Conflux: Exploiting Persistent Memory and RDMA Bandwidth via Adaptive I/O Mode Selection

Zhenlin Qi
qizhenlin@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Shengan Zheng*
shengan@sjtu.edu.cn
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
Shanghai, China

Yifeng Hui
sinemora@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Bowen Zhang
bowenzhang@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Linpeng Huang*
lphuang@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

ABSTRACT

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 system (DFS) designs. Existing DFSs often consider remote storage a performance constraint, assuming it delivers lower bandwidth and higher latency than local storage devices. However, the advances in RDMA technology provide an opportunity to bridge the performance gap between local and remote access, enabling DFSs to leverage both local and remote PM bandwidth and achieve higher overall throughput.

We propose Conflux, a new DFS architecture that leverages the aggregated bandwidth of PM and RDMA networks. Conflux dynamically steers I/O requests to local and remote PM to fully utilize PM and RDMA bandwidth under heavy workloads. To adaptively decide the I/O run-time path, we propose SEED, a learning-based policy engine predicting Conflux I/O latency and making decisions in a real-time system. Furthermore, Conflux adopts a fine-grained concurrency control approach to improve its scalability. Experimental results show that Conflux achieves up to 4.7× throughput compared to existing DFSs on multi-threaded workloads.

CCS CONCEPTS

• **Information systems** → **Distributed storage**; **Storage class memory**; • **Social and professional topics** → **File systems management**; • **Computing methodologies** → *Neural networks.*

KEYWORDS

persistent memory, RDMA, machine learning

ACM Reference Format:
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 *52nd International Conference on Parallel Processing (ICPP 2023), August 07–10, 2023, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605573.3605574

1 INTRODUCTION

Emerging byte-addressable persistent memory (PM) provides orders of magnitude higher performance than conventional storage devices. Meanwhile, data centers nowadays 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 dramatically improves 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 storage systems. Specifically, several distributed file system (DFS) designs benefit from the high performance of PM and RDMA technologies. However, existing DFS designs fall short in fully exploiting the performance of both hardware simultaneously.

In conventional DFSs, file I/O performance is hampered by network bandwidth constraints and the poor performance of disk devices [16, 40]. There are two kinds of optimization in the existing literature: The first is to adopt new network hardware and redesign the software to exploit the network performance. Systems designed within this paradigm always reduce the communication latency and increase the remote data access throughput [22, 32, 33]. The second is to adopt new storage technology and redesign the storage system software. Systems designed within this paradigm follow the conventional wisdom: *local access is better than remote* and build DFS with client local cache architecture to improve data access performance [2, 26, 46].

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 the latest PM and RDMA devices. In the traditional distributed storage architecture, the bandwidth of accessing the local DRAM cache is orders of magnitude higher than that of accessing remote disks, resulting in an imbalance between local and remote I/O performance. However, current

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 I/O performance. Intensive usage of either device will tip the performance balance, leaving the other under-utilized.

The key to maintaining the performance balance 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 designer must properly address: First, it is challenging for a file system to steer I/O tasks to local and remote modes dynamically, instead of the conventional deterministic I/O path. Second, it is challenging to assign proper I/O (remote/local) modes at run-time. Different applications tend to have different file access patterns, and improper I/O mode scheduling will deteriorate the overall performance. Third, it is challenging to fully take advantage of the high bandwidth hardware with conventional concurrency control approaches, which generally enforce file-level read-write locks.

We propose Conflux, a new DFS architecture that fully exploits PM and RDMA bandwidth. It tackles the above-mentioned challenges and provides aggregated bandwidth to I/O-intensive applications. Conflux comes up with a dynamic I/O management mechanism, which can serve user I/O requests with different internal data transmission modes. As a result, local and remote PM devices can be utilized simultaneously at runtime. To generate adaptive I/O mode selection choices for various applications, we propose SEED, a learning-based I/O mode selection policy. It adopts a lightweight neural network model which makes I/O mode decisions according to estimation on current I/O pressure. Meanwhile, the design of SEED enables online training paradigm which reduces computational overhead and facilitates model evolution. To eliminate the concurrency control bottleneck and take advantage of Conflux in scalable working set, we design a fine-grained concurrency control mechanism facilitating highly concurrent I/O requests.

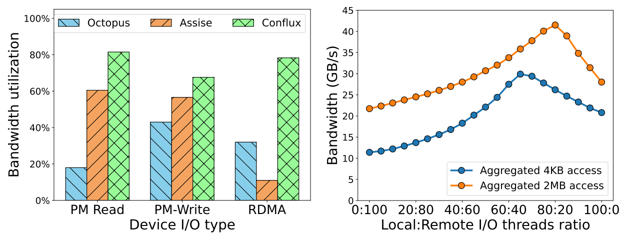
In summary, we make the following contributions:

- We present the design of Conflux, a DFS architecture that exploits both local and remote PM bandwidth by dynamically managing the two I/O modes at run-time. We further design a lightweight fine-grained concurrency control mechanism that provides higher scalability.
- We present SEED, the first learning-based I/O mode selection policy in DFS. SEED learns from the history of I/O information and helps Conflux to make adaptive runtime choices between local and remote I/O path.
- We implement Conflux and quantify its performance on various benchmarks. Experimental results show that Conflux fully exploits the local and remote PM performance, significantly outperforming existing DFS designs.

