Orion: A Distributed File System for Non-Volatile Main Memory and RDMA-Capable Networks

Jian Yang, Joseph Izraelevitz, and Steven Swanson, *UC San Diego*

https://www.usenix.org/conference/fast19/presentation/yang

This paper is included in the Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19).

February 25–28, 2019 • Boston, MA, USA

978-1-939133-09-0

Open access to the Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19) is sponsored by

NetApp®

Orion: A Distributed File System for Non-Volatile Main Memories and RDMA-Capable Networks

Jian Yang Joseph Izraelevitz Steven Swanson
UC San Diego
{*jianyang, jizraelevitz, swanson*}*@eng.ucsd.edu*

Abstract

High-performance, byte-addressable non-volatile main memories (NVMMs) force system designers to rethink tradeoffs throughout the system stack, often leading to dramatic changes in system architecture. Conventional distributed file systems are a prime example. When faster NVMM replaces block-based storage, the dramatic improvement in storage performance makes networking and software overhead a critical bottleneck.

In this paper, we present *Orion*, a distributed file system for NVMM-based storage. By taking a clean slate design and leveraging the characteristics of NVMM and high-speed, RDMA-based networking, Orion provides high-performance metadata and data access while maintaining the byte addressability of NVMM. Our evaluation shows Orion achieves performance comparable to local NVMM file systems and outperforms existing distributed file systems by a large margin.

1 Introduction

In a distributed file system designed for block-based devices, media performance is almost the sole determiner of performance on the data path. The glacial performance of disks (both hard and solid state) compared to the rest of the storage stack incentivizes complex optimizations (e.g., queuing, striping, and batching) around disk accesses. It also saves designers from needing to apply similarly aggressive optimizations to network efficiency, CPU utilization, and locality, while pushing them toward software architectures that are easy to develop and maintain, despite the (generally irrelevant) resulting software overheads.

The appearance of fast non-volatile memories (e.g., Intel's 3D XPoint DIMMs [28]) on the processor's memory bus will offer an abrupt and dramatic increase in storage system performance, providing performance characteristics comparable to DRAM and vastly faster than either hard drives or SSDs. These non-volatile main memories (NVMM) upend the traditional design constraints of distributed file systems.

For an NVMM-based distributed file system, media access performance is no longer the major determiner of performance. Instead, network performance, software overhead, and data placement all play central roles. Furthermore, since NVMM is byte-addressable, block-based interfaces are no longer a constraint. Consequently, old distributed file systems squander NVMM performance — the previously negligible inefficiencies quickly become the dominant source of delay.

This paper presents Orion, a distributed file system designed from the ground up for NVMM and Remote Direct Memory Access (RDMA) networks. While other distributed systems [41, 55] have integrated NVMMs, Orion is the first distributed file system to systematically optimize for NVMMs throughout its design. As a result, Orion diverges from block-based designs in novel ways.

Orion focuses on several areas where traditional distributed file systems fall short when naively adapted to NVMMs. We describe them below.

Use of RDMA Orion targets systems connected with an RDMA-capable network. It uses RDMA whenever possible to accelerate both metadata and data accesses. Some existing distributed storage systems use RDMA as a fast transport layer for data access [10, 18, 62, 63, 71] but do not integrate it deeply into their design. Other systems [41, 55] adapt RDMA more extensively but provide object storage with customized interfaces that are incompatible with file system features such as unrestricted directories and file extents, symbolic links and file attributes.

Orion is the first full-featured file system that integrates RDMA deeply into all aspects of its design. Aggressive use of RDMA means the CPU is not involved in many transfers, lowering CPU load and improving scalability for handling incoming requests. In particular, pairing RDMA with NVMMs allows nodes to directly access remote storage without any target-side software overheads.

Software Overhead Software overhead in distributed files system has not traditionally been a critical concern. As such, most distributed file systems have used two-layer designs that divide the network and storage layers into separate modules.

	Read Latency	Bandwidth GB/s	
	512 B	Read	Write
DRAM	80 ns	60	30
NVMM	300 ns	8	2
RDMA NIC	3 μs	5 (40 Gbps)	
NVMe SSD	70 μs	3.2	1.3

Table 1: **Characteristics of memory and network devices** We measure the fisrt 3 lines on Intel Sandy Bridge-EP platform with a Mellanox ConnectX-4 RNIC and an Intel DC P3600 SSD. NVMM numbers are estimated based on assumptions made in [75].

Two-layer designs trade efficiency for ease of implementation. Designers can build a user-level daemon that stitches together off-the-shelf networking packages and a local file system into a distributed file system. While expedient, this approach results in duplicated metadata, excessive copying, unnecessary event handling, and places user-space protection barriers on the critical path.

Orion merges the network and storage functions into a single, kernel-resident layer optimized for RDMA and NVMM that handles data, metadata, and network access. This decision allows Orion to explore new mechanisms to simplify operations and scale performance.

Locality RDMA is fast, but it is still several times slower than local access to NVMMs (Table 1). Consequently, the location of stored data is a key performance concern for Orion. This concern is an important difference between Orion and traditional block-based designs that generally distinguish between client nodes and a pool of centralized storage nodes [18, 53]. Pooling makes sense for block devices, since access latency is determined by storage, rather than network latency, and a pool of storage nodes simplifies system administration. However, the speed of NVMMs makes a storage pool inefficient, so Orion optimizes for locality. To encourage local accesses, Orion migrates durable data to the client whenever possible and uses a novel delegated allocation scheme to efficiently manage free space.

Our evaluation shows that Orion outperforms existing distributed file systems by a large margin. Relative to local NVMM filesystems, it provides comparable application-level performance when running applications on a single client. For parallel workloads, Orion shows good scalability: performance on an 8-client cluster is between 4.1× and 7.9× higher than running on a single node.

The rest of the paper is organized as follows. We discuss the opportunities and challenges of building a distributed file system utilizing NVMM and RDMA in Section 2. Section 3 gives an overview of Orion's architecture. We describe the design decisions we made to implement high-performance metadata access and data access in Sections 4 and 5 respectively. Section 6 evaluates these mechanisms. We cover related work in Section 7 and conclude in Section 8.

2 Background and Motivation

Orion is a file system designed for distributed shared NVMM and RDMA. This section gives some background on NVMM and RDMA and highlights the opportunities these technologies provide. Then, it discusses the inefficiencies inherent in running existing distributed file systems on NVMM.

2.1 Non-Volatile Main Memory

NVMM is comprised of nonvolatile DIMMs (NVDIMMs) attached to a CPU's memory bus alongside traditional DRAM DIMMs. Battery-backed NVDIMM-N modules are commercially available from multiple vendors [46], and Intel's 3DX-Point memory [28] is expected to debut shortly. Other technologies such as spin-torque transfer RAM (STT-RAM) [45], ReRAM [27] are in active research and development.

NVMMs appear as contiguous, persistent ranges of physical memory addresses [52]. Instead of using block-based interface, file systems can issue load and store instructions to NVMMs directly. NVMM file systems provide this ability via direct access (or "DAX"), which allows read and write system calls to bypass the page cache.

Researchers and companies have developed several file systems designed specifically for NVMM [15, 21, 25, 73, 74]. Other developers have adapted existing file systems to NVMM by adding DAX support [14, 70]. In either case, the file system must account for the 8-byte atomicity guarantees that NVMMs provide (compared to sector atomicity for disks). They also must take care to ensure crash consistency by carefully ordering updates to NVMMs using cache flush and memory barrier instructions.

2.2 RDMA Networking

Orion leverages RDMA to provide low latency metadata and data accesses. RDMA allows a node to perform one-sided read/write operations from/to memory on a remote node in addition to two-sided send/recv operations. Both user- and kernel-level applications can directly issue remote DMA requests (called *verbs*) on pre-registered memory regions (MRs). One-sided requests bypass CPU on the remote host, while two-sided requests require the CPU to handle them.

