SwitchFS: Exploiting Persistent Memory Bandwidth in Cluster via Learning-based I/O Path Selection

ZHENLIN QI, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
SHENGAN ZHENG, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
BOWEN ZHANG, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
LINPENG HUANG, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

Persistent Memory (PM) and Remote Direct Memory Access (RDMA) technologies have significantly improved the storage and network performance in data centers and spawned a slew of distributed file systems (DFS) designs. Existing DFSs often consider remote storage a performance constraint, assuming it delivers lower bandwidth and higher latency than local devices. However, the advancements in RDMA technology present an opportunity to narrow the performance disparity between local and remote access, empowering DFSs to harness both local and remote I/O capabilities, thereby attaining higher aggregated throughput.

We propose SwitchFS, a new DFS architecture that exploits all the available PM bandwidth in a cluster with generative I/O path assignment policy. It dynamically steers client-side I/O requests to the local cache and the remote devices accessible through RDMA network. To determine the run-time I/O path adaptively, we introduce MELON, an I/O path selection policy based on a reinforcement learning model, designed to optimize expected I/O latency. Meanwhile, SwitchFS adopts a model-driven opportunistic replication mechanism at the server-side, which further improves I/O bandwidth utilization of remote devices across the cluster.

Extension of Conference Paper. This article is an extended version of confernce paper: Qi, Zhenlin, et al. "Conflux: Exploiting Persistent Memory and RDMA Bandwidth via Adaptive I/O Mode Selection." Proceedings of the 52nd International Conference on Parallel Processing. 2023. The present version contains approximately 35 learning-based I/O latency optimizer that dynamically adapts to system-level I/O characteristics, which provides more explorative path selection policy than the original supervised learning methods. (2) Model-driven Replication Strategy: We propose a novel RL-informed replication mechanism that leverages runtime model feedback to opportunistically place replicas, improving both fault tolerance and performance compared to the original version. (3) Comprehensive Evaluation: We evaluate the enhanced SwitchFS under a range of workloads and compare it with both the original Conflux and other representative distributed file systems. Our results demonstrate significant improvements in latency, throughput, and scalability.
This work is supported by National Key Research and Development Program of China (Grant No. 2024YFB4505203), National Natural Science Foundation of China (NSFC) (Grant No. 62227809, 62332012, 62302290), the Fundamental Research Funds for the Central Universities, and Key Research and Development Program of Xinjiang Uygur Autonomous Region (Grant No. 2023B01027, 2023B01027-1).
Authors' Contact Information: Zhenlin Qi, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, e-mail: qizhenlin@sjtu.edu.cn; Shengan Zheng (corresponding author), Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, e-mail: shengan@sjtu.edu.cn; Bowen Zhang, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, Shanghai, China, e-mail: bowenzhang@sjtu.edu.cn; Linpeng Huang (corresponding author), Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, e-mail: lphuang@sjtu.edu.cn.

ACM 1544-3973/2026/06-ART67
https://doi.org/10.1145/3818684

Furthermore, we adopt the fine-grained concurrency control approach to improve scalability. Across synthetic and real-world application workloads, SwitchFS improves throughput by up to 5.9× over existing DFS designs while preserving low tail latency and incurring modest learning overhead.

CCS Concepts: • **Information systems** → *Phase change memory*; **Distributed storage**; • **Social and professional topics** → **File systems management**; • **Computing methodologies** → *Neural networks;*

Additional Key Words and Phrases: Persistent memory, RDMA, distributed file systems, reinforcement learning, storage systems

ACM Reference Format:
Zhenlin Qi, Shengan Zheng, Bowen Zhang, and Linpeng Huang. 2026. SwitchFS: Exploiting Persistent Memory Bandwidth in Cluster via Learning-based I/O Path Selection. *ACM Trans. Arch. Code Optim.* 23, 2, Article 67 (June 2026), 26 pages. https://doi.org/10.1145/3818684

1 Introduction

The evolvement of byte-addressable **persistent memory** (**PM**) has offered orders of magnitude higher performance than conventional block devices in the storage stack. Meanwhile, current data centers are actively adopting **remote direct memory access** (**RDMA**) technology, which provides lower latency and higher bandwidth than traditional network protocols. The combination of these two hardware technologies has dramatically improved the performance of remote persistent data access. As a result, massive research works have been proposed to re-organize and optimize the software stack of **distributed file systems** (**DFS**). Specifically, several DFS designs have benefited from the high performance PM and RDMA devices via unbuffering and direct data access. However, existing DFS designs fall short in fully exploiting all the aggregated bandwidth of local and remote storage devices due to their simply inheritance of the conventional deterministic I/O paradigm.

In traditional DFSs, I/O performance is hampered by the limited network bandwidth and the poor performance of disk devices [19, 54]. Given the advancement in hardware technologies, there are two types of optimization in the existing literature: The first is to utilize the new network hardware, redesigning the software to exploit the remote storage performance. Systems designed according to this paradigm always reduce the communication latency and increase the remote data access throughput [24, 37, 38]. The second is to utilize the new storage technology, redesigning the local storage system software. Systems designed according to this paradigm follow the conventional wisdom: *local access is better than remote*, and build DFSs with client local cache architecture to exploit local access performance [2, 30, 60].

Nevertheless, the conventional pearls of wisdom are not entirely applicable to current hardware. Specifically, the ***local access is better than remote*** principle is ***no longer valid*** for DFS built on PM and RDMA devices. In the traditional distributed storage architecture, the bandwidth of accessing the client-local cache is orders of magnitude higher than that of accessing remote disks, resulting in an imbalance between local and remote I/O performance. However, both PM and RDMA devices provide extremely low latency and tens of GB per second of bandwidth. The combination of PM and RDMA restores the balance between local and remote storage I/O performance. Rigid I/O path decided at the compile-time is no longer suitable for the new hardware, leaving part of the available devices under-utilized in a cluster. The key to exploiting all available hardware performance is to simultaneously utilize both local and remote devices, taking advantage of the aggregated bandwidth of local PM and the RDMA networks. Nevertheless, such a system comes with three main challenges that a DFS design must properly address: First, it is challenging for a file system to dynamically steer I/O tasks to local and remote devices, instead of the conventional function with deterministic I/O paths. Second, it is challenging to decide and assign low-latency I/O paths at run-time. Different applications tend to have different file access patterns, and improper

I/O path scheduling in the cluster will deteriorate its overall performance. Third, it is challenging to fully take advantage of the high bandwidth hardware with conventional concurrency control approaches, which generally enforce coarse-grained read-write locks.

We propose SwitchFS, a new DFS architecture that ***fully exploits all available I/O bandwidth within a storage cluster.*** It tackles the above-mentioned challenges and provides aggregated bandwidth to I/O-intensive applications. SwitchFS adopts a dynamic I/O management mechanism, which can serve user I/O requests with different internal data transmission paths at the client side. As a result, local and remote PM devices are transparently exposed to the user space, providing available I/O bandwidth simultaneously. To generate adaptive I/O path selection decisions for various applications, we propose MELON, a learned I/O selection policy that utilizes the reinforcement learning paradigm to estimate the benefits of taking different I/O path choices. MELON adopts a light-weight neural network model to learn from the historical latency changes and related working environments, aiming at optimizing the I/O latency of every application request. Meanwhile, we design a model-driven replication mechanism which opportunistically allocates more replicas to the node with lower predicted I/O pressure, resulting to an increased device bandwidth utilization at the server side. To eliminate the concurrency control bottleneck and take advantage of SwitchFS in scalable working set, we design a fine-grained concurrency control mechanism facilitating highly concurrent I/O requests. To the best of our knowledge, SwitchFS is the first DFS that employs an online reinforcement learning agent to aggregate the bandwidth of local and remote PM devices via adaptive I/O path selection.

In summary, we make the following contributions:

- We present the design of SwitchFS, a DFS architecture that exploits both local and remote device bandwidth in a cluster by activating different I/O paths dynamically at run-time.
- We present MELON, an online reinforcement learning-based I/O path selection policy in SwitchFS. MELON adopts the reinforcement learning paradigm, extracting knowledge from the historical I/O logs and system state to help SwitchFS make adaptive run-time decisions among the available I/O paths and aggregate local and remote PM bandwidth without relying on static thresholds or offline training.
- We design a model-driven opportunistic replication mechanism that further improves the utilization of device bandwidth in a cluster, and refine a lightweight fine-grained concurrency control mechanism that provides higher scalability with clear consistency guarantees between local and remote devices.
- We implement SwitchFS and conduct a comprehensive evaluation on synthetic benchmarks and real-world applications. Our experiments quantify the computational overhead of MELON, analyze throughput and tail latency, demonstrate adaptation under dynamic and mixed workloads, and evaluate replication space–performance tradeoffs, showing that SwitchFS fully exploits the available device bandwidth in a rack-scale cluster and significantly outperforms existing DFS designs, including Conflux.

The remainder of this article presents the background, design, implementation, evaluation, and related work of SwitchFS. Section 2 describes the background, which includes our motivation of utilizing reinforcement learning for I/O path selection. Section 3 presents our design, including the MELON I/O policy and the model-driven replication strategy. Section 4 gives details about our implementation of SwitchFS. In Section 5, we evaluate SwitchFS and compare it with existing DFSs, including Conflux, focusing on throughput, tail latency, learning overheads, dynamic and mixed workloads, and replication tradeoffs. Section 6 discusses generalization, security, and integration issues, Section 7 shows related work, and Section 8 concludes the article.

2 Background and Motivation

2.1 PM and RDMA Hardware Evolution

The development of computer hardware technologies is rapid in recent years. Intel introduced the Optane PM series of non-volatile memory, which offers high bandwidth for load/store operations [23]. Meanwhile, Mellanox has launched the ConnectX-7 device for RDMA technology, which doubles the network bandwidth compared to its previous generation.