2 BACKGROUND AND MOTIVATION

2.1 Progress in PM and RDMA technologies

The development of computer hardware technologies is rapid in recent years. Intel has introduced the Optane PM series of non-volatile memory, which offers high bandwidth for load/store operations [21]. Meanwhile, Mellanox has launched the ConnectX-7 device for RDMA technology, which doubles the network bandwidth compared to its previous generation.



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

Figure 1: Space for hardware utilization improvement.

A large body of research has focused on leveraging high performance PM and RDMA hardware and redesigning the file system. The straight-forward design paradigm of utilizing PM and RDMA is to store file data in PM and accelerate the data transmission via RDMA. Octopus [33] is a recent research work 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 [46] 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.

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 utilization problems using servers (see details in Section 5.1) that are armed with Mellanox ConnectX-6 adapter, which provides 200 Gbps network bandwidth.

We conduct two experiments to show the space for performance improvement. 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 1a. We discover that they do not utilize hardware bandwidth sufficiently (as high as 84% of bandwidth wasted). Our proposed system, named Conflux, takes more advantages from the high bandwidth devices. It is because Conflux utilizes both local and remote PM to serve the I/O requests.

The second experiment shows examples of the bandwidth aggregation phenomenon. 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 bandwidth results in Figure 1b. We draw two conclusions from the results: (i) Simultaneous usage of local and remote PM can provide much higher aggregated bandwidth for applications. If a system relies on local PM or remote PM access, it can only achieve 70% or 40% of the highest possible bandwidth. (ii) It is not a trivial problem to fine-tune the I/O system where we seek to utilize aggregated bandwidth. With different I/O sizes and concurrency levels, fixed ratio cannot exploit the highest bandwidth. To make it more complicated, the real working environment may contain mixed sizes of I/O and change its concurrency by time. We need to devise an adaptive policy to deal with the complicated scenarios.

2.3 Concepts from machine learning

The success of machine learning (ML) has motivated OS design researchers to automate the system tuning process. ML research field encompasses three main paradigms: supervised learning, unsupervised learning, and reinforcement learning. These learning-based techniques have been widely applied to OS design researches with promising results [6, 18, 38].

Previous works have demonstrated the predictability of operating system I/O performance using supervised learning [18, 29]. However, these works rely on intensive pre-training of neural network models and extensive dataset collection procedures, which are time-consuming and lack adaptiveness. Reinforcement learning (RL) is a paradigm that can train a model to adapt to a dynamic environment and apply an optimal policy. Several research works have shown that RL can achieve excellent outcomes in OS designs [37, 49, 50]. Here we highlight two essential conceptions in RL: exploitation and exploration. The balance of exploitation and exploration [23] is always an active theme in policy-learning research. Exploitation represents choosing what is known and getting a reward close to the model's expectation. The exploitation ability helps a policy to make the known optimal choice. On the other hand, exploration represents choosing what is unsure and getting more knowledge about the environment. The exploration ability is vital for a learning model to evolve in a dynamic environment.

3 CONFLUX DESIGN

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

- **Exploiting both PM and RDMA bandwidth.** One of the main goals of Conflux is to simultaneously expose client local PM bandwidth and RDMA NIC bandwidth to applications in a transparent manner. Conflux introduces a novel dynamic I/O management mechanism, which enables local and remote modes at the run-time. It manages PM space and RDMA connections to activate different I/O modes for requests.

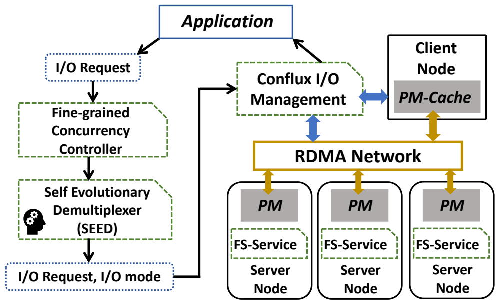


Figure 2: Conflux architecture overview.

- **Adaptive I/O mode selection.** To make proper run-time I/O mode decisions, Conflux introduces a novel learning-based I/O mode selection policy named *SElf Evolutionary Demultiplexer* (SEED). SEED learns from the performance statistics of Conflux and makes the proper I/O mode decision. It is adaptive concerning the dynamic local storage pressure and network device pressure.
- **Fine-grained I/O concurrency control.** The centralized coarse-grained concurrency control mechanism is a bottleneck for concurrent data I/O. Conflux comes up with a fine-grained tree-structured data lock at the server side and a lease mechanism at the client side. They work together to provide higher parallelism under concurrent I/O requests.

The software components of Conflux are shown in Figure 2. Applications are running at the client side. Conflux serves all the application issued I/O requests and pushes them through three main software components at client. The fine-grained concurrency controller is first activated to grant access admission. Then the I/O request passes through SEED, which generates an I/O mode choice according to the request and run-time I/O environment estimation. Finally the request is sent to Conflux 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 and support concurrent file operations.