An RDMA NIC (RNIC) is capable of handling MRs registered on both virtual and physical address ranges. For MRs on virtual addresses, the RDMA hardware needs to translate from virtual addresses to DMA addresses on incoming packets. RNICs use a hardware pin-down cache [65] to accelerate lookups. Orion uses physically addressed DMA MRs, which do not require address translation on the RNIC, avoiding

the possibility of pin-down cache misses on large NVMM regions.

Software initiates RDMA requests by posting work queue entries (WQE) onto a pair of send/recv queues (a queue pair or "QP"), and polling for their completion from the completion queue (CQ). On completing a request, the RNIC signals completion by posting a completion queue entry (CQE).

A send/recv operation requires both the sender and receiver to post requests to their respective send and receive queues that include the source and destination buffer addresses. For one-sided transfers, the receiver grants the sender access to a memory region through a shared, secret 32-bit "rkey." When the receiver RNIC processes an inbound one-sided request with a matching rkey, it issues DMAs directly to its local memory without notifying the host CPU.

Orion employs RDMA as a fast transport layer, and its design accounts for several idiosyncrasies of RDMA:

Inbound verbs are cheaper Inbound verbs, including recv and incoming one-sided read/write, incur lower overhead for the target, so a single node can handle many more inbound requests than it can initiate itself [59]. Orion's mechanisms for accessing data and synchronizing metadata across clients both exploit this asymmetry to improve scalability.

RDMA accesses are slower than local accesses RDMA accesses are fast but still slower than local accesses. By combining the data measured on DRAM and the methodology introduced in a previous study [75], we estimate the one-sided RDMA NVMM read latency to be $\sim$9$\times$ higher than local NVMM read latency for 64 B accesses, and $\sim$20$\times$ higher for 4 KB accesses.

RDMA favors short transfers RNICs implement most of the RDMA protocol in hardware. Compared to transfer protocols like TCP/IP, transfer size is more important to transfer latency for RDMA because sending smaller packets involves fewer PCIe transactions [35]. Also, modern RDMA hardware can inline small messages along with WQE headers, further reducing latency. To exploit these characteristics, Orion aggressively minimizes the size of the transfers it makes.

RDMA is not persistence-aware Current RDMA hardware does not guarantee persistence for one-sided RDMA writes to NVMM. Providing this guarantee generally requires an extra network round-trip or CPU involvement for cache flushes [22], though a proposed [60] RDMA "commit" verb would provide this capability. As this support is not yet available, Orion ensures persistence by CPU involvement (see Section 5.3).

3 Design Overview

Orion is a distributed file system built for the performance characteristics of NVMM and RDMA networking. NVMM's low latency and byte-addressability fundamentally alter the relationship among memory, storage, and network, motivating Orion to use a clean-slate approach to combine the file system and networking into a single layer. Orion achieves the following design goals:

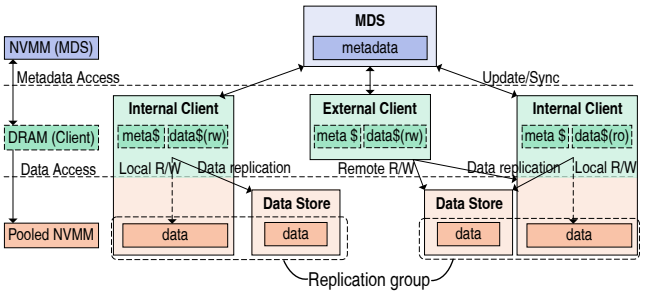

Figure 1: **Orion cluster organization** An Orion cluster consists of a metadata server, clients and data stores.

- **Scalable performance with low software overhead:** Scalability and low-latency are essential for Orion to fully exploit the performance of NVMM. Orion achieves this goal by unifying file system functions and network operations and by accessing data structures on NVMM directly through RDMA.
- **Efficient network usage on metadata updates:** Orion caches file system data structures on clients. A client can apply file operations locally and only send the changes to the metadata server over the network.
- **Metadata and data consistency:** Orion uses a log-structured design to maintain file system consistency at low cost. Orion allows read parallelism but serializes updates for file system data structures across the cluster. It relies on atomically updated inode logs to guarantee metadata and data consistency and uses a new coordination scheme called *client arbitration* to resolve conflicts.
- **DAX support in a distributed file system:** DAX-style (direct load/store) access is a key benefit of NVMMs. Orion allows clients to access data in its local NVMM just as it could access a DAX-enabled local NVMM file system.
- **Repeated access become local access:** Orion exploits locality by migrating data to where writes occur and making data caching an integral part of the file system design. The log-structured design reduces the cost of maintaining cache coherence.
- **Reliability and data persistence:** Orion supports metadata and data replication for better reliability and availability. The replication protocol also guarantees data persistency.

The remainder of this section provides an overview of the Orion software stack, including its hardware and software organization. The following sections provide details of how Orion manages metadata (Section 4) and provides access to data (Section 5).

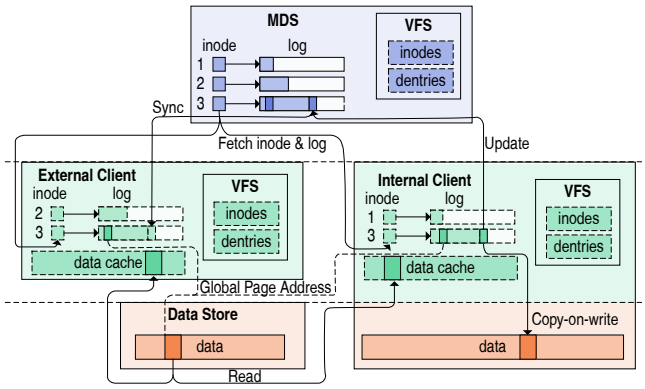

Figure 2: **Orion software organization** Orion exposes as a log-structured file system across MDS and clients. Clients maintain local copies of inode metadata and sync with the MDS, and access data at remote data stores or local NVMM directly.

3.1 Cluster Organization

An Orion cluster consists of a *metadata server (MDS)*, several *data stores (DSs)* organized in replication groups, and *clients* all connected via an RDMA network. Figure 1 shows the architecture of an Orion cluster and illustrates these roles.

The MDS manages metadata. It establishes an RDMA connection to each of the clients. Clients can propagate local changes to the MDS and retrieve updates made by other clients.

Orion allows clients to manage and access a global, shared pool of NVMMs. Data for a file can reside at a single DS or span multiple DSs. A client can access a remote DS using one-sided RDMA and its local NVMMs using load and store instructions.

Internal clients have local NVMM that Orion manages. Internal clients also act as a DSs for other clients. *External clients* do not have local NVMM, so they can access data on DSs but cannot store data themselves.

Orion supports replication of both metadata and data. The MDS can run as a high-availability pair consisting of a primary server and a mirror using Mojim [76]-style replication. Mojim provides low latency replication for NVMM by maintaining a single replica and only making updates at the primary.

Orion organizes DSs into replication groups, and the DSs in the group have identical data layouts. Orion uses broadcast replication for data.

3.2 Software Organization

Orion's software runs on the clients and the MDS. It exposes a normal POSIX interface and consists of kernel modules that manage file and metadata in NVMM and handle communication between the MDS and clients. Running in the kernel avoids the frequent context switches, copies, and kernel/user crossing that conventional two-layer distributed file systems designs require.

The file system in Orion inherits some design elements from NOVA [73, 74], a log-structured POSIX-compliant local NVMM file system. Orion adopts NOVA's highly-optimized mechanisms for managing file data and metadata in NVMM. Specifically, Orion's local file system layout, inode log data structure, and radix trees for indexing file data in DRAM are inherited from NOVA, with necessary changes to make metadata accessible and meaningful across nodes. Figure 2 shows the overall software organization of the Orion file system.

An Orion inode contains pointers to the head and tail of a metadata log stored in a linked list of NVMM pages. A log's entries record all modifications to the file and hold pointers to the file's data blocks. Orion uses the log to build virtual file system (VFS) inodes in DRAM along with indices that map file offsets to data blocks. The MDS contains the metadata structures of the whole file system including authoritative inodes and their logs. Each client maintains a local copy of each inode and its logs for the files it has opened.