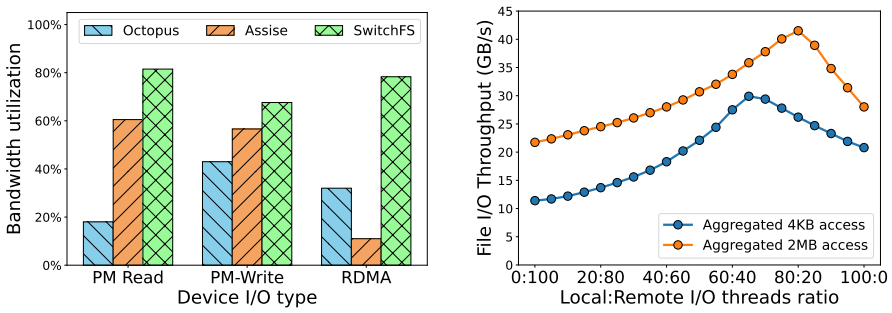
A large amount of existing works have focused on leveraging high performance PM and RDMA hardware and redesigning the file system software stack. The straight-forward design paradigm of adopting PM and RDMA is to store file system structures in PM and accelerate the data transmission via RDMA. Octopus [38] is a classical solution that follows this design paradigm. It creates a shared PM pool for file data storage and introduces RDMA-aware **remote procedure call** (**RPC**) mechanisms. The PM pool offers byte-addressable, persistent, and large-scale storage service compared to traditional disk devices. However, Octopus deploys PM on the server side, and its clients can access PM devices only through the RDMA network. Another design paradigm is to include PM not only on the server side but also on the client side as a local cache. Orion [60] and Assise [2] are two systems that follow this paradigm. They have demonstrated the benefits of using local PM, with lower latency and higher throughput.

Although our current prototype uses Intel Optane PM devices, Intel has discontinued this product line and the ecosystem is converging toward **Compute Express Link** (**CXL**), which also supports persistent memory devices. SwitchFS is intentionally media-agnostic such that it merely assumes a byte-addressable, low-latency, high-bandwidth storage tier that can be exposed either locally or remotely. The same architecture and I/O path selection mechanism apply when replacing Optane with CXL-attached storage or similar devices, with only the initial latency and bandwidth configuration used by our model changed.

2.2 Exploiting both Local and Remote PM

A common problem among existing RDMA-aware PM-friendly DFS designs is the low utilization of hardware performance. Some DFSs, such as Octopus, use PM only on the server side and do not exploit the local storage performance. Their performance is limited by the RDMA network bottleneck. Other DFSs, such as Assise and Orion, use local PM cache extensively on the client side and do not fully exploit the RDMA bandwidth. They evaluate the system performance in a testbed with 40 Gbps RDMA NICs, which hide the low network bandwidth utilization. We reveal the low network utilization problems using servers (see details in Section 5.1) that are armed with Mellanox ConnectX-6 adapter, which provides 200 Gbps network bandwidth. As newer network hardware continues to arrive, this mismatch becomes even more severe. For example, Mellanox ConnectX-7 provides 400 Gbps (i.e., 50 GBps) RDMA bandwidth, while modern PM or CXL-attached memory can offer tens of GBps per socket. As a result, both the local and remote bandwidths are ample in current and emerging systems, making it increasingly important to actively aggregate them rather than favoring a single path.

We conduct two experiments to show the improvement space for available bandwidth to user applications. The first experiment shows the device bandwidth utilization in different DFS designs. We generate multi-process workloads with concurrent bulk read/write requests using Fio [3] benchmark. We run the workloads on Octopus and Assise to collect their run-time bandwidth utilization statistics. The results are shown in Figure 1(a). We discover that they do not utilize hardware bandwidth sufficiently (as high as 84% of bandwidth wasted) because of the deterministic I/O path. Our proposed system, named SwitchFS, takes more advantages from the high bandwidth devices by utilizing both local and remote PM via dynamic I/O path selection.



(a) Bandwidth utilization. (b) Aggregated bandwidth.

Fig. 1. Space for hardware utilization improvement.

The second experiment shows examples of the bandwidth aggregation phenomenon, which exposes both local and remote storage devices to the client applications, leading to higher I/O performance. We generate 40 threads to access file blocks, in which a fixed percentage of threads perform local PM access, and the rest perform remote PM access through RDMA. We conduct experiments with 4 KB and 2 MB I/O size, and show the throughput in Figure 1(b). We draw two conclusions from the results: (i) Simultaneous usage of local and remote PM can provide much higher I/O bandwidth for applications without involving more CPU threads. If a system relies on local PM or remote PM access, it can only achieve 70% or 40% of the highest possible throughput. (ii) It is not a trivial problem to fine-tune the I/O system where we seek to utilize all available bandwidth. With different I/O sizes and concurrency levels, fixed ratio cannot exploit the highest throughput. To make it more complicated, the real-world workloads may contain mixed sizes of I/O and change their concurrency by time. In a cluster with distributed storage devices, the network bandwidth is a shared resource, making the I/O path selection more challenging. To deal with the complicated scenarios, we need an adaptive policy that has the knowledge of hardware characteristic while being aware of the environment changes.

2.3 Machine Learning for OS Design

The success of **machine learning** (**ML**) has motivated OS design researchers to automate the system tuning process. The learning-based designs and strategies have been widely applied to OS research field with promising results. Using the supervised learning paradigm, existing works have demonstrated the predictability of some operating system features, such as I/O performance and QoS metrics [8, 21, 34, 48]. For unsupervised learning, researchers have shown its debugging and prediction capability in the cloud computing context [7, 18, 46]. Nevertheless, it is worth noting that basic supervised learning methods always rely on intensive model pre-training and dataset preparation procedures, which are time-consuming and not adaptive facing dynamic environment.

Reinforcement learning (**RL**) is a machine learning paradigm that generates and evolves a policy adapting to a dynamic environment, taking different actions based on different run-time states, while optimizing the expected rewards. Several research works have shown that RL mechanism can achieve excellent results in OS design, especially for scheduling and placement problems [45, 47, 64, 65]. They design and test several effective RL methods including policy gradient and Q-learning [17, 53]. However, existing RL methods deployed in OS always encounter a more time-consuming training process compared to the other two types of ML schemes, due to the delayed reward and unstable exploration process.

There are two main challenges in applying RL methods to OS design. The first is the choice of optimization objective and the definition of **reward function**. Decisions made by a learned policy may affect the system performance in both short-term and long-term, explicitly or implicitly. Long-term implicit effects, such as resource utilization and cache hit rate, are hard to parameterize in the reward function. The second challenge is the balance of **exploitation and exploration** [25], which is a key point in the policy-learning strategies in RL research area. Exploitation helps a policy to reduce regretful action or survive longer in the game-playing texture. On the other hand, exploration is vital for an RL model to explore the possible unrevealed reward of under-estimated actions and avoid suboptimal choices in long-term execution. Practical balance strategies include ϵ-greedy, Boltzmann exploration, and UCB [49, 52].

Our method, named MELON, employs a lightweight RL method that has **immediate rewards and stable exploration mechanism**, which is distinct from existing RL-based OS policy designs.

3 SwitchFS Design

In this section, we present a new DFS architecture named SwitchFS. As an overview, SwitchFS adopts the client-server architecture for cluster organization. It focuses on improving data I/O scalability and achieves the following design goals:

- **Exploiting more available PM bandwidth.** The main goal of SwitchFS is to transparently expose local and remote PM device bandwidth within a cluster to the applications. At the client-side, SwitchFS introduces a novel dynamic I/O management mechanism, managing PM cache and RDMA connections to activate different I/O paths. At the server-side, it introduces a model-driven replication scheme to improve the bandwidth utilization across the cluster.
- **Learned I/O path selection at run-time.** SwitchFS introduces a novel I/O path selection policy named ***Multi-path lEarned Latency OptimizatioN*** (**MELON**). MELON incorporates an RL model to learn from the performance statistics logs of SwitchFS and choose the I/O path with low expected latency. Moreover, it is lightweight while having adequate learning capability to adapt dynamic storage pressure and network pressure.
- **Fine-grained I/O concurrency control.** The centralized coarse-grained concurrency control mechanism is a bottleneck for concurrent I/O. SwitchFS comes up with a fine-grained tree-structured data lock at the server side and a lease mechanism at the client side. They cooperate to provide high parallelism for concurrent I/O requests.

Compared with our prior work Conflux, SwitchFS retains the PM utilization, RDMA data plane, and basic FS-service architecture, but replaces the supervised-learning based I/O mode selector with the online RL-based MELON policy. It also adds the model-driven opportunistic replication logic on the server side, and refines the fine-grained lock and lease mechanisms to support sub-file granularity and an explicit consistency model.

The software components of SwitchFS are shown in Figure 2. Applications are running at the client side, where SwitchFS serves all the application issued I/O requests and pushes them through three main software components. First, the fine-grained concurrency controller is activated, granting access admission to the working thread. Then the I/O request structure passes through MELON, which generates an I/O path decision according to the request metadata and its run-time I/O environment estimation. Finally, the request is sent to SwitchFS I/O manager where local and remote I/O tasks are performed simultaneously. Meanwhile, a distributed FS-service is designed at the server-side to establish a PM pool, maintaining the file system structures and data regions. It utilizes a model-driven replication mechanism and the fine-grained lock and lease mechanism, providing a scalable and consistent data I/O service to clients.

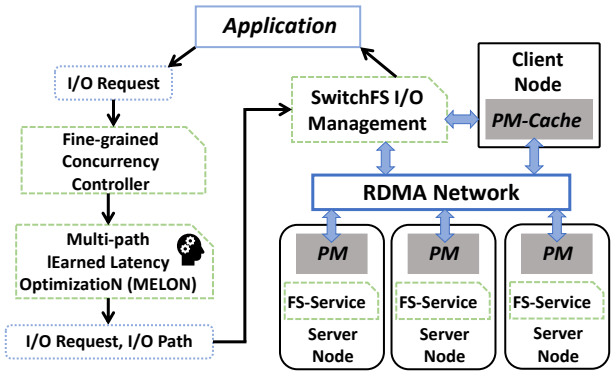


Fig. 2. SwitchFS architecture overview.