3.1 Conflux I/O management mechanism

On the client-side, Conflux employs a unique I/O management mechanism that exploits both PM and RDMA bandwidth for user applications. Conflux I/O manager receives I/O requests from the application at run-time and dynamically selects either local or remote mode to serve them. On the server-side, Conflux 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 with mode advice from the control plane, it forwards them to the Conflux I/O manager. The Conflux 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 provide both local

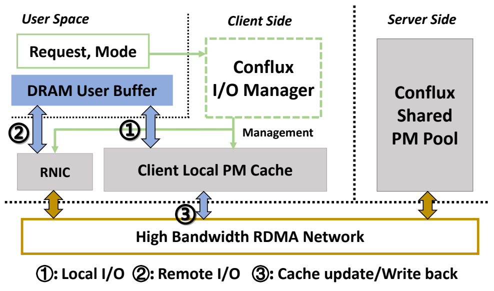


Figure 3: Conflux dynamic I/O management.

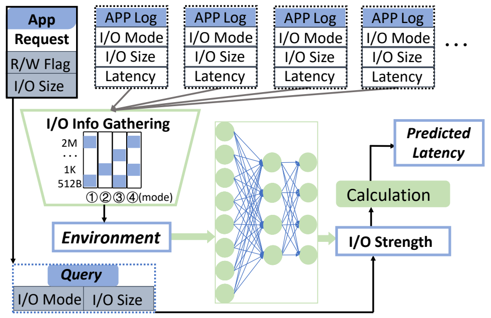


Figure 4: The working flow of SEED I/O latency prediction.

and remote I/O mode to serve the incoming I/O tasks. Moreover, it manages the cache update and write back tasks to maintain the cache consistency. Conflux I/O manager initiates 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 Conflux client cache includes both data and metadata. Metadata cache maintains file status (e.g., file mode, size, access time) information and the address translation table. They reduce the cross-node communication overhead and facilitates 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, Conflux 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 SEED: adaptive I/O mode selection

We devise a novel I/O mode selection policy named *SElf Evolutionary Demultiplexer* (SEED). As an overview, SEED uses a machine learning model to predict the latency of local and remote I/O, and generates I/O mode decision according to latency values. Lower latency indicates better exploitation of the hardware bandwidth, thus SEED helps fully take advantage of the dynamic I/O mechanism in Conflux. As its name shows, SEED adopts online learning paradigm which saves the system from complicated training data processing procedure and facilitates the adaptive I/O mode selection.

The reason we choose I/O latency as the prediction target, instead of straight-forward local/remote mode choice, is two-fold: (i) While Conflux seeks to exploit more hardware bandwidth, it is non-trivial to measure the correlation between I/O mode decisions and the bandwidth benefits. Latency is a proper measurement indicating bandwidth utilization for each I/O task with known I/O size. (ii) For a machine learning model, the training data should be ground-truth. It is almost impossible to retrieve ground-truth *better I/O mode* at run-time, which can only be generated by heavy offline pre-processing. Latency recording incurs low CPU overhead, even performed for each I/O task. Meanwhile, its ground-truth value is confirmed right after data transmission, thus facilitates the online model evolution process.

We divide SEED into four subsystems as follow.

3.2.1 ***I/O request monitor.*** SEED tracks the I/O request execution information within Conflux. The tracked information has two main uses: (1) for working threads to estimate the dynamic runtime I/O pressure, and (2) for the machine learning model to collect the training data on I/O metadata. SEED maintains a per-thread in-DRAM log area for every Conflux user to record I/O request metadata, which consists of I/O size, I/O type, and ground-truth latency. The I/O request log is a circular buffer with a producer and a consumer. Conflux working thread is the I/O record producer, and the SEED learning model is the consumer. The I/O request log of each Conflux thread is shared among all the other threads, such that SEED policy can generate comprehensive estimation of the I/O pressure on local and remote devices in the whole system.

3.2.2 ***Latency prediction model.*** SEED uses a neural network model to finish the latency prediction task. We devise vector generation and post-calculation functions to compose feasible inputs and outputs data structures for the neural network. Figure 4 shows the input generation process, neural network prediction, and output manner of the model.

In detail, the inputs of the neural network include two parts: an *environment* and a *query*. The *environment* represents the collection of recent I/O information. We design an I/O info gathering procedure to generate its contents. It reads the four most recent I/O requests from each Conflux thread, and groups all the I/O log items according to their I/O size and I/O type. Different I/O types have different influences over the local and remote devices. Different I/O sizes are separated to emphasize their different contributions to I/O pressure. SEED maintains a counter for each group and combine all the counter values as a result of *environment*. By I/O info gathering procedure, we represent the I/O information from various working threads as a fixed-size real-value vector, leading to efficient neural network model processing.

Given an *environment* vector, SEED pushes it through the neural network. The last layer of the model outputs a set of real numbers falling in the $[0, 1]$ interval, interpreted as the predicted I/O *strength*. The *strength* represents the ratio between the devices' I/O latency lower-bound and the predicted runtime I/O latency. The lower-bound depends on PM and RDMA NIC hardware, which can be generated by a stand-alone single-threaded latency test and manually configured in our model. Finally the second part of inputs named *query*, including the incoming request's mode and size, is pushed into the model. The model selects a predicted I/O *strength* by the *query* I/O type and I/O size. It calculates the required latency value, dividing the latency lower-bound by the I/O *strength*.

The neural network in SEED is a lightweight model with two fully connected hidden layers. It only contains less than 3K parameters such that the whole model can be stored in a file (or loaded into memory) of about 13 KB size. It incurs low CPU and memory overhead while meets the accuracy requirement for real-time prediction. Targeting at runtime latency prediction, we view the *environment* to *latency* relationship as a functional mapping. Since it is impractical to maintain a lookup table for the high-dimensional inputs generated by multi-thread workload, SEED approximates this high-dimensional function using the neural network. It has been proved that a multi-layer neural network is a universal approximator [20] that can model functional relationships effectively. Moreover, we present the model training loss (see Section 5.4) to support this claim in our case.