Copying the logs to the clients simplifies and accelerates metadata management. A client can recover all metadata of a file by walking through the log. Also, clients can apply a log entry locally in response to a file system request and then propagate it to the MDS. A client can also tell whether an inode is up-to-date by comparing the local and remote log tail. An up-to-date log should be equivalent on both the client and the MDS, and this invariant is the basis for our metadata coherency protocol. Because MDS inode log entries are immutable except during garbage collection and logs are append-only, logs are amenable to direct copying via RDMA reads (see Section 4).

Orion distributes data across DSs (including the internal clients) and replicates the data within replication groups. To locate data among these nodes, Orion uses *global page addresses* (GPAs) to identify pages. Clients use a GPA to locate both the replication group and data for a page. For data reads, clients can read from any node within a replication group using the global address. For data updates, Orion performs a copy-on-write on the data block and appends a log entry reflecting the change in metadata (e.g., write offset, size, and the address to the new data block). For internal clients, the copy-on-write migrates the block into the local NVMM if space is available.

An Orion client also maintains a client-side data cache. The cache, combined with the copy-on-write mechanism, lets Orion exploit and enhance data locality. Rather than relying on the operating system's generic page cache, Orion manages DRAM as a customized cache that allows it to access cached pages using GPAs without a layer of indirection. This also simplifies cache coherence.

4 Metadata Management

Since metadata updates are often on an application's critical path, a distributed file system must handle metadata requests quickly. Orion's MDS manages all metadata updates and holds the authoritative, persistent copy of metadata. Clients cache metadata locally as they access and update files, and they must propagate changes to both the MDS and other clients to maintain coherence.

Below, we describe how Orion's metadata system meets both these performance and correctness goals using a combination of communication mechanisms, latency optimizations, and a novel arbitration scheme to avoid locking.

4.1 Metadata Communication

The MDS orchestrates metadata communication in Orion, and all authoritative metadata updates occur there. Clients do not exchange metadata. Instead, an Orion client communicates with the MDS to fetch file metadata, commit changes and apply changes committed by other clients.

Clients communicate with the MDS using three methods depending on the complexity of the operation they need to perform: (1) direct *RDMA reads*, (2) speculative and highly-optimized *log commits*, and (3) acknowledged *remote procedure calls* (RPCs).

These three methods span a range of options from simple/-lightweight (direct RDMA reads) to complex/heavyweight (RPC). We use RDMA reads from the MDS whenever possible because they do not require CPU intervention, maximizing MDS scalability.

Below, we describe each of these mechanisms in detail followed by an example. Then, we describe several additional optimizations Orion applies to make metadata updates more efficient.

RDMA reads Clients use one-sided RDMA reads to pull metadata from the MDS when needed, for instance, on file open. Orion uses wide pointers that contain a pointer to the client's local copy of the metadata as well as a GPA that points to the same data on the MDS. A client can walk through its local log by following the local pointers, or fetch the log pages from the MDS using the GPAs.

The clients can access the inode and log for a file using RDMA reads since NVMM is byte addressable. These accesses bypass the MDS CPU, which improves scalability.

Log commits Clients use log commits to update metadata for a file. The client first performs file operations locally by appending a log entry to the local copy of the inode log. Then it forwards the entry to the MDS and waits for completion.

Log commits use RDMA sends. Log entries usually fit in two cache lines, so the RDMA NIC can send them as inlined messages, further reducing latencies. Once it receives the acknowledgment for the send, the client updates its local log

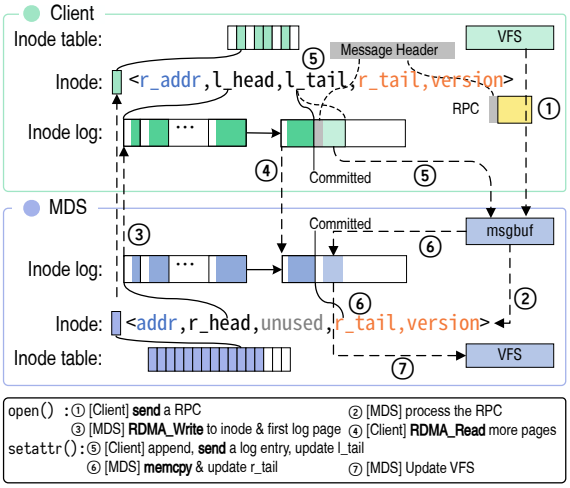

Figure 3: **Orion metadata communication** Orion maintains metadata structures such as inode logs on both MDS and clients. A client commit file system updates through Log Commits and RPCs.

tail, completing the operation. Orion allows multiple clients to commit log entries of a single inode without distributed locking using a mechanism called *client arbitration* that can resolve inconsistencies between inode logs on the clients (Section 4.3).

Remote procedure calls Orion uses synchronous remote procedure calls (RPCs) for metadata accesses that involve multiple inodes as well as operations that affect other clients (e.g., a file write with `O_APPEND` flag).

Orion RPCs use a send verb and an RDMA write. An RPC message contains an opcode along with metadata updates and/or log entries that the MDS needs to apply atomically. The MDS performs the procedure call and responds via one-sided RDMA write or message send depending on the opcode. The client blocks until the response arrives.

Example Figure 3 illustrates metadata communication. For `open()` (an RPC-based metadata update), the client allocates space for the inode and log, and issues an RPC ①. The MDS handles the RPC ② and responds by writing the inode along with the first log page using RDMA ③. The client uses RDMA to read more pages if needed and builds VFS data structures ④.

For a `setattr()` request (a log commit based metadata update), the client creates a local entry with the update and issues a log commit ⑤. It then updates its local tail pointer atomically after it has sent the log commit. Upon receiving the log entry, the MDS appends the log entry, updates the log tail ⑥, and updates the corresponding data structure in VFS ⑦.

RDMA Optimizations Orion avoids data copying within

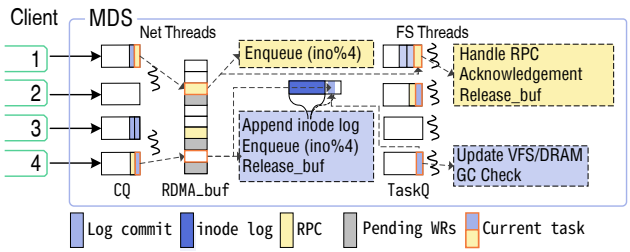

Figure 4: **MDS request handling** The MDS handles client requests in two stages: First, networking threads handle RDMA completion queue entries (CQEs) and dispatch them to file system threads. Next, file system threads handle RPCs and update the VFS.

a node whenever possible. Both client-initiated RDMA reads and MDS-initiated RDMA writes (e.g., in response to an RPC) target client file system data structures directly. Additionally, log entries in Orion contain extra space (shown as *message headers* in Figure 3) to accommodate headers used for networking. Aside from the DMA that the RNIC performs, the client copies metadata at most once (to avoid concurrent updates to the same inode) during a file operation.

Orion also uses *relative pointers* in file system data structures to leverage the linear addressing in kernel memory management. NVMM on a node appears as contiguous memory regions in both kernel virtual and physical address spaces. Orion can create either type of address by adding the relative pointer to the appropriate base address. Relative pointers are also meaningful across power failures.

4.2 Minimizing Commit Latency

The latency of request handling, especially for log commits, is critical for the I/O performance of the whole cluster. Orion uses dedicated threads to handle per-client receive queues as well as file system updates. Figure 4 shows the MDS request handling process.

For each client, the MDS registers a small (256 KB) portion of NVMM as a communication buffer. The MDS handles incoming requests in two stages: A *network thread* polls the RDMA completion queues (CQs) for work requests on pre-posted RDMA buffers and dispatches the requests to *file system threads*. As an optimization, the MDS prioritizes log commits by allowing network threads to append log entries directly. Then, a file system thread handles the requests by updating file system structures in DRAM for a log commit or serving the requests for an RPC. Each file system thread maintains a FIFO containing pointers to updated log entries or RDMA buffers holding RPC requests.