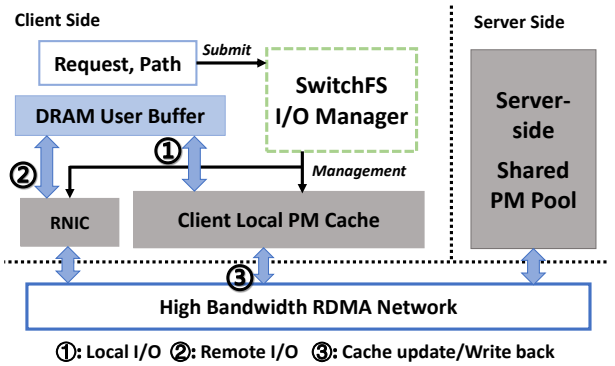


Fig. 3. SwitchFS dynamic I/O management.

3.1 SwitchFS I/O Management Mechanism

On the client-side, SwitchFS employs a unique I/O management mechanism that exploits both local and remote I/O bandwidth for user applications. SwitchFS I/O manager receives I/O requests from the application at run-time and dynamically selects an I/O path among the local cache and remote replica nodes to serve the request. On the server-side, SwitchFS builds and maintains the file system through FS-service, which stores both metadata and data segments in PM regions. Each file's metadata resides in a primary server node, determined by a consistent hash algorithm. FS-service also collects all nodes' data segments into a PM pool and exposes it to clients.

The dynamic I/O management mechanism is illustrated in Figure 3. As explained earlier, when our system receives application requests, it will generate corresponding I/O path advice from the policy engine and forward them to the SwitchFS I/O manager. The SwitchFS I/O manager maintains a client local PM cache space and organizes RDMA connections to the server-side PM pool. Thus, it is able to activate both local and remote I/O functions, targeting different nodes in the cluster to serve the incoming I/O tasks. Moreover, it manages the cache update and write back tasks to maintain the cache consistency. SwitchFS I/O manager initiates an I/O thread pool for each of the three I/O types. As a result, the local and remote I/O are executed simultaneously under parallel requests.

The SwitchFS client cache includes both data and metadata. Metadata cache maintains file status (e.g., file mode, size, access time) information, location of its primary node and replica, as well as an address translation table. They reduce the cross-node communication overhead, facilitating straight-forward remote I/O and local I/O tasks. As for metadata consistency, server-side CPUs are involved in metadata updating, where in-DRAM per-inode atomic locks ensure a strong consistency. For data segment, it uses a block-aligned cache with MESI protocol ensuring cache coherence. When I/O request refers to data blocks that are currently not in local PM, new space will be allocated in the data segment. When the local PM space is full-filled, SwitchFS issues batched eviction, where the victims are selected by an LRU algorithm. Therefore, the data cache has its own metadata, which includes block status and a table mapping the local cache address to the shared PM pool. It also facilitates the address translation for remote I/O tasks.

3.2 MELON: Learning-Based I/O Path Selection

We design a novel run-time I/O path selection policy named MELON. MELON adopts a reinforcement learning paradigm that is modeling the relationship between I/O path choices and their benefits to the latency value, and estimates a proper policy accordingly.

As an overview, MELON records and manages all the I/O tasks information in the cluster and trains the model in an online learning manner, where all SwitchFS clients maintain their own copies of the model and update them with shared training data. We introduce an I/O request monitor to collect the I/O metadata and prepare inputs for both model inference and model training process. We devise to adopt the Q-learning paradigm in MELON with runtime latency reduction as the reward, which provides immediate feedback and facilitates the evolution of a lightweight backbone neural network model. With respect to the balance between exploitation and exploration, MELON introduces a variant of Boltzmann exploration that allows the model to choose arbitrary I/O paths other than the predicted fastest one with certain probabilities, leading to a stable reinforcement learning process.

In RL terms, MELON observes a state S that encodes the current I/O environment, selects an action A corresponding to a particular local or remote I/O path, and receives a scalar latency-based reward R after the I/O completes. SwitchFS adopts a centralized training and a distributed inference architecture within each client. A single training thread maintains and updates the primary model, while worker threads hold read-only copies that are used for per-request inference on the critical path.

We introduce the different design aspects of MELON as follows:

3.2.1 Global Online Learning. MELON uses a global online learning architecture that can train the neural network efficiently while keeping the training data shared between different clients. By separating the inference process and the training process, MELON achieves a balance between low CPU overhead and high prediction precision in each client. By using a shared training data pool, MELON successfully distributes learned I/O performance knowledge across the cluster and maintains consistent models among clients.

In each client node, MELON maintains a primary trainable model and multiple inference models, each for an I/O worker thread. As shown in Figure 4, there is a dedicated thread in each client node that continuously trains the primary model with the collected I/O metadata. Though all the SwitchFS working threads keep their own copies of the model, they only perform the inference operation and do not execute the training process. Due to the fewer arithmetic operations included, the neural network inference time costs are much lower than the model training costs, which reduce the latency overhead on the critical path of I/O tasks. Meanwhile, the centered model training architecture allows batched data feeding in training process, which is more efficient than the single

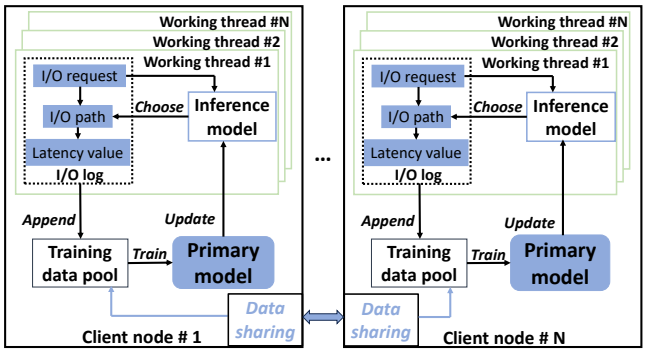


Fig. 4. Global online learning architecture of MELON.

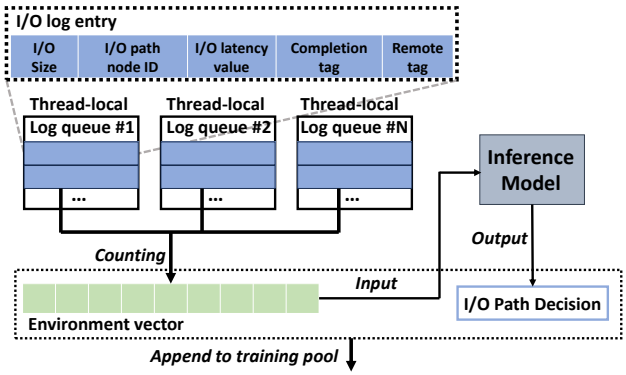


Fig. 5. I/O request monitor generating inputs to MELON model.

data feeding in a neural network model. When MELON discovers a significant error decrease at the end of a training epoch, it persists the new model in a new file and informs all SwitchFS worker threads to update their inference models.

To keep a consistent inference model across different clients, MELON utilizes a shared training data pool. When a client trains and updates its primary model, it pushes the corresponding training data batch to all the other clients and inform their training thread to append the new data to their training data pool. To avoid model divergence resulted by local device performance estimation, when a MELON client receives a training data batch from others, it filters the data by selecting I/O logs (see details in subsection 3.2.2) with the remote I/O path tag. In this way, all the clients train their primary models with the same set of training data eventually, leading to a consistent latency model that estimates the RDMA devices' performance within the cluster.

3.2.2 I/O Request Monitor. MELON tracks all the I/O requests to collect and log their execution information within a SwitchFS cluster, which generates the input vectors including I/O task environment and facilitates the model training process.

As shown in Figure 5, SwitchFS maintains an in-DRAM log area for every client worker thread to record their I/O request metadata, which consists of I/O size, I/O path indexed by the target node ID, and its ground-truth latency. Specifically, the latency value is initialized as zero and updated as the real latency right after the I/O task is completed. These I/O information logs are

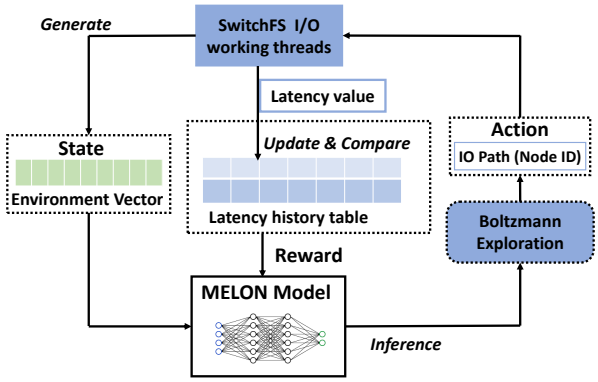


Fig. 6. The Q-learning paradigm in MELON.

transformed into a fixed-size I/O environment vector, which are used as the input of the neural network model during the training and inference process. Different positions of the I/O environment vector represent I/O pressure of different I/O types and sizes, where the sizes are determined by their highest bit, and the values in each position are counted by the number of I/O requests in parallel.

Each I/O request log is a circular buffer operated as a queue structure, where the user applications act as the producer and the MELON model trainer as the consumer. Meanwhile, the inference models in the working threads are granted to read the logs to generate I/O environment vectors as the input of the neural network. As I/O request logs from multiple SwitchFS threads are shared among all the working threads in the client node, MELON can generate comprehensive estimation of the I/O pressure on local and remote access paths from the view of each client. Moreover, to facilitate the shared training data pool among different clients, all the I/O request logs with non-local target node ID are tagged as remote I/O tasks.

3.2.3 Q-Learning with Latency Reward. MELON adopts the Q-learning [17] paradigm to model the relationship between run-time I/O environment, I/O path choices and their impacts on the latency value.

As shown in Figure 6, the I/O environment vector is used as the ***State(S)*** value in the Q-learning paradigm. The ***Action(A)*** set is defined as all the possible I/O path choices, including local and remote node IDs in the cluster. The ***Reward(R)*** is designed as the latency reduction of the current I/O task. To determine the runtime latency reduction, MELON maintains a historical average latency table for different I/O types and sizes, where each entry represents the approximated average latency L_{ave} of recent I/O tasks. When new I/O tasks are completed, MELON updates the average latency table with the new latency value

$$L'_{ave} = \eta L_{ave} + (1 - \eta) L_{new}, \tag{1}$$

where the update rate η is settled as 0.99 by default, which will smooth the average latency value and avoid over-fitting to the most recent I/O tasks. With the historical average latency, MELON calculates the reward as

$$R(A) = -log\left(\frac{L_{new}}{L_{ave}}\right), \tag{2}$$

which encourages the path selections that lead to reduced latency and penalize ones that lead to increased latency. For the estimation of $Q(S, A)$, we use a lightweight neural network model with

three fully connected hidden layers. It only contains about 4.4K float parameters such that the whole model can be stored in a file of less than 16 KB size. While introducing low CPU and memory overhead for real-time prediction process, it satisfies the accuracy requirement at stable states.