3.2.3 ***Semi-online learning scheme***. Due to the fewer arithmetic operations included, the neural network inference time costs are much lower than the model training costs. SEED uses a semi-online learning strategy that can train the neural network efficiently. By separating the inference tasks and the training procedure, it achieves a balance between low overhead and high precision.

SEED keeps a background process to collect the inputs-outputs data pairs generated by client threads. It gathers them into a training data pool and trains a new model continuously. The *train after collection* manner allows batched data feeding in model training, which is more efficient than a single data pass in a neural network model. Meanwhile, all the client processes have a read-only copy of the prediction model and execute the inference procedure at run-time. The background process maintains a history of training error values as a neural network model quality measurement. When it finds a significant error decrease at the end of a training epoch, it views the new model as better than the old model. Under that condition, it persists the new model in a new file and informs all Conflux client processes to update their model.

We further control the overhead for extremely small I/O requests by adding a short circuit path and allowing immediate I/O latency prediction. SEED maintains a threshold for admission to the neural network. For I/O size under that threshold, we store the previous latency value as the prediction result. SEED adjusts the threshold dynamically according to the neural network elapsed CPU time ratio, thereby constraining the costs.

3.2.4 ***Latency-based decision policy***. Having the I/O latency predictor in hand, SEED makes a latency-based I/O mode decision for each request in Conflux. It examines the relationship between the predicted local and remote I/O latency and advises on the mode choice. We define a value *Latency Ratio* as:

$$Latency\ Ratio = \frac{Lat_{PM}}{Lat_{PM} + Lat_{RDMA}}, \tag{1}$$

where Lat_{PM} and Lat_{RDMA} represent the predicted I/O latency for an incoming I/O request. Denote the *Latency Ratio* as variable r

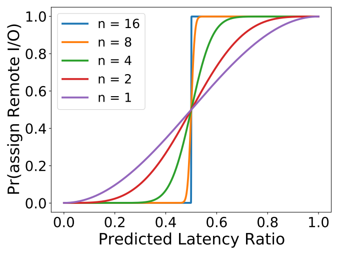


Figure 5: Decision curves with various certainty levels.

(where $r \in [0,\ 1]$), and the I/O mode choice v as a function of r that takes binary value, i.e., $v(r) \in \{0,\ 1\}$. Specifically, $v = 0$ indicates a local I/O mode and $v = 1$ indicates a remote (local bypass) I/O.

In SEED we design a probabilistic function for v. Let $u(x)$ be a shifted sine function mapping $[0, 1]$ to $[0, 1]$, i.e.,

$$u(x) = \frac{1}{2} sin(\pi x - \frac{\pi}{2}) + \frac{1}{2}, \tag{2}$$

and the value of v is decided by:

$$Pr\ (v(r) = 1) = u^n(r). \tag{3}$$

Here $u^n()$ denotes an n times composition of $u()$ itself. Different n values will generate different v-decision curves. To show the influence of n values on decision curves, we change the n value from 1 to 16 and draw Figure 5 including different curves. As a bigger n value generates a more definitive decision curve, we call n the *certainty level*. Heuristically, SEED configures 3 as the default *certainty level*. The advantage of such configuration will be shown by experimental results in Section 5.4.

There are three advantages of deciding the I/O mode according to the probabilistic policy defined by Equation 3. The first advantage is its unbiased behavior. As we explained in Section 2, both network devices and local storage devices should be utilized. Therefore, a good policy should make correct decision no matter higher I/O pressure incurs in local or remote devices. The second advantage is the balance between exploitation and exploration. On one hand, we need exploitation ability as runtime I/O mode decisions are based on SEED's advice. The decision according to Equation 3 shows the exploitation ability. With $n \rightarrow \infty$, $v(r) \rightarrow V(r)$, where $V(r)$ is a policy defined as:

$$V(r) = \begin{cases} 0 & , if\, r \leq 0.5 \\ 1 & , if\, r > 0.5 \end{cases} \tag{4}$$

that always chooses the predicted faster I/O mode. On the other hand, the exploration ability enhances the diversity of I/O mode selection and facilitates the neural network model evolution. Our decision-making policy exhibits this ability because it allows for the possibility of choosing the slower I/O mode predicted. The third benefit of this policy is its computational efficiency. The $sin()$ function is highly optimized in standard math libraries. Therefore, the decision-making process consumes a few CPU cycles, which is insignificant compared to other software overheads.

3.3 Fine-grained concurrency control

We now introduce the lightweight fine-grained concurrency control mechanism in Conflux. At the server-side, Conflux introduces a

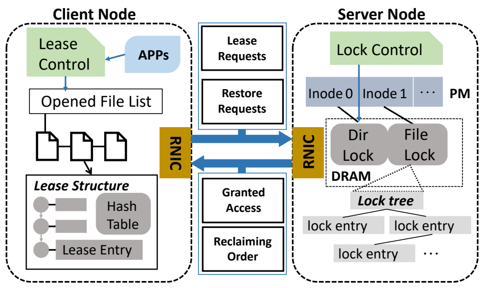


Figure 6: The fine-grained lock and lease mechanisms.

fine-grained sub-file level data lock which improves shared file I/O scalability. At the client-side, Conflux designs a lease control mechanism which reduces network communication.