For a log commit, a network thread reads the inode number, appends the entry by issuing non-temporal moves and then atomically updates the tail pointer. At this point, other clients can read the committed entry and apply it to their local copy

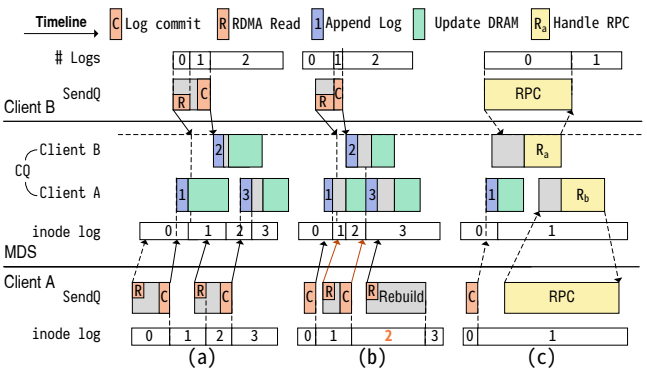

Figure 5: **Metadata consistency in Orion** The inode log on Client A is consistent after (a) updating the log entry committed by another client using RDMA reads, (c) issuing an RPC, and (b) rebuilding the log on conflicts.

of the inode log. The network thread then releases the recv buffer by posting a recv verb, allowing its reuse. Finally, it dispatches the task for updating in-DRAM data structures to a file system thread based on the inode number.

For RPCs, the network thread dispatches the request directly to a file system thread. Each thread processes requests to a subset of inodes to ensure better locality and less contention for locks. The file system threads use lightweight journals for RPCs involving inodes that belong to multiple file system threads.

File system threads perform garbage collection (GC) when the number of "dead" entries in a log becomes too large. Orion rebuilds the inode log by copying live entries to new log pages. It then updates the log pointers and increases the version number. Orion makes this update atomic by packing the version number and tail pointer into 64 bits. The thread frees stale log pages after a delay, allowing ongoing RDMA reads to complete. Currently we set the maximal size of file writes in a log entry to be 512 MB.

4.3 Client Arbitration

Orion allows multiple clients to commit log entries to a single inode at the same time using a mechanism called *client arbitration* rather than distributed locking. Client arbitration builds on the following observations:

1. Handling an inbound RDMA read is much cheaper than sending an outbound write. In our experiments, a single host can serve over 15 M inbound reads per second but only 1.9 M outbound writes per second.
2. For the MDS, CPU time is precious. Having the MDS initiate messages to maintain consistency will reduce Orion performance significantly.
3. Log append operations are lightweight: each one takes around just 500 CPU cycles.

A client commits a log entry by issuing a send verb and

polling for its completion. The MDS appends log commits based on arrival order and updates log tails atomically. A client can determine whether a local inode is up-to-date by comparing the log length of its local copy of the log and the authoritative copy at the MDS. Clients can check the length of an inode's log by retrieving its tail pointer with an RDMA read.

The client issues these reads in the background when handling an I/O request. If another client has modified the log, the client detects the mismatch and fetches the new log entries using additional RDMA reads and retries.

If the MDS has committed multiple log entries in a different order due to concurrent accesses, the client blocks the current request and finds the last log entry that is in sync with the MDS, it then fetches all following log entries from the MDS, rebuilds its in-DRAM structures, and re-executes the user request.

Figure 5 shows the three different cases of concurrent accesses to a single inode. In (a), the client A can append the log entry #2 from client B by extending its inode log. In (b), the client A misses the log entry #2 committed by client B, so it will rebuild the inode log on the next request. In (c), the MDS will execute concurrent RPCs to the same inode sequentially, and the client will see the updated log tail in the RPC acknowledgment.

A rebuild occurs when all of the following occur at the same time: (1) two or more clients access the same file at the same time and one of the accesses is log commit, (2) one client issues two log commits consecutively, and (3) the MDS accepts the log commit from another client after the client RDMA reads the inode tail but before the MDS accepts the second log commit.

In our experience this situation happens very rarely, because the "window of vulnerability" – the time required to perform a log append on the MDS – is short. That said, Orion lets applications identify files that are likely targets of intensive sharing via an `ioctl`. Orion uses RPCs for all updates to these inodes in order to avoid rebuilds.

5 Data Management

Orion pools NVMM spread across internal clients and data stores. A client can allocate and access data either locally (if the data are local) or remotely via one-sided RDMA. Clients use local caches and migration during copy-on-write operations to reduce the number of remote accesses.

5.1 Delegated Allocation

To avoid allocating data on the critical path, Orion uses a distributed, two-stage memory allocation scheme.

The MDS keeps a bitmap of all the pages Orion manages. Clients request large chunks of storage space from the MDS via an RPC. The client can then autonomously allocate space

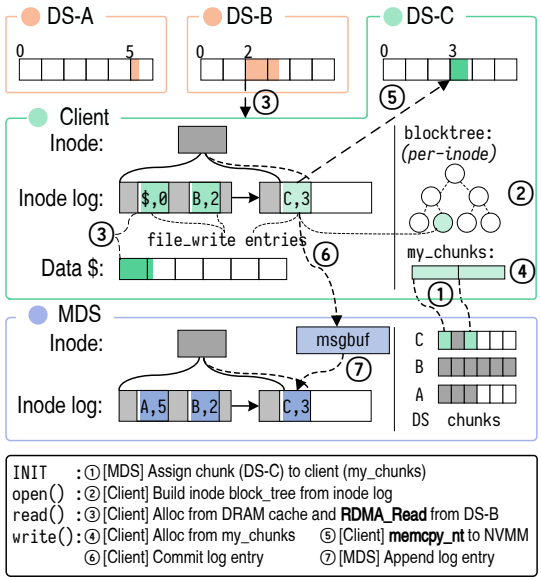

Figure 6: **Orion data communication** Orion allows clients manage and access data independently.

within those chunks. This design frees the MDS from managing fine-grain data blocks, and allows clients to allocate pages with low overhead.

The MDS allocates internal clients chunks of its local NVMM when possible since local writes are faster. As a result, most of their writes go to local NVMM.

5.2 Data Access

To read file data, a client either communicates with the DS using one-sided RDMA or accesses its local NVMM via DAX (if it is an internal client and the data is local). Remote reads use one-sided RDMA reads to retrieve existing file data and place it in local DRAM pages that serve as a cache for future reads.

Remote writes can also be one-sided because allocation occurs at the client. Once the transfer is complete, the client issues a log commit to the MDS.

Figure 6 demonstrates Orion's data access mechanisms. A client can request a block chunk from the MDS via an RPC ①. When the client opens a file, it builds a radix tree for fast lookup from file offsets to log entries ②. When handling a `read()` request, the client reads from the DS (DS-B) to its local DRAM and update the corresponding log entry ③. For a `write()` request, it allocates from its local chunk ④ and issues `memcpy_nt()` and `sfence` to ensure that the data reaches its local NVMM (DS-C) ⑤. Then a log entry containing information such as the GPA and size is committed to the MDS ⑥. Finally, the MDS appends the log entry ⑦.

5.3 Data Persistence

Orion always ensures that metadata is consistent, but, like many file systems, it can relax the consistency requirement on data based on user preferences and the availability of replication.

The essence of Orion's data consistency guarantee is the extent to which the MDS delays the log commit for a file update. For a weak consistency guarantee, an external client can forward a speculative log commit to the MDS before its remote file update has completed at a DS. This consistency level is comparable to the write-back mode in ext4 and can result in corrupted data pages but maintains metadata integrity. For strong data consistency that is comparable to NOVA and the data journaling mode in ext4, Orion can delay the log commit until after the file update is persistent at multiple DSs in the replication group.

Achieving strong consistency over RDMA is hard because RDMA hardware does not provide a standard mechanism to force writes into remote NVMM. For strongly consistent data updates, our algorithm is as follows.