Compared to supervised learning paradigms, RL is more suitable for the I/O path selection problem, as it can capture the potential of different choices and find a proper policy accordingly. For an I/O task with known size, a lower latency indicates better exploitation of the hardware bandwidth, thus MELON helps fully take advantage of the dynamic I/O mechanism in SwitchFS. Moreover, the latency value is returned immediately after the I/O task is completed, which provides online reward and relieves MELON from long-term parameter training facing new workloads and possible hardware changes. It is also a lightweight measurement reflecting the I/O pressure, which requires moderate CPU resources for metadata collection, making real-time decision-making feasible.

Importantly, MELON does not need pre-training or offline data collection process. It starts with a randomly initialized model and learns from the real I/O tasks at run-time, which is facilitated by the immediate I/O latency reward. MELON also provides a model reloading mechanism, which allows the model to be saved and loaded from files to preserve the learned knowledge across application runs and system reboots. Converging to a stable I/O path selection policy, MELON continuously improves the overall I/O performance in SwitchFS when facing dynamic workloads and changing hardware environments.

3.2.4 Automatic Exploration Strategy. To balance the exploration and exploitation ability of our RL model, MELON utilizes the Boltzmann exploration strategy and includes an automatic temperature adjustment scheme.

With different action a_i, their Q values are estimated as

$$q_i = Q(S, A = a_i). \tag{3}$$

MELON adopts the policy that has a probability of choosing a_i as

$$P(a_i) = \frac{e^{q_i/\tau}}{\sum_j e^{q_j/\tau}}, \tag{4}$$

where j indexes all the possible actions within the action set, and τ is the temperature factor of Boltzmann exploration. With the definition of Equations (1), (2), and (4), when the neural network training process approximates $Q(S, A)$ that converges to $R(A)$, and temperature τ diminishes to zero, MELON is guaranteed to find the optimal I/O path selection policy that minimizes the expected I/O latency.

For the temperature factor adjustment strategy, MELON initializes $\tau = 1$ and decreases it by a factor of 0.95 after every training epoch of the MELON client primary model if the training loss is decreasing. When the training loss is not decreasing or even increasing, MELON will increase the temperature by a factor of 1.05 to encourage more exploration. The selection of initial temperature factor $\tau = 1$ is moderate in the following sense: with this temperature, when $Q(S, A) = R(A)$, the probabilities of choosing different I/O paths are reversed proportional to their latency value, which is a policy with both exploitation and exploration ability.

A potential exception caused by this automatic exploration strategy is called the unavailable I/O path. It happens when MELON makes a remote I/O path decision, while the corresponding server node does not hold requested file blocks. SwitchFS then triggers a local I/O task as fallback to fix this issue, and sends a record including the unavailable file inode and its timestamp to the primary server of the related file. Our model-driven opportunistic replication strategy in Section 3.3 resolves the unavailable I/O path problem in an opportunistic manner.

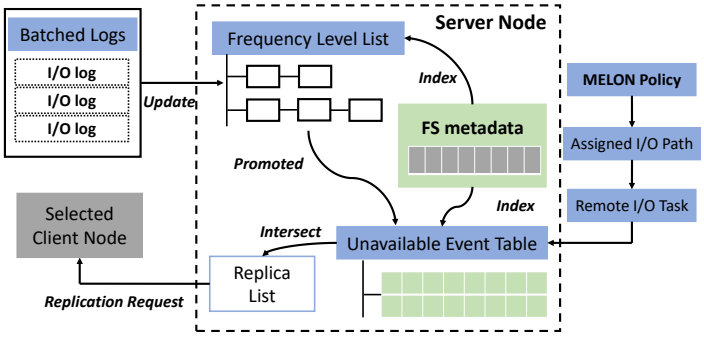


Fig. 7. The model-driven opportunistic replication mechanism.

3.3 Model-Driven Replication

SwitchFS introduces a model-driven opportunistic replication mechanism, which strategically allocates extra storage space in under-utilized server nodes, to store additional replicas for frequently accessed files. It identifies the requirement for adjusting the number of replicas by considering the file access frequency level, and decides the placement of new replicas by considering the model-generated unavailable I/O path events. With this model-driven replication strategy relying on the learned I/O path, SwitchFS replicates file blocks on-demand and optimizes the locations of these replicas using the model's predictions, which leads to an improvement of the overall I/O performance from the server-side. Replication operates at the granularity of fixed-size data blocks, where the replication factor is tracked per file, but only blocks that are actually accessed and promoted by the access-frequency mechanism are replicated. Through this approach, for a very large file with a small hot region, only the hot blocks incur additional replicas and space overhead.

As shown in Figure 7, SwitchFS first maintains a replication factor for each file, which decides the number of replicas stored in the servers. The default value is set as 3, a wide-adopted replica count in distributed file systems [24, 42, 54], and SwitchFS adjusts the replication factor according to the access frequency of files at run-time. To record the frequency information, SwitchFS maintains a global access frequency table and updates it at each file system function call. It is indexed by the inode number and distributed across the cluster by the FS-service at hosts. The access frequency table is organized as multi-level lists, of which each level represents a distinct access frequency range. When the access frequency of a file exceeds twice the lowest of its current level, SwitchFS will promote it to the upper level. This frequency-based promotion mechanism ensures that the most frequently accessed files are replicated more times, which improves the availability and performance of the critical data.

Second, SwitchFS selects a new server node to replicate the file that has a promoted replica level, and updates the file's metadata on its host node. The new replica location is determined by the unavailable I/O path events reported by the RL models at client nodes. As we mentioned in Section 3.2.4, the unavailable I/O path events represent unsatisfied remote I/O decision driven by our MELON policy, which are stored in a global visible table indexed by the inode ids at the file's host node. When a new replica is planned for a file, the server node query the unavailable I/O tables, as well as all its existing replica locations, and selects a target node with the highest frequency of unavailable I/O while not holding the replica. If the target node is fulfilled, victim of evicted files will be randomly selected from the replicated ones with the lowest replication factor. This model-driven opportunistic replication strategy allows SwitchFS to make more remote

device bandwidth available by replicating frequently accessed files to under-utilized nodes, which is selected by the MELON model predicted remote I/O events.

By prioritizing replication for frequently-demanded contents, SwitchFS provides wider available I/O paths to critical data and improve the overall system performance. By selecting the node with the highest frequency of unavailable I/O events, SwitchFS ensures that the new replica is placed in the most demanded location estimated by MELON policy. Through dynamic-level replication allocation, SwitchFS enhances device accessibility across the network, while promoting scalability and fault tolerance. This proactive resource distribution reduces the access latency by mitigating potential bottlenecks caused by network conjunction, bolstering system resilience and availability in dynamic I/O-intensive contexts. To bound replication overheads, the maximum per-file replication factor is configurable, and Section 5 quantifies the resulting space-performance tradeoffs.

3.4 Fine-Grained Concurrency Control

SwitchFS introduces the lightweight fine-grained concurrency control mechanism to facilitate concurrent I/O requests. At the server-side, it introduces a fine-grained sub-file level data lock, which improves shared file I/O scalability. At the client-side, SwitchFS designs a lease control mechanism that reduces network communication.

These mechanisms together define SwitchFS's data consistency guarantees. Let $W(f, [o, o+l))$ denote a completed write to file f covering byte range $[o, o+l)$, and let $R(f, [o, o+l))$ denote a read of the same range. SwitchFS enforces the following invariants:

- **Per-client read-after-writes.** For any client c, if a write $W_c(f, [o, o+l))$ completes before a subsequent read $R_c(f, [o, o+l))$ issued by the same client, then R_c observes at least the value written by W_c (or a newer value).
- **Single-writer, multi-reader serialization.** Concurrent writes to overlapping byte ranges of the same file are serialized by the server-side tree-structured locks, and readers are ordered with respect to these writes, so that all clients observe a total order of writes to each byte range.
- **Replica consistency.** For lock-protected ranges, data written by a completed $W(f, [o, o+l))$ is propagated to all replicas chosen by the model-driven replication mechanism before the write is acknowledged, so that subsequent reads from any replica respect the same write order.

For regions that are accessed without overlapping writes (e.g., disjoint ranges in a shared file), the combination of sub-file locks and client-side leases provides POSIX-like close-to-open semantics. Once a lease is granted and the corresponding write or read completes, subsequent operations on that range see a consistent view across clients.

As shown in Figure 8, the lock and lease mechanisms cooperate in providing a fast concurrent data locking service to applications. The fine-grained data lock is applied in SwitchFS servers. Every inode is related to a lock structure. For every file, SwitchFS maintains an in-DRAM tree-structured lock group. The tree-structured lock group uses block size as its granularity. It allows concurrent write operations within the same file, as long as they do not overlap on blocks. It organizes all requested lock entries as a binary search tree to provide higher efficiency. Here we denote N as the maximum between the number of concurrent I/O requests and the block number occupied by the file. SwitchFS finishes every lock/unlock operation within an $O(logN)$ time-complexity algorithm, which is a small overhead for either large files or highly concurrent I/O demands. For many real-world I/O intensive applications, the concurrent data I/O can occur within shared files [16, 35, 44, 63]. That indicates the fine-grained lock will improve scalability under I/O-heavy workloads with severe contention.

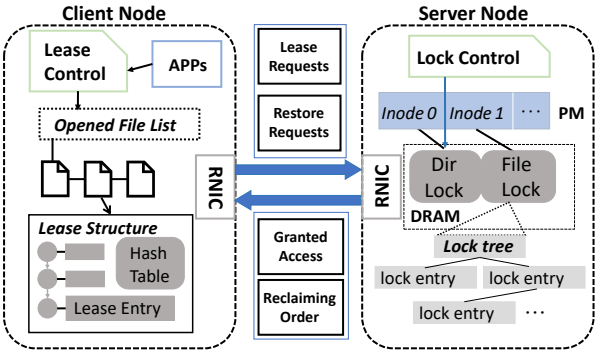


Fig. 8. The fine-grained lock and lease mechanisms.