As shown in Figure 6, the lock and lease mechanisms cooperate in providing a fast concurrent data locking service to applications. The fine-grained data lock is applied in Conflux servers. Every inode is related to a lock structure. For every file, Conflux 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. Conflux 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 [15, 30, 36, 48]. That indicates the fine-grained lock will improve scalability under I/O-heavy workloads with severe contention.

Furthermore, a lease controller resides in each Conflux 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 in Conflux.

4 IMPLEMENTATION

We implement Conflux as user-level libraries written in C. LibServer (6.5K LOC) is for server utilities. It includes the implementation of FS-service and server-side lock control functions. LibClient (6.6K LOC) is for client utilities. It includes the implementation of the lease control module, Conflux I/O manager, and SEED.

LibServer is the software component running at the Conflux 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 [35]. RDMA connection management, network message send/receive, and data read/write are implemented using `verbs` API provided by the RNIC driver.

LibClient is also implemented as a user-level module. It intercepts all POSIX system calls related to file system operation and invokes Conflux 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 [12]. 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 [24]. However, Conflux makes some of the DRAM buffer hot regions for cross-node messaging. Thus we keep the DDIO option on in our implementation. We implement a Python module, with the help of Tensorflow [17] package, for the neural network construction and training.

5 EVALUATION

In this section, we evaluate Conflux and compare it with existing DFS designs. We describe the experimental setup and evaluate Conflux with microbenchmarks and macrobenchmarks. Moreover, we show the effectiveness of SEED I/O mode selection policy and performance breakdown for different Conflux components.

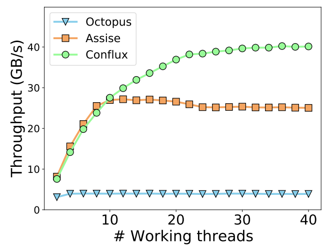
5.1 Experimental setup

In our testbed, nodes are armed with dual-socket Intel Xeon Gold 6348 CPU, where a single socket has 28 physical cores. For memory, each node has eight 16GiB DDR4 DRAM and eight 128 GB Optane persistent memory. For the network, each node has a Mellanox ConnectX-6 dual-port 200 Gbps HCA.

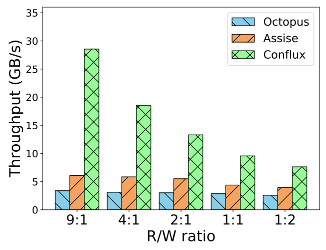
We compare Conflux with two DFSs designed for PM and RDMA, named Octopus [33, 53], and Assise [2], on microbenchmarks. As we use filebench [39] for macrobenchmark and Octopus does not fit in this benchmark, we replace it with NFS over RDMA. Note that the MLNX OFED driver supports NFS over RDMA, and we build EXT4-DAX [41] file system for NFS server-side exporting. For cluster organization, we use four nodes for server and client in Conflux by default. All benchmarks are running in clients, and the results are collected at steady-state and averaged between different client nodes. For Assise, we configure the hot replica number as two. In evaluation, we limit CPU usage, PM allocation, and RDMA NIC port within a single NUMA node for stable performance.

5.2 Microbenchmark results and analysis

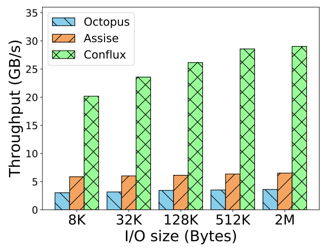
First, we conduct multi-threaded bandwidth tests using Fio. Figure 7a shows the results of concurrent read workload, where each thread allocates a private file and reads in 256 KB I/O size. Octopus cannot scale as thread number exceed four, and its bandwidth is lower than Assise and Conflux. Assise reaches the maximum bandwidth of local PM but cannot scale up beyond ten threads. Conflux outperforms Assise by a large margin (up to 1.6×). Because



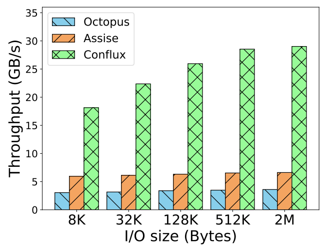
(a) Read-only workload.



(b) Read/write mixed workload.

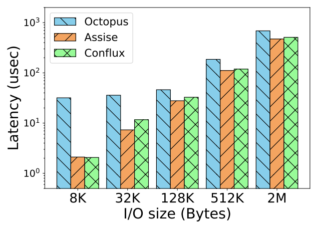


(c) Sequential access.

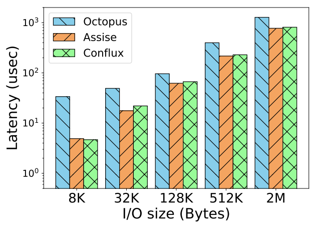


(d) Random access.

Figure 7: Bandwidth test results.



(a) Read latency.



(b) Write latency.

Figure 8: Single-thread I/O latency with various I/O sizes, vertical axis in log-scale.

