai

[57] Yang Wang, Manos Kapritsos, Zuocheng Ren, Prince Mahajan, Jeevitha Kirubanandam, Lorenzo Alvisi, and Mike Dahlin. 2013. Robustness in the Salus Scalable Block Store. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, Lombard, IL, 357–370. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/wang_yang

[58] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: A Scalable, High-Performance Distributed File System. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06)*. USENIX Association, Seattle, WA. https://www.usenix.org/conference/osdi-06/ceph-scalable-high-performance-distributed-file-system

[59] Jian Xu and Steven Swanson. 2016. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. USENIX Association, Santa Clara, CA, 323–338. https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu

[60] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2019. Orion: A Distributed File System for Non-Volatile Main Memory and RDMA-Capable Networks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. USENIX Association, Boston, MA, 221–234. https://www.usenix.org/conference/fast19/presentation/yang

[61] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2020. FileMR: Rethinking RDMA Networking for Scalable Persistent Memory. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 111–125. https://www.usenix.org/conference/nsdi20/presentation/yang