Furthermore, a lease controller resides in each SwitchFS client thread to maintain thread-local lease entries. The lease controller creates a lease entry list for every opened file descriptor. When a *read/write* request successfully acquires a lock on an address range in a file, a new lease entry is added to the corresponding list. Subsequent I/O operations with an address range that is contained in the lease list will be admitted without querying servers. The existing lease entries are indexed by a hash table. Thus, repeated I/O requests can get admission with several local memory access latency, which significantly improves over the remote lock acquisition. On file *close* operation, all acquired leases will be marked as invalid. Moreover, the lease reclaiming operations are asynchronous and batched, which helps shorten the lease reclaiming critical path and preserve more RDMA bandwidth for data I/O operations in the SwitchFS cluster.

4 Implementation

We implement SwitchFS as user-level libraries written in C. LibServer (6.7K LOC) is for server utilities. It includes the implementation of FS-service, model-driven replication scheme, and the lock control functions. LibClient (6.6K LOC) is for client utilities. It includes the implementation of SwitchFS I/O manager, MELON learning paradigm, and the fine-grained lease control module.

4.1 LibServer

LibServer is the software component running at the SwitchFS server side. We implement it as a user-level module that follows the design of FS-service and the lock control mechanism. In our implementation, LibServer is initialized with a thread pool such that it can scale to serve massive concurrent applications. LibServer uses shared memory for lock structures and IPC to coordinate concurrent server-side processes, as some existing works suggest [40]. RDMA connection management, message send/receive, and data read/write are implemented using `verbs` API provided by the RNIC driver. All metadata, replication, and locking logic runs in software on commodity RDMA NICs such that SwitchFS does not rely on programmable switches or in-network computation.

4.2 LibClient

LibClient is also implemented as a user-level module. It intercepts all POSIX system calls related to file system operation and invokes SwitchFS internal functions. We view the return of RDMA *write* as the finish point of remote data updates, though current RNIC hardware cannot guarantee *write* persistency. Some existing literature has shown the feasibility of ensuring data persistency in one

network round trip [14]. We also notice that **data direct I/O (DDIO)** technology, an optimization option in current x86 CPU architecture, can be harmful to PM-involved remote data persistence [26]. However, SwitchFS makes some of the DRAM buffer hot regions for cross-node messaging. Thus we keep the DDIO option on in our implementation. Within each client process, MELON is implemented as a dedicated training thread plus per-worker inference instances. The training thread runs in the background and updates the primary model, while I/O worker threads only perform lightweight inference on a local copy of the model, keeping MELON's CPU overhead off the critical I/O path. Section 5 quantifies the CPU and memory overhead of MELON compared to a non-learning baseline. We use a Python module, built with Tensorflow [20], to specify the neural network architecture and generate the training loop. It invokes the inference model from LibClient through the Tensorflow Lite [51] C API so that all inference run inside the client process without invoking the Python interpreter.

5 Evaluation

In this section, we evaluate SwitchFS and compare it with existing DFS designs. We describe the experimental setup and evaluate SwitchFS with microbenchmarks, macrobenchmarks and real-world applications. Moreover, we show the effectiveness of MELON I/O path selection policy, performance analysis for different design parts, and the resource utilization of SwitchFS.

5.1 Experimental Setup

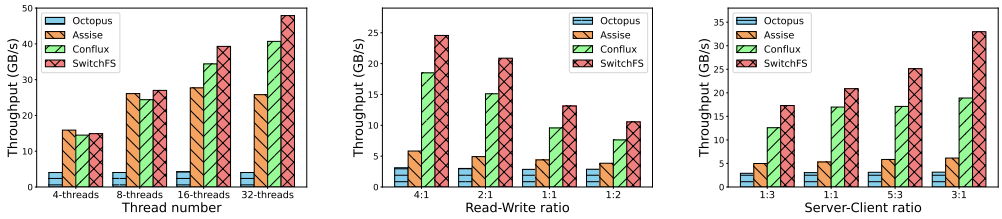
In our cluster, each server and client node is equipped with dual-socket Intel Xeon Gold 6348 CPU, of which a single socket has 28 physical cores. For memory, each node has eight 16GiB DDR4 DRAM and four 128 GB Optane persistent memory modules. For the network, each node has a Mellanox ConnectX-6 HCA, which supports 4X HDR (200 Gbps) and 4X EDR (100 Gbps) InfiniBand links.

We compare **SwitchFS** with existing DFSs designed for PM and RDMA, including **Octopus** [38, 68] and **Assise** [2], on microbenchmarks. Since we use filebench [50] for macrobenchmark and Octopus does not fit in this benchmark, we replace it with **NFS over RDMA** with the help of MLNX OFED driver's support and use EXT4-DAX [55] as backend file system for NFS server-side exporting. We also conduct evaluation with real-world applications including RocksDB [16] and SQLite [41], and compare the performance of SwitchFS with existing DFSs. Our comparison systems also include the base version named Conflux [43], which does not include the MELON policy and the model-driven opportunistic replication mechanism, to show the effectiveness of our design.

For cluster organization, we use four nodes for file system servers and four nodes for the clients in SwitchFS by default. All benchmark processes are executed in client-side, and the results are collected and averaged through all the client nodes. For Assise, we configure the *hot replica* number as two. In evaluation, we limit the CPU usage of all working threads in a single CPU node by using `numactl`, and we also bind the PM allocation and RDMA NIC port to the same NUMA node for stable performance [31, 62].

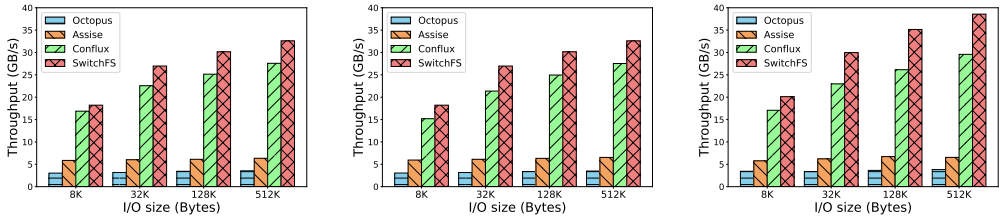
5.2 Microbenchmark Results and Analysis

First, we conduct multi-threaded read/write bandwidth tests using Fio, with various thread numbers, read/write ratios, and server-to-client ratios. Figure 9(a) shows the results of concurrent read workloads, where each thread allocates a private file and reads its content in 256 KB I/O size. Octopus cannot scale as the thread number exceeds eight, so that its bandwidth is lower than Assise and SwitchFS. Assise is able to fully utilize the local PM bandwidth but cannot scale up facing higher concurrency level. Conflux outperforms Octopus and Assise by 1.6× by utilizing remote



(a) Read-only workload. (b) Read/write mixed. (c) Various server-to-client ratio.

Fig. 9. Bandwidth with different R/W ratio and server-to-client ratio.

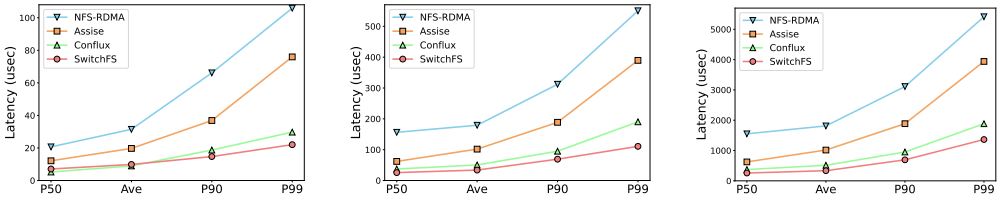


(a) Sequential access. (b) Random access. (c) Skewed access.

Fig. 10. Bandwidth with different access pattern.

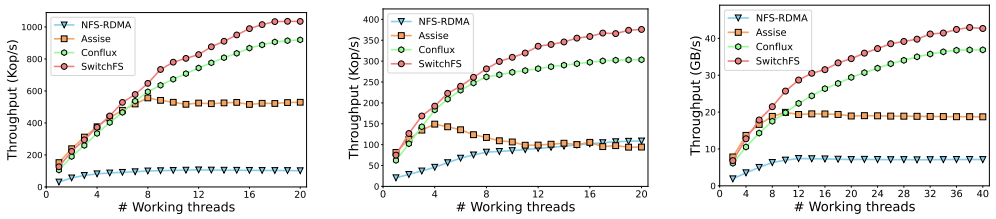
I/O bandwidth, while it cannot fully utilize the remote PM across the cluster under various server-client ratio. SwitchFS outperforms Conflux by a large margin because it can exploit the remote PM bandwidth adaptively. Moreover, we show the results of mixed read/write workloads with different server-to-client ratio in Figure 9(b). A read/write mixed workload leads to performance degradation in DFSes as there are read-write contentions at the PM bus level. SwitchFS is able to identify and resolve such contention by learning and reducing the runtime I/O latency, where the reduction of latency is a result of steering I/O requests into under-utilized I/O paths. As Figure 9(c) shows, SwitchFS can scale up well with the number of server nodes. When there are more server nodes available, SwitchFS is the only DFS that adaptively improves the I/O throughput because its opportunistic replication mechanism exploits more remote device bandwidth. We also conduct evaluation with various I/O sizes and access patterns, with results shown in Figure 10(a), (b), and (c). SwitchFS outperforms the other three systems under various access patterns and granularity. It has no performance degradation when serving random requests, as the fine-grained lock mechanism supports scalable I/O with page-level access control.

Second, we conduct multi-threaded latency tests with various I/O sizes, and show the median, average, P90, and P99 latency results in Figure 11. We use Fio [3] to create 1 GB shared files and each thread performs sequential r/w on it. Figure 11 shows the P99 latency results. Octopus has higher latency than the other two DFSs, because it takes static remote I/O path. SwitchFS has a similar latency as Assise for 8 KB I/O size because MELON rarely affect the I/O path selection for small I/O sizes, and it has lower latency for larger I/O sizes as it chooses the less congested I/O path at runtime. It is worth noting that under 8 KB I/O size, the latency of SwitchFS is slightly higher than Conflux, because the MELON policy increases the inference overhead for each I/O path selection. In our implementation, the lightweight RL model inference process introduces 3.2 us CPU time overhead to each I/O task. However, under highly concurrent workloads and larger I/O



(a) I/O size = 8K. (b) I/O size = 128K. (c) I/O size = 2M.