Conflux can scale up when local PM bandwidth is saturated, where it uses the bandwidth of RDMA reading from remote servers. Figure 7b shows the results of various mixed read/write workloads. A read/write mixed workload always leads to performance degradation in a PM-aware system due to read-write contention. Conflux finds the latency increasing situation and relief it by steering them into different I/O modes. We also evaluate different systems using various I/O sizes and random I/O patterns and show results in figure 7c, 7d. Conflux outperforms the other two systems under various workloads.

Second, we conduct single-threaded I/O latency tests with various I/O sizes. We use Fio [3] to create a 1 GB file and perform sequential r/w on it. Figure 8 shows the results. Octopus encounters a higher latency than Assise and Conflux in general. For 8 KB I/O, Conflux has a similar latency as Assise. For larger I/O size, Conflux has a slightly larger latency than Assise. There are two reasons behind the single-threaded latency overhead: (1) Conflux has a neural network inference overhead for each I/O, except for those with I/O size under admission threshold. In our implementation, this inference process incurs 1.2 us CPU time overhead to the I/O task. (2) SEED adopts a probabilistic I/O mode selection, which leads to a small portion of choosing-the-slower decision. The combinative effect of the two factors is visible in the latency results of 32 KB I/O size. While in 2 MB I/O, the first factor is negligible, and the second factor dominates in the latency-increasing phenomenon.

5.3 Macrobenchmark results and analysis

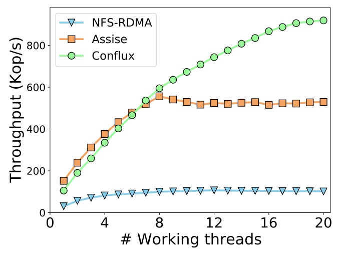
We take *varmail*, *fileserver* and *webserver* from Filebench [39] as macrobenchmarks. We make varmail and fileserver to operate on a working set with 10 K files, where the average initial file sizes are 16 KB and 128 KB, respectively. The overall read-to-write ratios are 1:1 and 1:2 in varmail and fileserver. As Assise's current implementation does not support the multi-thread execution of filebench, we use multiple processes instead. Figure 9 shows the throughput on these two benchmarks. Data I/O operations are expensive for NFS over RDMA, as it needs intensive data transmission between server-side PM storage and the client-side buffer cache. Assise stops scaling at eight threads for varmail and four threads for fileserver. The main reason is the centralized kernel module for log digestion in Assise. Conflux outperforms the other two DFSs by 9× and 1.7× on varmail, and by 2.8× and 3.2× on fileserver. Because we adopt the fine-grained concurrency control mechanisms at user-space, and utilize both local and remote PM for I/O tasks.

To evaluate the system with larger working set, we use Webserver with 40 K files and 1 MB average file size. We also specify a flag so that the generated file sizes follow a Gamma distribution. Consequently, this benchmark includes I/O tasks with various I/O sizes. Figure 9c 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×.

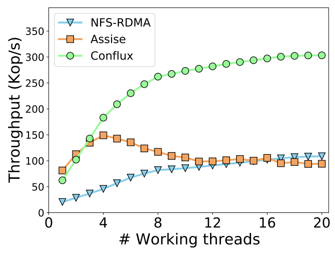
5.4 Effectiveness of SEED policy

5.4.1 Adaptiveness of SEED. We conduct a series of experiments to show that SEED can optimize the overall I/O performance adaptively, under various I/O size and different concurrency levels.

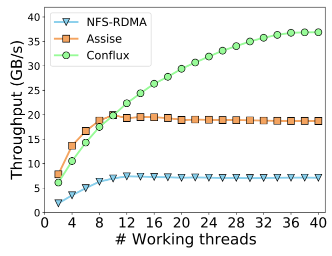
For comparison, we configure a group of I/O policies in Conflux, i.e., using fixed local to remote I/O ratios ranging from 0:100 to 100:0. We use webserver benchmark, with 1 MB average file size following Gamma distribution, to generate I/O tasks with various I/O sizes. The throughput results under different concurrency levels (thread numbers) are shown in figure 10. Generally, Conflux outperforms all the fixed-ratio I/O mode 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. Conflux 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. Conflux, however, performs better than all the fixed-ratio policies. Because through modeling the I/O latency, Conflux applies different policies



(a) Varmail results.



(b) Fileserver results.



(c) Webserver with increased filesize.

Figure 9: Performance on filebench workloads.

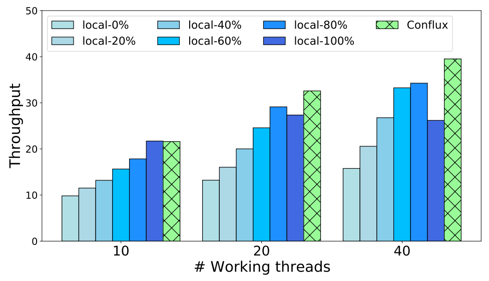


Figure 10: Comparison with fixed-ratio I/O policies.