A client that wishes to make a consistent file update uses copy-on-write to allocate new pages on all nodes in the appropriate replica group, then uses RDMA writes to update the pages. In parallel, the client issues a speculative log commit to the MDS for the update.

DSs within the replica group detect the RDMA writes to new pages using an RDMA trick: when clients use RDMA writes on the new pages, they include the page's global address as an *immediate value* that travels to the target in the RDMA packet header. This value appears in the target NIC's completion queue, so the DS can detect modifications to its pages. For each updated page, the DS forces the page into NVMM and sends an acknowledgment via a small RDMA write to the MDS, which processes the client's log commit once it reads a sufficient number of acknowledgments in its DRAM.

5.4 Fault Tolerance

The high performance and density of NVMM makes the cost of rebuilding a node much higher than recovering it. Consequently, Orion makes its best effort to recover the node after detecting an error. If the node can recover (e.g., after a power failure and most software bugs), it can rejoin the Orion cluster and recover to a consistent state quickly. For NVMM media errors, module failures, or data-corrupting bugs, Orion rebuilds the node using the data and metadata from other replicas. It uses relative pointers and global page addresses to ensure metadata in NVMM remain meaningful across power failures.

In the metadata subsystem, for MDS failures, Orion builds a Mojim-like [76] high-availability pair consisting of a primary MDS and a mirror. All metadata updates flow to the primary MDS, which propagates the changes to the mirror. When the primary fails, the mirror takes over and journals all the incoming requests while the primary recovers.

In the data subsystem, for DS failures, the DS journals the immediate values of incoming RDMA write requests in a circular buffer. A failed DS can recover by obtaining the pages committed during its downtime from a peer DS in the same replication group. When there are failed nodes in a replication group, the rest of the nodes work in the strong data consistency mode introduced in Section 5.3 to ensure successful recovery in the event of further failures.

6 Evaluation

In this section, we evaluate the performance of Orion by comparing it to existing distributed file systems as well as local file systems. We answer the following questions:

- How does Orion's one-layer design affect its performance compared to existing two-layer distributed file systems?
- How much overhead does managing distributed data and metadata add compared to running a local NVMM file system?
- How does configuring Orion for different levels of reliability affect performance?
- How scalable is Orion's MDS?

We describe the experimental setup and then evaluate Orion with micro- and macrobenchmarks. Then we measure the impact of data replication and the ability to scale over parallel workloads.

6.1 Experimental Setup

We run Orion on a cluster with 10 nodes configured to emulate persistent memory with DRAM. Each node has two quad-core Intel Xeon (Westmere-EP) CPUs with 48 GB of DRAM, with 32 GB configured as an emulated `pmem` device. Each node has an RDMA NIC (Mellanox ConnectX-2 40 Gbps HCA) running in Infiniband mode and connects to an Infiniband switch (QLogic 12300). We disabled the Direct Cache Access feature on DSs. To demonstrate the impact to co-located applications, we use a dedicated core for issuing and handling RDMA requests on each client.

We build our Orion prototype on the Linux 4.10 kernel with the RDMA verb kernel modules from Mellanox OFED [42]. The file system in Orion reuses code from NOVA but adds $\sim$8K lines of code to support distributed functionalities and data structures. The networking module in Orion is built from scratch and comprises another $\sim$8K lines of code.

We compare Orion with three distributed file systems Ceph [69], Gluster [19], and Octopus [41] running on the same RDMA network. We also compare Orion to ext4 mounted on a remote iSCSI target hosting a ramdisk using iSCSI Extension over RDMA (iSER) [12] (denoted by Ext4/iSER), which provides the client with private access to

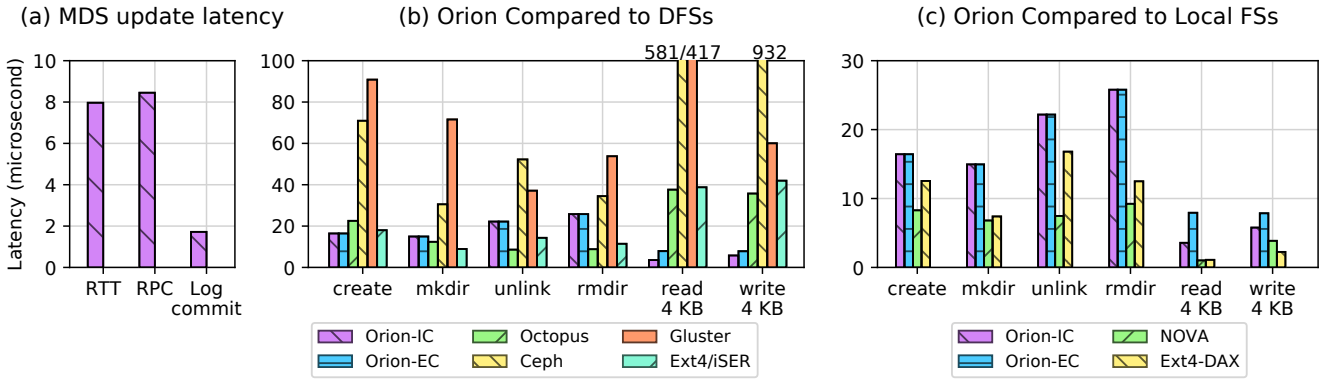

Figure 7: **Average latency of Orion metadata and data operations** Orion is built on low-latency communication primitive (a). These lead to basic file operation latencies that are better than existing remote-access storage system (b) and within a small factor of local NVMM file systems (c).

a remote block device. Finally, we compare our system with two local DAX file systems: NOVA [73, 74] and ext4 in DAX mode (ext4-DAX).

6.2 Microbenchmarks

We begin by measuring the networking latency of log commits and RPCs. Figure 7(a) shows the latency of a log commit and an RPC compared to the network round trip time (RTT) using two sends verbs. Our evaluation platform has a network round trip time of 7.96 μs. The latency of issuing an Orion RPC request and obtaining the response is 8.5 μs. Log commits have much lower latency since the client waits until receiving the acknowledgment of an RDMA send work request, which takes less than half of the network round trip time: they complete in less than 2 μs.

Figure 7(b) shows the metadata operation latency on Orion and other distributed file systems. We evaluated basic file system metadata operations such as `create`, `mkdir`, `unlink`, `rmdir` as well as reading and writing random 4 KB data using FIO [6]. Latencies for Ceph and Gluster are between 34% and 443% higher than Orion.

Octopus performs better than Orion on `mkdir`, `unlink` and `rmdir`, because Octopus uses a simplified file system model: it maintains all files and directories in a per-server hash table indexed by their full path names and it assigns a fixed number of file extents and directory entries to each file and directory. This simplification means it cannot handle large files or directories.

Ext4/iSER outperforms Orion on some metadata operations because it considers metadata updates complete once they enter the block queue. In contrast, NVMM-aware systems (such as Orion or Octopus) report the full latency for persistent metadata updates. The 4 KB read and write measurements in the figure give a better measure of I/O latency – Orion outperforms Ext4/iSER configuration by between 4.9× and 10.9×.

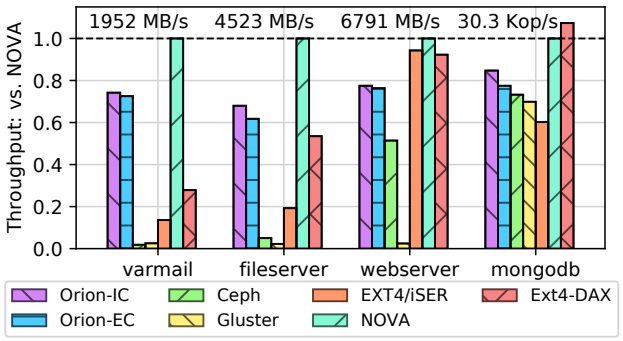

Figure 8: **Application performance on Orion** The graph is normalized to NOVA, and the annotations give NOVA's performance. For write-intensive workloads, Orion outperforms Ceph and Gluster by a wide margin.