Fig. 11. Multi-threaded I/O latency distribution with various I/O sizes.



(a) Varmail results. (b) Fileserver results. (c) Webserver results.

Fig. 12. Performance on filebench workloads.

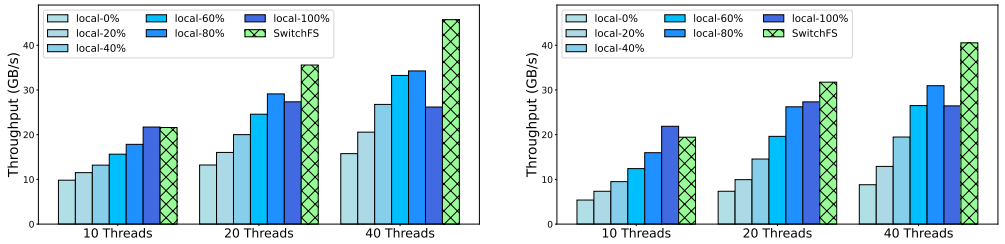
sizes, the model inference overhead is always negligible compared to the latency reduction due to optimized I/O path.

5.3 Macrobenchmark Results and Analysis

We use *varmail*, *fileserver* and *webserver* from Filebench [50] as macrobenchmarks.

We configure varmail and fileserver to operate on a working set with 10 K files, where the average initial file sizes are 16 KB and 128 KB, respectively. The overall read-to-write ratios are 1:1 and 1:2 in varmail and fileserver. As Assise's current implementation does not support the multi-thread execution of filebench, we use multiple processes instead. Figure 12 shows the results. NFS over RDMA has low throughput as it needs intensive data copy from server-side PM storage to the client-side buffer cache. Assise stops scaling at eight threads for varmail and four threads because there is centralized kernel module for log digestion in Assise. SwitchFS outperforms the other two DFSs, and even Conflux, by as much as 2.1×. Because it replicates the hot files to different remote PM devices, and it chooses the lower latency I/O path for each request.

To evaluate the system with larger working set, we use Webserver with 40 K files and 1 MB average file size. We also configure the access pattern so that the generated file sizes follow a Gamma distribution. Consequently, this benchmark send I/O requests with various I/O sizes. Figure 12(c) shows the results. Because there is no read-write contention, NFS over RDMA can also scale within 12 threads. Assise can exploit about 80 percent of the local PM bandwidth for this workload. Conflux can exploit both local and remote PM bandwidth, such that it outperforms the other two DFSs by 5× and 2×. SwitchFS can further improve the performance by 1.7× by utilizing the MELON policy with an opportunistic replication mechanism.



(a) Infiniband rate 4X HDR (200 Gbps). (b) Infiniband rate 4X EDR (100 Gbps).

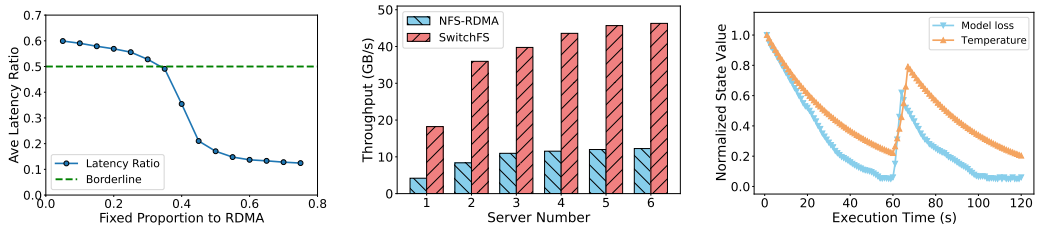
Fig. 13. Comparison with static I/O path selection policies.

5.4 Effectiveness of MELON Policy

5.4.1 Adaptiveness of MELON. We conduct a series of experiments to show that MELON can optimize the overall I/O performance adaptively, under various I/O size, concurrency level, and storage load factors.

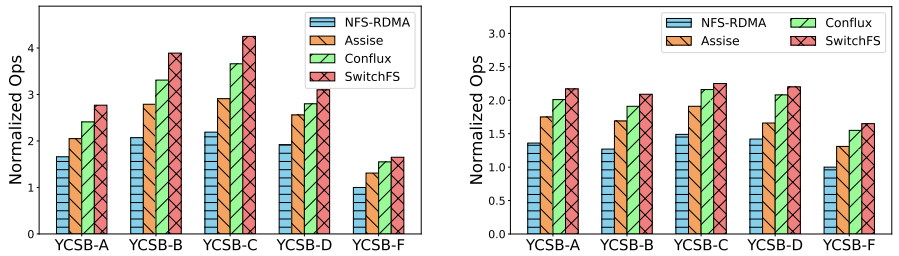
We configure different static I/O policies in SwitchFS as baselines, i.e., using a series of fixed local to remote I/O ratios ranging from 0:100 to 100:0, and keep the server-to-client ratio as 3:1 to provide enough network bandwidth for remote I/Os. We use webserver benchmark and show the throughput results under different concurrency levels (thread numbers) in Figure 13(a). Generally, SwitchFS outperforms all the fixed-ratio I/O path selection policies. For the 10-thread benchmark, we find that overall throughput increases with the local I/O ratio. It is because the concurrency level is not enough to saturate the local I/O bandwidth. SwitchFS successfully finds this situation and assigns almost all the I/O to local PM, and achieves similar performance as the 100% local I/O policy. For 20-thread and 40-thread benchmarks, we find that some of the fixed-ratio remote I/O policies perform better than the all-local policy. SwitchFS, however, performs better than all the fixed-ratio policies. Even under different RDMA rates, as shown in Figure 13(b), SwitchFS can adapt to the changing I/O environment and optimize the I/O path selection policy at runtime. Through modeling the I/O latency, SwitchFS applies different policies to different I/O sizes, which is more advanced than any of the fixed-ratio policies and leads to better bandwidth utilization.

5.4.2 Advantages of RL-Based Policy. Compared to Conflux, our MELON policy utilizes the reinforcement learning framework to optimize the I/O path selection at runtime. We conduct evaluation to show that MELON: (1) can adapt to different I/O environments and optimizes the I/O path selection policy. (2) can converge to a stable policy with a moderate temperature level. We start with a 32-thread fio workload with 256 KB I/O size and 1:1 server-client ratio. Capturing the relationship between *Latency Ratio* and remote I/O ratio in Figure 14(a), we find that the latency-balance point is 0.34, meaning that the optimal remote I/O ratio is about $\frac{1}{3}$. Denoting the PM bandwidth as BW_{PM}, we calculate that the best aggregated bandwidth under the 1:1 server-client ratio is $(1 + \frac{0.34}{1-0.34}) \times BW_{PM} \approx 1.5 \cdot BW_{PM}$. Under the same workload, Figure 14(b) shows SwitchFS performance with different configurations. The PM bandwidth is reached by setting I/O path to local-only. MELON helps SwitchFS to exploit more bandwidth by steering the I/O requests to remote PM devices, which surpasses the optimal 1:1 server-client ratio performance. It reaches the peak performance (1.9× local access bandwidth) at its steady status under 3:1 server-client ratio, as there are more remote PM devices available. The throughput under different server-client ratio is shown in Figure 14(b), which shows that SwitchFS scales well with the number of server nodes by using the opportunistic replication mechanism.



(a) Latency Ratio v.s. Remote I/O. (b) Throughput under various #server. (c) Training loss and temperature.

Fig. 14. Advantages of the RL-based I/O policy.



(a) YCSB on SQLite. (b) YCSB on RocksDB.

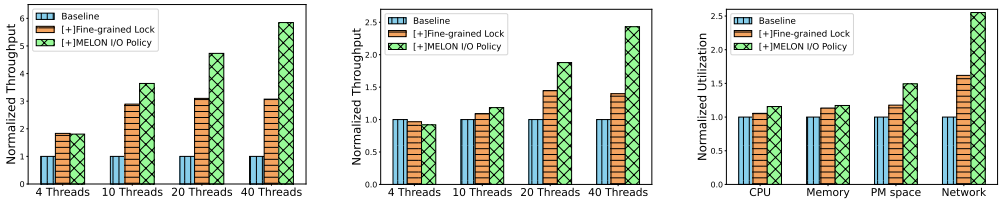
Fig. 15. Real-world application performance.

We also conduct a two-stage dynamic workload that changes the server-client ratio from 1:1 to 3:1 after 60 seconds, and show the training loss and temperature of the RL model in Figure 14(c). The training loss converges to a low value after 53 seconds of execution and the temperature decreases continuously. After the server-client ratio changes, the remote I/O pressure decreases, and the training loss increases rapidly. SwitchFS successfully identified this situation and increased the temperature to explore more I/O paths. After another 37 seconds, the training loss converges to a new low value, indicating that the RL model has learned a new I/O policy that is adaptive to the new I/O environment.

5.5 Real-world Application Performance

To evaluate the performance of SwitchFS with real-world applications, we take RocksDB [16] and SQLite [41] as examples of key-value store and relational database, and utilize the YCSB [6] benchmark to generate KV workloads for these two applications. RocksDB is a well-known persistent database based on LSM-tree. It is designed as a library component embedded in higher-level applications and optimized for large-scale distributed utilities [13]. SQLite is a widely used embedded relational database, which is originally designed for single-node applications and optimized for low-latency access. We run the YCSB workload A, B, C, D, and F on these KV applications in each client node, with 32 threads and 10 M key-value pairs.

Results are shown in Figure 15. SwitchFS outperforms the other three systems in all the YCSB workloads for both applications. Specifically, SwitchFS outperforms Conflux by up to 17% for SQLite and 8% for RocksDB, which shows the effectiveness of the MELON policy. The performance gap between SwitchFS and Assise is larger for YCSB-B and YCSB-C on SQLite, which is a result of the



(a) Bandwidth test. (b) Webserver test. (c) Resource utilization.