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 our certainty level. We have present our *certainty level* design in section 3.2.4. In practice, it has two advantages: (1) exploiting high aggregated bandwidth. (2) reducing the model training period. We construct a multi-threaded workload reading a shared file at 256 KB I/O size and find the oracle policy exploiting the highest bandwidth. It is equivalent to finding the I/O mode selection strategy that leads to a *Latency Ratio* = 0.5 (defined in Equation 1). We depict the relationship between *Latency Ratio* and remote I/O ratio in figure 11a. It indicates that the optimal remote I/O ratio is 0.34. Denoting the PM bandwidth as BW_{PM}, we calculate the best aggregated bandwidth as $(1+\frac{0.34}{1-0.34})\times BW_{PM} \approx 1.5\cdot BW_{PM}$. Under the same workload, Figure 11b shows Conflux's performance with different configurations. The PM bandwidth is reached by setting I/O mode to local-only. Conflux with SEED helps to exploit more bandwidth, while the moderate *certainty level* choice $n = 3$ acts better than the larger ($n = 6$) and smaller ($n = 1$) *certainty level*. It reaches the peak performance (1.5× local access bandwidth) at its steady status and has a shorter fluctuating period.

The unstable performance of SEED at the beginning of benchmark execution is typical. The reason is that the latency prediction model needs to collect enough run-time samples to learn a stable model. Moreover, the neural network evolution velocity is subject to hardware performance, and our testbed has only CPUs to run the training task. Figure 11c visualize the model training loss history during the benchmark execution. We find that lower loss value is related to higher throughput performance. However, the training loss is not always decreasing. The reason is that temporary skewed I/O mode selection in Conflux leads to temporary skewed training dataset. Similarly, the training loss history show the advantages of a moderate *certainty level* choice $n = 3$. Under that condition, it converges to a lower value with a shorter fluctuating period.

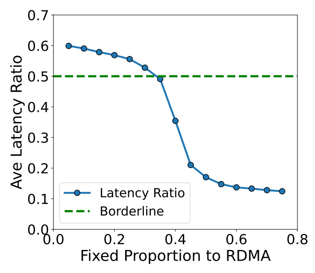
5.5 Performance breakdown and analysis

To figure out how the different design components affect the performance of Conflux, we conduct experiments to break down the performance results. We use Assise as the baseline, and activate the implementation of different design parts of Conflux to compare them. The '[+] Fine-grained Lock' system adopts the lock and lease mechanisms described in Section 3.3, but does not include local-bypass I/O policy. The '[+] Adaptive I/O mode' system is identical with our description of Conflux, which can dynamically choose the I/O mode for each request.

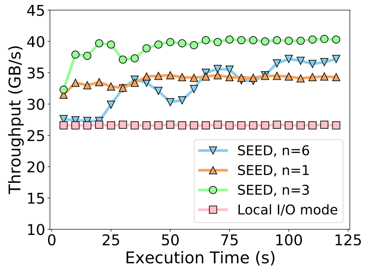
Figure 12 shows the performance of different system fragments under bandwidth tests and webserver workload. The workloads' configurations are identical to those in Section 5.2 and Section 5.3. We select four different working thread numbers to observe the performance breakdown with various concurrency. We find that both the fine-grained lock design and the adaptive I/O mode selection policy benefit a lot to Conflux. For workloads above 10-thread, the adaptive I/O mode selection, which is enabled by our SEED policy, improves the throughput by a large margin. Because the high concurrency level makes space for I/O bandwidth aggregation, and SEED helps to utilize both local and remote PM devices. In some case, e.g., the 4-thread webserver workload, Conflux achieves similar throughput as the baseline system. The low concurrency workload can not benefit a lot from our design inherently. To conclude, Conflux improves the I/O performance for high concurrency workloads while sacrifices little for low concurrency workloads, because the SEED I/O mode selection scheme is adaptive.

5.6 Real-world application performance

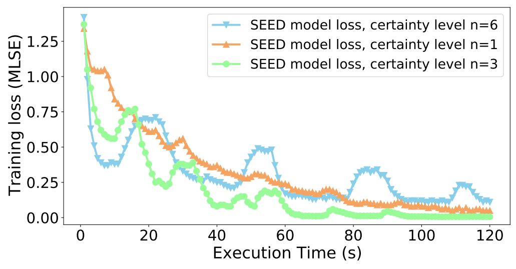
RocksDB [15] is a well-known persistent database based on LSM-tree. It is designed as a library component embedded in higher-level applications and optimized for large-scale distributed utilities [11]. We use the DBbench [14] benchmark, which can assign the database working directory, to compare the RocksDB performance built on different file systems. We run a workload that executes sequential write and 32-threads random read, with 10 M key-value pairs, key size set as 128 B, and value size set as 4 KB. NFS over RDMA achieves a throughput of 8406 MBps, while Conflux outperforms it by 1.32×. It is worth noting that the throughput has not reached the local PM



(a) Latency Ratio v.s. Remote I/O.

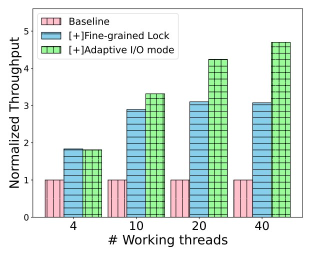


(b) Throughput, various certainty levels.

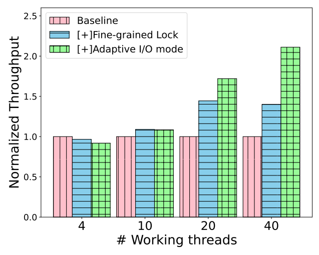


(c) Training loss, various certainty levels.

Figure 11: Advantages of the *certainty level* configuration.



(a) Bandwidth tests.



(b) Webserver.

Figure 12: Performance breakdown results.

bandwidth nor the RDMA bandwidth. The improvement here in Conflux mainly relies on the local PM cache usage and concurrency control mechanism.