For file reads and writes, Orion has the lowest latency among all the distributed file systems we tested. For internal clients (Orion-IC), Orion's 4 KB read latency is 3.6 μs and 4 KB write latency of 5.8 μs. For external clients (Orion-EC), the write latency is 7.9 μs and read latency is similar to internal clients because of client-side caching. For cache misses, read latency is 7.9 μs.

We compare Orion to NOVA and Ext4-DAX in Figure 7(c). For metadata operations, Orion sends an RPC to the MDS on the critical path, increasing latency by between 98% to 196% compared to NOVA and between 31% and 106% compared to Ext4-DAX. If we deduct the networking round trip latency, Orion increases software overheads by 41%.

6.3 Macrobenchmarks

We use three Filebench [64] workloads (varmail, fileserver and webserver) as well as MongoDB [4] running YCSB's [16]

Workload	# Threads	# Files	Avg. File Size	R/W Size	Append Size
varmail	8	30 K	16 KB	1 MB	16 KB
fileserver	8	10 K	128 KB	1 MB	16 KB
webserver	8	50 K	64 KB	1 MB	8 KB
mongodb	12	YCSB-A, RecordCount=1M, OpCount=10M			

Table 2: **Application workload characteristics** This table includes the configurations for three filebench workloads and the properties of YCSB-A.

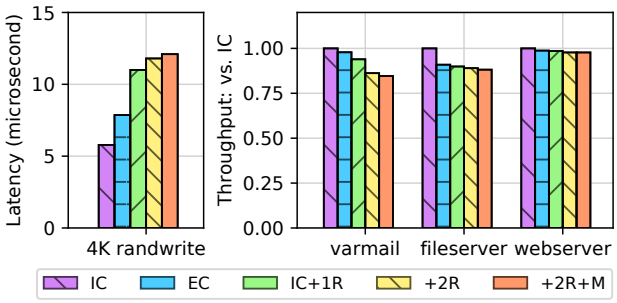

Figure 9: **Orion data replication performance** Updating a remote replica adds significantly to random write latency, but the impact on overall benchmark performance is small.

Workload A (50% read/50% update) to evaluate Orion. Table 2 describes the workload characteristics. We could not run these workloads on Octopus because it limits the directory entries and the number of file extents, and it ran out of memory when we increased those limits to meet the workloads' requirements.

Figure 8 shows the performance of Orion internal and external clients along with other file systems. For filebench workloads, Orion outperforms Gluster and Ceph by a large margin (up to 40×). We observe that the high synchronization cost in Ceph and Gluster makes them only suitable for workloads with high queue depths, which are less likely on NVMM because media access latency is low. For MongoDB, Orion outperforms other distributed file systems by a smaller margin because of the less intensive I/O activities.

Although Ext4/iSER does not support sharing, file system synchronization (e.g., `fsync()`) is expensive because it flushes the block queue over RDMA. Orion outperforms Ext4/iSER in most workloads, especially for those that require frequent synchronization, such as varmail (with 4.5× higher throughput). For webserver, a read-intensive workload, Ext4/iSER performs better than local Ext4-DAX and Orion because it uses the buffer cache to hold most of the data and does not flush writes to storage.

Orion achieves an average of 73% of NOVA's throughput. It also outperforms Ext4-DAX on metadata and I/O intensive workloads such as varmail and filebench. For Webserver, a read-intensive workload, Orion is slower because it needs to communicate with the MDS.

The performance gap between external clients and internal clients is small in our experiments, especially for write requests. This is because our hardware does not support the optimized cache flush instructions that Intel plans to add in the near future [51]. Internal clients persist local writes using `clflush` or non-temporal memory copy with fences; both of which are expensive [76].

6.4 Metadata and Data Replication

Figure 9 shows the performance impact of metadata and data replication. We compare the performance of a single internal client (IC), a single external client (EC), an internal client with one and two replicas (IC+1R, +2R), and an internal client with two replicas and MDS replication (+2R+M). For a 4 KB write, it takes an internal client 12.1 μs to complete with our strongest reliability scheme (+2R+M), which is 2.1× longer than internal client and 1.5× longer than an external client. For filebench workloads, overall performance decreases by between 2.3% and 15.4%.

6.5 MDS Scalability

Orion uses a single MDS with a read-only mirror to avoid the overhead of synchronizing metadata updates across multiple nodes. However, using a single MDS raises scalability concerns. In this section, we run an MDS paired with 8 internal clients to evaluate the system under heavy metadata traffic.

We measure MDS performance scalability by stressing it with different types of requests: client initiated inbound RDMA reads, log commits, and RPCs. Figure 10 measures throughput for the MDS handling concurrent requests from different numbers of clients. For inbound RDMA reads (a), each client posts RDMA reads for an 8-byte field, simulating reading the log tail pointers of inodes. In (b) the client sends 64-byte log commits spread across 10,000 inodes. In (c) the clients send 64-byte RPCs and the MDS responds with 32-byte acknowledgments. Each RPC targets one of the 10,000 inodes. Finally, in (d) we use FIO to perform 4 KB random writes from each client to private file.

Inbound RDMA reads have the best performance and scale well: with eight clients, the MDS performs 13.8 M RDMA reads per second – 7.2× the single-client performance. For log commits, peak throughput is 2.5 M operations per second with eight clients – 4.1× the performance for a single client. Log commit scalability is lower because the MDS must perform the log append in software. The MDS can perform 772 K RPCs per second with seven clients (6.2× more than a single). Adding an eighth does not improve performance due to contention among threads polling CQEs and threads handling RPCs. The FIO write test shows good scalability – 7.9× improvement with eight threads. Orion matches NOVA performance with two clients and out-performs NOVA by

Figure 10: **Orion metadata scalability for MDS metadata operations and FIO 4K randwrite** Orion exhibits good scalability with rising node counts for inbound 8 B RDMA reads (a), 64 B log commits (b), RPCs (c), and random writes (d).

4.1× on eight clients.

Orion is expected to have good scaling under these conditions. Similar to other RDMA based studies, Orion is suitable to be deployed on networks with high bisectional bandwidth and predictable end-to-end latency, such as rack-scale computers [17, 39]. In these scenarios, the single MDS design is not a bottleneck in terms of NVMM storage, CPU utilization, or networking utilization. Orion metadata consumes less than 3% space compared to actual file data in our experiments. Additionally, metadata communication is written in tight routines running on dedicated cores, where most of the messages fit within two cache lines. Previous works [7,40] show similar designs can achieve high throughput with a single server.

In contrast, several existing distributed file systems [8, 19, 30, 69] target data-center scale applications, and use mechanisms designed for these conditions. In general, Orion's design is orthogonal to the mechanisms used in these systems, such as client side hashing [19] and partitioning [69], which could be integrated into Orion as future work. On the other hand, we expect there may be other scalability issues such as RDMA connection management and RNIC resource contention that need to be addressed to allow further scaling for Orion. We leave this exploration as future work.

7 Related work

Orion combines ideas from NVMM file systems, distributed file systems, distributed shared memory, user level file systems with trusted services, and recent work on how to best utilize RDMA. Below, we place Orion in context relative to key related work in each of these areas.

NVMM file systems Emerging NVMM technologies have inspired researchers to build a menagerie NVMM-specific file systems. Orion extends many ideas and implementation details from NOVA [73,74] to the distributed domain, especially in how Orion stores and updates metadata. Orion also relies on key insights developed in earlier systems [15,21,25,68,70,72].

Distributed file systems There are two common ways to provide distributed file accesses: the first is to deploy a Clustered File System (CFS) [8,11,20,26,31,54] running on block devices exposed via a storage area network (SAN) protocol like iSCSI [53], Fiber Channel or NVMe Over Fabrics [18]. They use RDMA to accelerate the data path [12, 18, 61] and they can accelerate data transfers using zero-copy techniques while preserving the block-based interface.