Fig. 16. Performance breakdown and resource utilization.

random read access pattern. SwitchFS identifies the cacheable read requests and serves them with local PM, while it still serves the hot data read requests with remote PM to avoid the I/O contention on local devices.

5.6 Performance Breakdown and Resource Utilization

To figure out how the different design components affect the performance of SwitchFS, we conduct experiments to break down the performance results. We use Assise as the baseline, and activate the implementation of different design parts of SwitchFS to compare them. The '[+] Fine-grained Lock' system adopts the lock and lease mechanisms described in Section 3.4, but does not include local-bypass I/O policy. The '[+] Learned I/O path' system is identical with our description of SwitchFS, which can dynamically choose the I/O path under MELON generated policy.

Figure 16 shows the performance of different system fragments under bandwidth tests and webserver workload. The workloads' configurations are identical to those in Section 5.2 and Section 5.3. We select four different working thread numbers to observe the performance breakdown with various concurrency. We find that both the fine-grained lock design and the adaptive I/O path selection policy benefit a lot to SwitchFS. For workloads above 10-thread, the learned I/O path selection, which is enabled by our MELON policy, improves the throughput by a large margin. Because the high concurrency level makes space for I/O bandwidth aggregation, and MELON helps to utilize both local and remote PM devices. In some case, e.g., the 4-thread webserver workload, SwitchFS achieves similar throughput as the baseline system. The low concurrency workload can not benefit a lot from our design inherently. To conclude, SwitchFS improves the I/O performance for high concurrency workloads while sacrifices little for low concurrency workloads, because the MELON I/O path selection scheme is adaptive.

We also evaluate the resource utilization of SwitchFS with different system implementation integrations over YCSB-B workload on RocksDB. As shown in Figure 16(c), the CPU utilization rises less than 15% after enabling the fine-grained lock and the MELON policy, which is a reasonable overhead for model inference and lock management. The memory overhead of SwitchFS is under 17%, where the main memory consumption is from the fine-grained lock metadata and the RL model parameters. As for PM device space utilization, SwitchFS consumes up to 47% more PM space than Assise because of the opportunistic replication mechanism, and the local cache allocation. The network bandwidth utilization of SwitchFS is 2.55× higher than Assise, which indicates that SwitchFS can utilize more remote PM devices across the cluster, leaving less wasted network bandwidth.

6 Discussion

In this section, we discuss how SwitchFS and MELON generalize beyond our current prototype, as well as the limitations and assumptions of the present design.

6.1 Generalization to Multi-Tier Environments

Although our implementation targets a two-tier setting with local PM and remote PM over a high speed RDMA fabric, the design of MELON and the dynamic I/O path selection mechanism naturally extend to various configurations.

First, SwitchFS can be generalized to multiple storage tiers with different latency and bandwidth characteristics (for example, CXL-attached memory and NVMe SSDs). In this setting, the action space of MELON would be expanded from *local versus remote PM* to *a set of per tier paths*, and the state representation would be augmented with tier-specific features such as static latency for each tier. The Q-learning formulation and latency based reward remain unchanged under these conditions. MELON would still aim at minimizing expected latency while implicitly aggregating bandwidth across all eligible tiers. As the model is trained online from observed I/O outcomes, it can adapt to the relative performance of new tiers without requiring redesign of the policy logic.

Second, SwitchFS can accommodate heterogeneous networks, including a mixture of RDMA links with different speeds (e.g., 200 Gbps, 100 Gbps, and 25 Gbps) or even a combination of RDMA and non-RDMA transports. We have shown that MELON optimizes the I/O performance under different RDMA rates in our evaluation, which suggests that it can learn to adapt to various network conditions. In our current prototype, network heterogeneity is captured indirectly through the observed per path latency that feeds into MELON's reward and state. In a more explicitly heterogeneous deployment, additional network level indicators (such as link type, and per path bandwidth estimates) could be incorporated into the state vector, allowing MELON to learn path selection policies that account for both storage and network asymmetries.

Finally, we note that SwitchFS does not require programmable switches or in-network computation. All data and metadata operations, including MELON and the model-driven replication logic, execute in end hosts. This design choice simplifies deployment in current HPC and cloud environments where the availability and programmability of the network fabric can vary widely.

6.2 Security and Threat Model

Our current design assumes a trusted data center environment in which servers and clients are mutually authenticated and the RDMA fabric is not adversarial. Under this assumption, the primary security concern is correctness and robustness of metadata and data operations, rather than protection against malicious nodes.

Introducing learning-based components, however, does enlarge the potential attack surface. For example, a compromised client could attempt to skew MELON's training data, biasing the learned policy toward suboptimal or resource wasting paths, or it could inject spurious, unavailable I/O path reports to manipulate the placement of replicas. Similarly, a malicious client could try to violate consistency by forging or replaying lock and lease requests.

Mitigating these risks requires standard security mechanisms that are orthogonal to our current prototype. Countermeasures include authenticating all control messages (including I/O metadata, training logs, and replication hints), enforcing access control at the FS-service for both data and metadata operations, and validating lock and lease requests against server-side invariants. Broadly, a formal security analysis of MELON's training and decision interfaces, using techniques from robust or adversarial machine learning, is also a viable direction for future work.

6.3 Interaction with POSIX, NFS, and Parallel File Systems

SwitchFS presents a POSIX-like interface to applications by intercepting standard file system calls at the client side and enforcing sub-file locks and leases as described in Section 3.4. The data consistency invariants we define, per client read-after-writes, single-writer multi-reader serialization on overlapping ranges, and ordered propagation of updates across replicas, are compatible with traditional semantics used in network file systems.

In principle, SwitchFS could be integrated with existing POSIX, NFS, or parallel file system deployments in several ways. One possible approach is to expose SwitchFS through a user-space NFS or parallel I/O gateway that translates NFS or MPI-IO requests into SwitchFS operations. In this case, MELON's decisions would be driven by the aggregated I/O patterns of the gateway. Exploring the detailed implications for failure handling, cache coherence, and metadata scalability in such hybrid designs is left for future work.

6.4 Future Work

There are several promising directions for future work building on SwitchFS and MELON.

First, MELON relies on online learning from production I/O traffic, which implies a warm-up period before the policy converges to a stable strategy for a new workload or hardware configuration. Although our experiments show that convergence is fast enough for the evaluated scenarios, a deeper theoretical and empirical study of convergence speed and stability across a broader class of workloads would strengthen the claims.

Second, the current implementation does not yet incorporate explicit security mechanisms for protecting MELON's training data or replication hints, nor does it integrate with existing enterprise authentication and authorization frameworks. These are important considerations for real-world deployment and warrant further design and evaluation.

Overall, we view SwitchFS and MELON as a step toward learning augmented, bandwidth aggregating distributed file systems. We hope that the abstractions and mechanisms introduced here will inform the design of future systems that operate over multiple storage hierarchies and heterogeneous networks while providing strong consistency and security guarantees.

7 Related Works

The emergence of PM technology has aroused many research works on redesigning the storage system software stack. There are many kernel-level file systems that take advantage of byte-addressable PM [5, 15, 55, 59]. Notably, Ziggurat [66], Orthus [58], and SPFS [56] are considering new design pattern on heterogeneous memory and storage tiers. On the other hand, the user-level PM file system is rising. Strata [33], CrossFS [44], KucoFS [4] and MadFS [67] are taking user-level operations for file system acceleration. Moreover, researchers have also explored providing safe metadata operations to the user-level [11, 40].

When it comes to co-designing PM storage systems with RDMA, there exist lots of outstanding research works. AsymNVM [39] is shared PM system design under the guidance of resource disaggregation. FileMR [61] redesigns the RDMA metadata organization for DFS convenience. Octopus [38], Orion [60] and Assise [2] are well-known RDMA-aware DFS designs. There are also researches concentrating on analyzing general technology choice and integration strategy of RDMA [1, 27]. At the scale of geo-distributed HPC data centers and scientific computing, SciSpace [28] proposes a collaboration workspace across distributed sites, and recent work on persistent memory object storage and indexing for scientific computing [29] explores how to organize and query PM-backed scientific data. These systems focus on data sharing, object storage, and index structures over PM and high-performance networks, whereas SwitchFS targets online RL-based I/O path selection and model-driven replication to aggregate local and remote PM bandwidth within a single cluster.

Some researchers in operating system design get inspired by the success of artificial intelligence (AI) and come up with new design paradigms. For example, some researchers focus on the data center scheduling problem and apply classification-based methods to heterogeneous data center servers [8, 9, 57]. Network optimization may also get benefit from artificial intelligence or machine learning [12, 36]. Specifically, some existing works utilize AI technologies in the storage system and reveal the predictability of the key-value store system [10, 32]. There is also some existing work on learning the time series model in the operating system [22].

8 Conclusion

We have designed SwitchFS, a novel DFS architecture that exploits both local and remote PM bandwidth in a cluster. The advances of SwitchFS come from four distinctive designs: the SwitchFS dynamic I/O management mechanism, the MELON learning-based I/O path selection policy, the model-driven opportunistic replication scheme, and the fine-grained lock/lease mechanisms. We have implemented and evaluated SwitchFS in a rack-scale system with real PM and RDMA devices. Experimental results show that SwitchFS outperforms existing DFSs on I/O intensive benchmarks and real-world applications.

Acknowledgments

We thank our shepherd David Kaeli and the anonymous reviewers for their valuable feedback and insightful suggestions.

References

[1] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *Proceedings of the 15th European Conference on Computer Systems*. 1–16.

[2] 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 *Proceedings of the 14th {USENIX} Symposium on Operating Systems Design and Implementation*. 1011–1027.

[3] Jens Axboe. 2021. Flexible I/O tester. Retrieved May 15, 2025 from https://github.com/axboe/fio

[4] Youmin Chen, Youyou Lu, Bohong Zhu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Jiwu Shu. 2021. Scalable persistent memory file system with kernel-userspace collaboration. In *Proceedings of the 19th {USENIX} Conference on File and Storage Technologies*. 81–95.

[5] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. 2009. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 133–146.

[6] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. 143–154.

[7] Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. 2012. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th International Conference on Autonomic Computing*. 191–200.

[8] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices* 48, 4 (2013), 77–88.

[9] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices* 49, 4 (2014), 127–144.

[10] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. 2020. ALEX: An updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 969–984.