6 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, 13, 41, 45]. Notably, Ziggurat [51], Orthus [44], and SPFS [42] are considering new design pattern on heterogeneous memory and storage tiers. On the other hand, the user-level PM file system is rising. Strata [28], CrossFS [36], KucoFS [4] and MadFS [52] are taking user-level operations for file system acceleration. Moreover, researchers have also explored providing safe metadata operations to the user-level [9, 35].

When it comes to co-designing PM storage systems with RDMA, there exist lots of outstanding research works. AsymNVM [34] is shared PM system design under the guidance of resource disaggregation. FileMR [47] redesigns the RDMA metadata organization for DFS convenience. Octopus [33], Orion [46] 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, 25].

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 [6], [7], [43]. Network optimization may also get benefit from artificial intelligence or machine learning [10], [31]. Specifically, some existing works utilize AI technologies in the storage system and reveal the predictability of the key-value store system [8, 27]. There is also some existing work on learning the time series model in the operating system [19].

7 CONCLUSION

We have designed Conflux, a novel DFS architecture that exploits both local and remote PM bandwidth. The advances of Conflux come from three distinctive designs: the Conflux dynamic I/O management mechanism, the SEED learning-based I/O mode selection policy, and the fine-grained lock and lease mechanisms. We have implemented and evaluated Conflux on a cluster with real PM and RDMA devices. Experimental results show that Conflux outperforms existing DFSs significantly.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback and insightful suggestions. This work is supported by National Natural Science Foundation of China (NSFC) (Grant No. 62227809), National Key Research and Development Program of China (Grant No. 2022YFB4500303), the Fundamental Research Funds for the Central Universities, Shanghai Municipal Science and Technology Major Project (Grant No. 2021SHZDZX0102), Natural Science Foundation of Shanghai (Grant No. 22ZR1435400), and CCF Huawei Populus Grove Fund (No. CCF-HuaweiSY202202).

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 Fifteenth 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 *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 1011–1027.
[3] Jens Axboe. 2021. Flexible I/O tester. 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 *19th {USENIX} Conference on File and Storage Technologies ({FAST} 21)*. 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] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices* 48, 4 (2013), 77–88.

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

[8] 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.

[9] 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.

[10] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. {PCC} vivace: Online-learning congestion control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 343–356.

[11] 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 (TOS)* 17, 4 (2021), 1–32.

[12] 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.

[13] 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 Ninth European Conference on Computer Systems*. 1–15.

[14] Facebook-RocksDB. 2021. db_bench. https://github-wiki-see.page/m/facebook/rocksdb/wiki/Benchmarking-tools.

[15] Facebook-RocksDB. 2021. A Persistent Key-value Store for Fast Storage Environment. https://rocksdb.org/.

[16] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 29–43.

[17] Google-Tensorflow. 2021. TensorFlow. https://github.com/tensorflow/tensorflow/tree/master/tensorflow.

[18] 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 *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 173–190.

[19] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning memory access patterns. In *International Conference on Machine Learning*. PMLR, 1919–1928.

[20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[21] Intel. 2021. Achieve Greater Insight From Your Data with Intel Optane Persistent Memory. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html.

[22] 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 *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[23] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[24] 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.

[25] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. Design guidelines for high performance {RDMA} systems. In *2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16)*. 437–450.

[26] 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.

[27] 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.

[28] 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.

[29] Huaicheng Li, Martin L Putra, Ronald Shi, Xing Lin, Gregory R Ganger, and Haryadi S Gunawi. 2021. IODA: 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.

[30] 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 *SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. IEEE, 39–39.

[31] 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.

[32] 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 *2013 42nd International Conference on Parallel Processing*. IEEE, 641–650.

[33] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. 2017. Octopus: an rdma-enabled distributed persistent memory file system. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*. 773–785.

[34] 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 Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 757–773.

[35] 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.

[36] Yujie Ren, Changwoo Min, and Sudarsun Kannan. 2020. CrossFS: A cross-layered direct-access file system. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 137–154.

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

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

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

[40] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE 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.

[41] Matthew Wilcox. 2014. DAX: Page cache bypass for filesystems on memory storage. https://lwn.net/Articles/618064/

[42] 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 *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 281–296.

[43] 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.

[44] 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 *19th {USENIX} Conference on File and Storage Technologies ({FAST} 21)*. 307–323.

[45] Jian Xu and Steven Swanson. 2016. {NOVA}: A log-structured file system for hybrid volatile/non-volatile main memories. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*. 323–338.

[46] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2019. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks. In *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*. 221–234.

[47] Jian Yang, Joseph Izraelevitz, and Steven Swanson. 2020. FileMR: Rethinking {RDMA} Networking for Scalable Persistent Memory. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 111–125.

[48] 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 *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 933–944.

[49] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. 2020. RLScheduler: an automated HPC batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[50] Yiying Zhang and Yutong Huang. 2019. " Learned" Operating Systems. *ACM SIGOPS Operating Systems Review* 53, 1 (2019), 40–45.

[51] Shengan Zheng, Morteza Hoseinzadeh, and Steven Swanson. 2019. Ziggurat: a tiered file system for non-volatile main memories and disks. In *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*. 207–219.

[52] 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 *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 265–280.

[53] 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 (TOS)* 17, 3 (2021), 1–25.