The second is to build a Distributed File System (DFS) [8, 9, 19, 30, 37, 38, 47, 57, 57, 69] that uses local file systems running on a set of servers to create a single, shared file system image. They consist of servers acting in dedicated roles and communicating using customized protocols to store metadata and data. Some distributed file systems use RDMA as a drop-in replacement of existing networking protocols [10,18,63,71] while preserving the local file system logic.

Their diversity reflects the many competing design goals they target. They vary in the interfaces they provide, the consistency guarantees they make, and the applications and deployment scenarios they target. However, these systems target hard drives and SSDs and include optimizations such as queuing striping and DRAM caching. Orion adds to this diversity by rethinking how a full-featured distributed file system can fully exploit the characteristics of NVMM and RDMA.

Octopus [41] is a distributed file system built for RDMA and NVMM. Compared to Orion, its design has several limitations. First, Octopus assumes a simplifed file system model and uses a static hash table to organize file system metadata and data, which preventing it from running complex workloads. Second, Octopus uses client-side partitioning. This design restricts access locality: as the number of peers increases, common file system tasks such as traversing a directory become expensive. Orion migrates data to local NVMM to improve locality. Finally, Octopus does not provide provisions for replication of either data or metadata, so it cannot tolerate node failures.

Trusted file system services Another research trend is to decouple file system control plane and data plane, and build userspace file systems [36, 48, 68] with trusted services to reduce the number of syscalls. Orion MDS plays a similar role as the trusted service. However, Orion heavily leverages

kernel file system mechanisms, as well as the linear addressing of kernel virtual addresses and DMA addresses. In order to support DAX accesses, extending a uerspace file system to a distributed setting must deal with issues such as large page tables, sharing and protection across processes and page faults, which are all not RDMA friendly.

Distributed shared memory There has been extensive research on distributed shared memory (DSM) systems [44, 49, 50], and several of them have considered the problem of distributed, shared persistent memory [29, 55, 56]. DSM systems expose a simpler interface than a file system, so the designers have made more aggressive optimizations in many cases. However, that makes adapting existing software to use them more challenging.

Hotpot [55] is a distributed shared persistent memory system that allows applications to commit fine-grained objects on memory mapped NVMM files. It is built on a customized interface, requiring application modification. Hotpot uses a multi-stage commit protocol for consistency, while Orion uses client-side updates to ensure file system consistency.

Mojim [76] provides fine-grained replication on NVMM, and Orion uses this technique to implement metadata replication.

RDMA-optimized applications Many existing works explore how RDMA can accelerate data center applications, such as key-value stores [23, 33, 43], distributed transaction systems [13, 24, 34], distributed memory allocators [5, 23, 66, 67] and RPC implementations [35, 58]. There are several projects using RDMA protocols targeting to accelerate existing distributed storages [3, 32] or work as a middle layer [1, 2]. Orion differs from these systems in that it handles network requests directly within file systems routines, and uses RDMA to fully exploit NVMM's byte-addressability.

8 Conclusion

We have described and implemented Orion, a file system for distributed NVMM and RDMA networks. By combining file system functions and network operations into a single layer, Orion provides low latency metadata accesses and allows clients to access their local NVMMs directly while accepting remote accesses. Our evaluation shows that Orion outperforms existing NVMM file systems by a wide margin, and it scales well over multiple clients on parallel workloads.

Acknowledgments

The authors would like to thank our shepherd Florentina Popovici, and the anonymous reviewers for their insightful comments and suggestions. We thank members of Non-Volatile Systems Lab for their input. The work described in this paper is supported by a gift from Huawei.

References

[1] Accelio - Open-Source IO, Message, and RPC Acceleration Library. https://github.com/accelio/accelio.

[2] Alluxio - Open Source Memory Speed Virtual Distributed Storage. https://www.alluxio.org/.

[3] Crail: A fast multi-tiered distributed direct access file system. https://github.com/zrlio/crail.

[4] MongoDB Community. https://www.mongodb.com/community.

[5] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novaković, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote regions: a simple abstraction for remote memory. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 775–787, Boston, MA, 2018.

[6] Jens Axboe. Flexible I/O Tester. https://github.com/axboe/fio.

[7] Mahesh Balakrishnan, Dahlia Malkhi, John D Davis, Vijayan Prabhakaran, Michael Wei, and Ted Wobber. Corfu: A distributed shared log. *ACM Transactions on Computer Systems (TOCS)*, 31(4):10, 2013.

[8] Peter J Braam. The Lustre storage architecture, 2004.

[9] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011.

[10] Brent Callaghan, Theresa Lingutla-Raj, Alex Chiu, Peter Staubach, and Omer Asad. NFS over RDMA. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*, pages 196–208. ACM, 2003.

[11] Philip H Carns, Walter B Ligon III, Robert B Ross, Rajeev Thakur, et al. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th annual Linux showcase and conference*, pages 391–430, 2000.

[12] Mallikarjun Chadalapaka, Hemal Shah, Uri Elzur, Patricia Thaler, and Michael Ko. A Study of iSCSI Extensions for RDMA (iSER). In *Proceedings of the ACM SIGCOMM Workshop on Network-I/O Convergence: Experience, Lessons, Implications*, NICELI '03, pages 209–219. ACM, 2003.

[13] Yanzhe Chen, Xingda Wei, Jiaxin Shi, Rong Chen, and Haibo Chen. Fast and general distributed transactions using RDMA and HTM. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 26. ACM, 2016.

[14] Dave Chinner. xfs: DAX support. https://lwn.net/Articles/635514/. Accessed 2019-01-05.

[15] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, pages 133–146, 2009.

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

[17] Paolo Costa, Hitesh Ballani, Kaveh Razavi, and Ian Kash. R2c2: A network stack for rack-scale computers. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 551–564. ACM, 2015.

[18] Patrice Couvert. High speed IO processor for NVMe over fabric (NVMeoF). *Flash Memory Summit*, 2016.

[19] Alex Davies and Alessandro Orsaria. Scale out with GlusterFS. *Linux Journal*, 2013(235):1, 2013.

[20] Matt DeBergalis, Peter Corbett, Steve Kleiman, Arthur Lent, Dave Noveck, Tom Talpey, and Mark Wittle. The Direct Access File System. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, FAST '03, pages 175–188, Berkeley, CA, USA, 2003. USENIX Association.

[21] Mingkai Dong and Haibo Chen. Soft updates made simple and fast on non-volatile memory. In *2017 USENIX Annual Technical Conference (USENIX ATC 17). USENIX Association, Santa Clara, CA*, pages 719–731, 2017.

[22] Chet Douglas. RDMA with PMEM, Software mechanisms for enabling access to remote persistent memory. http://www.snia.org/sites/default/files/SDC15_presentations/persistant_mem/ChetDouglas_RDMA_with_PM.pdf. Accessed 2019-01-05.

[23] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. FaRM: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pages 401–414, 2014.

[24] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 54–70. ACM, 2015.

[25] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. System Software for Persistent Memory. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 15:1–15:15. ACM, 2014.

[26] Chandramohan A Thekkath Edward K. Lee. Petal: Distributed Virtual Disks. *ASPLOS VII*, pages 1–9, 1996.

[27] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, and G. Hush. A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 338–339, Feb 2014.

[28] Mike Ferron-Jones. A New Breakthrough in Persistent Memory Gets Its First Public Demo. https://itpeernetwork.intel.com/new-breakthrough-persistent-memory-first-public-demo/. Accessed 2019-01-05.

[29] João Garcia, Paulo Ferreira, and Paulo Guedes. The PerDiS FS: A transactional file system for a distributed persistent store. In *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*, pages 189–194. ACM, 1998.

[30] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43. ACM, 2003.

[31] Garth A. Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang, Eugene M. Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka. File server scaling with network-attached secure disks. In *ACM SIGMETRICS Performance Evaluation Review*, volume 25, pages 272–284. ACM, 1997.