[11] Mingkai Dong, Heng Bu, Jifei Yi, Benchao Dong, and Haibo Chen. 2019. Performance and protection in the ZoFS user-space NVM file system. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 478–493.

[12] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} vivace: Online-learning congestion control. In *Proceedings of the 15th {USENIX} Symposium on Networked Systems Design and Implementation*. 343–356.

[13] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: Evolution of development priorities in a key-value store serving large-scale applications. *ACM Transactions on Storage* 17, 4 (2021), 1–32.
[14] Zhuohui Duan, Haodi Lu, Haikun Liu, Xiaofei Liao, Hai Jin, Yu Zhang, and Song Wu. 2021. Hardware-supported remote persistence for distributed persistent memory. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.
[15] Subramanya R Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. 2014. System software for persistent memory. In *Proceedings of the 9th European Conference on Computer Systems*. 1–15.
[16] Facebook-RocksDB. 2021. A persistent key-value store for fast storage environment. Retrieved May 15, 2025 from https://rocksdb.org/
[17] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. 2020. A theoretical analysis of deep Q-learning. In *Proceedings of the Learning for Dynamics and Control*. PMLR, 486–489.
[18] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2022. Enabling practical cloud performance debugging with unsupervised learning. *ACM SIGOPS Operating Systems Review* 56, 1 (2022), 34–41.
[19] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. 29–43.
[20] Google-Tensorflow. 2021. TensorFlow. Retrieved May 15, 2025 from https://github.com/tensorflow/tensorflow/tree/ma%ster/tensorflow
[21] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S Gunawi. 2020. LinnOS: Predictability on unpredictable flash storage with a light neural network. In *Proceedings of the 14th {USENIX} Symposium on Operating Systems Design and Implementation*. 173–190.
[22] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning memory access patterns. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1919–1928.
[23] Intel. 2021. Achieve Greater Insight From Your Data with Intel Optane Persistent Memory. Retrieved May 15, 2025 from https://www.intel.com/content/www/us/en/products%/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-ser%ies-brief.html
[24] Nusrat S. Islam, Mohammad Wahidur Rahman, Jithin Jose, Raghunath Rajachandrasekar, Hao Wang, Hari Subramoni, Chet Murthy, and Dhabaleswar K. Panda. 2012. High performance RDMA-based design of HDFS over infiniband. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.
[25] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.
[26] Anuj Kalia, David Andersen, and Michael Kaminsky. 2020. Challenges and solutions for fast remote persistent memory access. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 105–119.
[27] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. Design guidelines for high performance {RDMA} systems. In *Proceedings of the 2016 {USENIX} Annual Technical Conference*. 437–450.
[28] Awais Khan, Taeuk Kim, Hyunki Byun, and Youngjae Kim. 2019. SciSpace: A scientific collaboration workspace for geo-distributed HPC data centers. *Future Generation Computer Systems* 101 (2019), 398–409.
[29] Awais Khan, Hyogi Sim, Sudharshan S. Vazhkudai, Jinsuk Ma, Myeong-Hoon Oh, and Youngjae Kim. 2020. Persistent memory object storage and indexing for scientific computing. In *Proceedings of the 2020 IEEE/ACM Workshop on Memory Centric High Performance Computing*. IEEE, 1–9.
[30] 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 *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 756–771.
[31] Wook-Hee Kim, R. Madhava Krishnan, Xinwei Fu, Sanidhya Kashyap, and Changwoo Min. 2021. PACTree: A high performance persistent range index using PAC guidelines. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 424–439.
[32] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. 489–504.
[33] 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*. 460–477.
[34] Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. 2021. lODA: A host/device Co-design for strong predictability contract on modern flash storage. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 263–279.
[35] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Robert Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. 2003. Parallel netCDF: A high-performance scientific I/O interface. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. IEEE, 39–39.

[36] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural packet classification. In *Proceedings of the ACM Special Interest Group on Data Communication*. 256–269.

[37] Xiaoyi Lu, Nusrat S Islam, Md Wasi-Ur-Rahman, Jithin Jose, Hari Subramoni, Hao Wang, and Dhabaleswar K Panda. 2013. High-performance design of hadoop RPC with RDMA over infiniband. In *Proceedings of the 2013 42nd International Conference on Parallel Processing*. IEEE, 641–650.

[38] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: An rdma-enabled distributed persistent memory file system. In *Proceedings of the 2017 {USENIX} Annual Technical Conference*. 773–785.

[39] Teng Ma, Mingxing Zhang, Kang Chen, Zhuo Song, Yongwei Wu, and Xuehai Qian. 2020. Asymnvm: An efficient framework for implementing persistent data structures on asymmetric nvm architecture. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 757–773.

[40] Nafiseh Moti, Frederic Schimmelpfennig, Reza Salkhordeh, David Klopp, Toni Cortes, Ulrich Rückert, and André Brinkmann. 2021. Simurgh: A fully decentralized and secure NVMM user space file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[41] Michael Owens. 2006. *The Definitive Guide to SQLite*. Springer.

[42] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. 2008. Amazon S3 for science grids: A viable solution?. In *Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing*. 55–64.

[43] Zhenlin Qi, Shengan Zheng, Yifeng Hui, Bowen Zhang, and Linpeng Huang. 2023. Conflux: Exploiting persistent memory and RDMA bandwidth via adaptive I/O mode selection. In *Proceedings of the 52nd International Conference on Parallel Processing*. 685–694.

[44] Yujie Ren, Changwoo Min, and Sudarsun Kannan. 2020. CrossFS: A cross-layered direct-access file system. In *Proceedings of the 14th {USENIX} Symposium on Operating Systems Design and Implementation*. 137–154.

[45] Subhash Sethumurugan, Jieming Yin, and John Sartori. 2021. Designing a cost-effective cache replacement policy using machine learning. In *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture*. IEEE, 291–303.

[46] P. Sherubha, S. P. Sasirekha, A. Dinesh Kumar Anguraj, J. Vakula Rani, Raju Anitha, S. Phani Praveen, and R. Hariharan Krishnan. 2023. An efficient unsupervised learning approach for detecting anomaly in cloud. *Comput. Syst. Sci. Eng.* 45, 1 (2023), 149–166.

[47] Gagandeep Singh, Rakesh Nadig, Jisung Park, Rahul Bera, Nastaran Hajinazar, David Novo, Juan Gómez-Luna, Sander Stuijk, Henk Corporaal, and Onur Mutlu. 2022. Sibyl: Adaptive and extensible data placement in hybrid storage systems using online reinforcement learning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 320–336.

[48] Warren Smith, Ian Foster, and Valerie Taylor. 1998. Predicting application run times using historical information. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 122–142.

[49] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

[50] Vasily Tarasov, Erez Zadok, and Spencer Shepler. 2016. Filebench: A flexible framework for file system benchmarking. *USENIX; login* 41, 1 (2016), 6–12.

[51] Tensorflow-Lite. 2021. TensorFlow Lite. Retrieved May 15, 2025 from https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite

[52] Joannes Vermorel and Mehryar Mohri. 2005. Multi-armed bandit algorithms and empirical evaluation. In *Proceedings of the European Conference on Machine Learning*. Springer, 437–448.

[53] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[54] 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 *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. 307–320.

[55] Matthew Wilcox. 2014. DAX: Page Cache Bypass for Filesystems on Memory storage. Retrieved May 15, 2025 from https://lwn.net/Articles/618064/

[56] Hobin Woo, Daegyu Han, Seungjoon Ha, Sam H. Noh, and Beomseok Nam. 2023. On stacking a persistent memory file system on legacy file systems. In *Proceedings of the 21st USENIX Conference on File and Storage Technologies*. 281–296.

[57] Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel, Wangchao Le, Shi Qiao, and Sriram Rao. 2018. Towards a learning optimizer for shared clouds. *Proceedings of the VLDB Endowment* 12, 3 (2018), 210–222.

[58] Kan Wu, Zhihan Guo, Guanzhou Hu, Kaiwei Tu, Ramnatthan Alagappan, Rathijit Sen, Kwanghyun Park, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2021. The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus. In *Proceedings of the 19th {USENIX} Conference on File and Storage Technologies*. 307–323.

[59] Jian Xu and Steven Swanson. 2016. {NOVA}: A log-structured file system for hybrid volatile/non-volatile main memories. In *Proceedings of the 14th {USENIX} Conference on File and Storage Technologies*. 323–338.

[60] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2019. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks. In *Proceedings of the 17th {USENIX} Conference on File and Storage Technologies*. 221–234.
[61] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2020. FileMR: Rethinking {RDMA} Networking for Scalable Persistent Memory. In *Proceedings of the 17th {USENIX} Symposium on Networked Systems Design and Implementation*. 111–125.
[62] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. 2020. An empirical guide to the behavior and use of scalable persistent memory. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies*. 169–182.
[63] Yongen Yu, Douglas H. Rudd, Zhiling Lan, Nickolay Y. Gnedin, Andrey Kravtsov, and Jingjin Wu. 2012. Improving parallel IO performance of cell-based AMR cosmology applications. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 933–944.
[64] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. 2020. RLScheduler: An automated HPC batch job scheduler using reinforcement learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.
[65] Yiying Zhang and Yutong Huang. 2019. "Learned" Operating Systems. *ACM SIGOPS Operating Systems Review* 53, 1 (2019), 40–45.
[66] Shengan Zheng, Morteza Hoseinzadeh, and Steven Swanson. 2019. Ziggurat: A tiered file system for non-volatile main memories and disks. In *Proceedings of the 17th {USENIX} Conference on File and Storage Technologies*. 207–219.
[67] Shawn Zhong, Chenhao Ye, Guanzhou Hu, Suyan Qu, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, and Michael Swift. 2023. {MadFS}:{per-file} virtualization for userspace persistent memory filesystems. In *Proceedings of the 21st USENIX Conference on File and Storage Technologies*. 265–280.
[68] Bohong Zhu, Youmin Chen, Qing Wang, Youyou Lu, and Jiwu Shu. 2021. Octopus+: An RDMA-enabled distributed persistent memory file system. *ACM Transactions on Storage* 17, 3 (2021), 1–25.

Received 30 October 2025; revised 8 April 2026; accepted 20 May 2026