[32] Nusrat Sharmin Islam, Md Wasi-ur Rahman, Xiaoyi Lu, and Dhabaleswar K Panda. High performance design for HDFS with byte-addressability of NVM and RDMA. In *Proceedings of the 2016 International Conference on Supercomputing*, page 8. ACM, 2016.

[33] Anuj Kalia, Michael Kaminsky, and David G Andersen. Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 295–306. ACM, 2014.

[34] Anuj Kalia, Michael Kaminsky, and David G Andersen. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *OSDI*, volume 16, pages 185–201, 2016.

[35] Anuj Kalia Michael Kaminsky and David G Andersen. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference*, page 437, 2016.

[36] Sudarsun Kannan, Andrea C Arpaci-Dusseau, Remzi H Arpaci-Dusseau, Yuangang Wang, Jun Xu, and Gopinath Palani. Designing a true direct-access file system with DevFS. In *16th USENIX Conference on File and Storage Technologies*, page 241, 2018.

[37] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. 2000.

[38] Edward K. Lee, Chandramohan A. Thekkath, and Timothy Mann. Frangipani: A Scalable Distributed File System. In *Proceedings of the sixteenth ACM symposium on Operating systems principles - SOSP '97*, volume 31, pages 224–237. ACM Press, 1997.

[39] Sergey Legtchenko, Nicholas Chen, Daniel Cletheroe, Antony IT Rowstron, Hugh Williams, and Xiaohan Zhao. Xfabric: A reconfigurable in-rack network for rack-scale computers. In *NSDI*, volume 16, pages 15–29, 2016.

[40] Sheng Li, Hyeontaek Lim, Victor W Lee, Jung Ho Ahn, Anuj Kalia, Michael Kaminsky, David G Andersen, O Seongil, Sukhan Lee, and Pradeep Dubey. Architecting to achieve a billion requests per second throughput on a single key-value store server platform. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 476–488. ACM, 2015.

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

[42] Mellanox. Mellanox OFED for Linux User Manual, 2017.

[43] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *USENIX Annual Technical Conference*, pages 103–114, 2013.

[44] Bill Nitzberg and Virginia Lo. Distributed shared memory: A survey of issues and algorithms. *Computer*, 24(8):52–60, 1991.

[45] H. Noguchi, K. Ikegami, K. Kushida, K. Abe, S. Itai, S. Takaya, N. Shimomura, J. Ito, A. Kawasumi, H. Hara, and S. Fujita. A 3.3ns-access-time 71.2uW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture. In *Solid-State Circuits Conference (ISSCC), 2015 IEEE International*, pages 1–3, Feb 2015.

[46] Colby Parkinson. NVDIMM-N: Where are we now? https://www.micron.com/about/blogs/2017/august/nvdimm-n-where-are-we-now. Accessed 2019-01-05.

[47] Swapnil Patil and Garth A Gibson. Scale and Concurrency of GIGA+: File System Directories with Millions of Files. In *FAST*, volume 11, pages 13–13, 2011.

[48] Simon Peter, Jialin Li, Irene Zhang, Dan RK Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The operating system is the control plane. *ACM Transactions on Computer Systems (TOCS)*, 33(4):11, 2016.

[49] Jelica Protic, Milo Tomasevic, and Veljko Milutinovic. A survey of distributed shared memory systems. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 1, pages 74–84. IEEE, 1995.

[50] Jelica Protic, Milo Tomasevic, and Veljko Milutinovic. Distributed shared memory: Concepts and systems. *IEEE Parallel & Distributed Technology: Systems & Applications*, 4(2):63–71, 1996.

[51] Andy Rudoff. Processor Support for NVM Programming. http://www.snia.org/sites/default/files/AndyRudoff_Processor_Support_NVM.pdf. Accessed 2019-01-05.

[52] Arthur Sainio. NVDIMM: Changes are Here So Whats Next. *In-Memory Computing Summit*, 2016.

[53] Julian Satran, Kalman Meth, C Sapuntzakis, M Chadalapaka, and E Zeidner. Internet small computer systems interface (iSCSI), 2004.

[54] Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. *Proceedings of the First USENIX Conference on File and Storage Technologies*, pages 231–244, 2002.

[55] Yizhou Shan, Shin-Yeh Tsai, and Yiying Zhang. Distributed shared persistent memory. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 323–337. ACM, 2017.

[56] Marc Shapiro and Paulo Ferreira. Larchant-RDOSS: a distributed shared persistent memory and its garbage collector. In *International Workshop on Distributed Algorithms*, pages 198–214. Springer, 1995.

[57] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. Ieee, 2010.

[58] Patrick Stuedi, Animesh Trivedi, Bernard Metzler, and Jonas Pfefferle. DaRPC: Data center RPC. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–13. ACM, 2014.

[59] Maomeng Su, Mingxing Zhang, Kang Chen, Zhenyu Guo, and Yongwei Wu. RFP: When RPC is Faster Than Server-Bypass with RDMA. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 1–15. ACM, 2017.

[60] Talpey and Pinkerton. RDMA Durable Write Commit. https://tools.ietf.org/html/draft-talpey-rdma-commit-00. Accessed 2019-01-05.

[61] T Talpey and G Kamer. High Performance File Serving With SMB3 and RDMA via SMB Direct. In *Storage Developers Conference*, 2012.

[62] Haodong Tang, Jian Zhang, and Fred Zhang. Accelerating Ceph with RDMA and NVMeoF. In *14th Annual OpenFabrics Alliance (OFA) Workshop*, 2018.

[63] Wittawat Tantisiriroj, Seung Woo Son, Swapnil Patil, Samuel J Lang, Garth Gibson, and Robert B Ross. On the duality of data-intensive file system design: reconciling HDFS and PVFS. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 67. ACM, 2011.

[64] Vasily Tarasov, Erez Zadok, and Spencer Shepler. Filebench: A flexible framework for file system benchmarking. *USENIX; login*, 41, 2016.

[65] Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. Pin-down cache: A virtual memory management technique for zero-copy communication. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*, pages 308–314. IEEE, 1998.

[66] Animesh Trivedi, Patrick Stuedi, Bernard Metzler, Clemens Lutz, Martin Schmatz, and Thomas R Gross. Rstore: A direct-access DRAM-based data store. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 674–685. IEEE, 2015.

[67] Shin-Yeh Tsai and Yiying Zhang. LITE: Kernel RDMA support for datacenter applications. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 306–324. ACM, 2017.

[68] Haris Volos, Sanketh Nalli, Sankarlingam Panneerselvam, Venkatanathan Varadarajan, Prashant Saxena, and Michael M Swift. Aerie: Flexible file-system interfaces to storage-class memory. In *Proceedings of the Ninth European Conference on Computer Systems*, page 14. ACM, 2014.

[69] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[70] Matthew Wilcox. Add support for NV-DIMMs to Ext4. https://lwn.net/Articles/613384/. Accessed 2019-01-05.

[71] J. Wu, P. Wyckoff, and Dhabaleswar Panda. PVFS over InfiniBand: Design and performance evaluation. In *2003 International Conference on Parallel Processing, 2003. Proceedings.*, pages 125–132. IEEE, 2003.

[72] Xiaojian Wu and A. L. Narasimha Reddy. SCMFS: A File System for Storage Class Memory. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 39:1–39:11. ACM, 2011.

[73] Jian Xu and Steven Swanson. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 323–338, Santa Clara, CA, February 2016. USENIX Association.

[74] Jian Xu, Lu Zhang, Amirsaman Memaripour, Akshatha Gangadharaiah, Amit Borase, Tamires Brito Da Silva, Steven Swanson, and Andy Rudoff. NOVA-Fortis: A Fault-Tolerant Non-Volatile Main Memory File System. In *26th Symposium on Operating Systems Principles (SOSP '17)*, pages 478–496, 2017.

[75] Yiying Zhang and Steven Swanson. A study of application performance with non-volatile main memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2015.

[76] Yiying Zhang, Jian Yang, Amirsaman Memaripour, and Steven Swanson. Mojim: A reliable and highly-available non-volatile memory system. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 3–18. ACM, 2015.