Chasing the Rabbit: A Systematic Exploration of Software-Defined HPC Storage

Hariharan Devarajan[1], Brian Behlendorf[1], Blake Devcich[2], Dean Roehrich[2], Matt Richerson[2],
Tony Floeder[2], Brent Petit[2], Don Boyd[2], Trace Hamada[2], Tim McCree[2], Gian-Carlo Defazio[1], James Corbett[1],
Elsa Gonsiorowski[1], Cameron Stanavige[1], Nikoli Dryden[1], Roger Pearce[1],
Kathryn Mohror[1], and Bronis R. de Supinski[1]
[1]Lawrence Livermore National Laboratory [2]Hewlett Packard Enterprise
hariharandev1@llnl.gov, behlendorf1@llnl.gov, blake.devcich@hpe.com, dean.roehrich@hpe.com, matthew.richerson@hpe.com,
anthony.floeder@hpe.com, brent.petit@hpe.com, don.boyd@hpe.com, trace.hamada@hpe.com, tim.mccree@hpe.com,
defazio1@llnl.gov, corbett8@llnl.gov, gonsiorowski1@llnl.gov, stanavige1@llnl.gov,
dryden1@llnl.gov, pearce7@llnl.gov, mohror1@llnl.gov, desupinski1@llnl.gov

***Abstract*—Storage accelerators are essential for efficient data exchange in modern scientific workflows, which increasingly integrate simulations, analytics, and AI. The Rabbit Storage System, recently deployed on El Capitan, introduces a software-defined model for dynamic, workload-aware I/O, but its capabilities remain understudied. We present the first systematic study of Rabbits, evaluating configurability and performance using IO500 and DLIO benchmarks. Our findings show that: (1) software-defined file system deployment (XFS, GFS2, Lustre) achieves up to 3.12× higher bandwidth than global Lustre for large sequential accesses; and (2) workload-centric Lustre tuning improves performance by 4.6×, reaching 26 GB/s per Rabbit node. With 16 Rabbit nodes, end-to-end benchmarks demonstrate strong potential, achieving 413GB/s in IO500 reads, up to 7× faster for metadata operations, and 175GB/s in DLIO with Cosmoflow, establishing the Rabbits as a versatile storage accelerator that enables workload-aware optimization and high utilization across diverse HPC and AI workloads.**

I. Introduction

Modern scientific workloads are increasingly data-intensive, exchanging vast amounts of data during execution [1]. These include high-performance simulations, large-scale data analytics, and AI applications across diverse domains [2]–[5] such as cosmology, particle physics, and biology. A significant challenge faced by these workloads is the I/O bottleneck, where a substantial portion of runtime is spent on data movement. For example [6], the Montage astronomy workflow spends 24% of its runtime on I/O, the CosmoFlow deep learning model 18%, and the 1000 Genomes bioinformatics workflow as much as 41%. To address this, modern supercomputers are equipped with storage accelerators [7]–[9], such as node-local storage on Frontier or DataWarp on Cori, that act as burst buffers to alleviate I/O overhead. These accelerators have demonstrated up to 3× reductions in I/O bottlenecks, significantly improving the overall performance of data-intensive workloads [10]–[12].

Lawrence Livermore National Laboratory (LLNL) has deployed the *Rabbit Storage System*, a novel software-defined storage accelerator, on its top supercomputers El Capitan [13] (ranked #1) and Tuolumne [14] (#12) on the TOP500 list (as of November 2025). Rabbit enables job-specific, workload-aware I/O acceleration by dynamically configuring hardware, software deployment, and runtime settings via Kubernetes [15]. This facilitates a non-uniform distribution of storage accelerators across jobs, significantly improving utilization. This approach overcomes inefficiencies of static node-local accelerators, which often go underutilized despite offering high bandwidth and low latency [16], [17]. Rabbit supports both predefined file systems and user-defined software stacks through container technologies, creating a flexible, software-defined storage platform that has the potential to improve performance for data-intensive workloads and improves storage accelerator utilization.

Despite their potential, we need to quantify their capabilities and performance characteristics to effectively support data-intensive scientific workloads. This challenge is multifaceted: (1) **Novel hardware architecture**: Unlike traditional burst buffers [7], [8], [18] that are either node-local via PCIe or remotely connected via high-speed networks, a Rabbit node is connected to a group of 16 compute nodes through both PCIe switches and network links. The performance depends on the communication path and file system used, making characterization complex. (2) **Complexity of the software layer**: The same hardware can be accessed through multiple storage solutions, including XFS [19], GFS2 [20], and Lustre [21], each with distinct data locality semantics, tuning options, and I/O behavior. Selecting the right configuration is crucial to maximizing performance for different workloads. (3) **Dynamic provisioning of hardware**: Rabbit storage resources can be allocated dynamically per job or persist across jobs, enabling flexible usage patterns such as those required by long-running workflows like MuMMI [22]. This introduces varying performance profiles due to heterogeneous and time-varying hardware contexts. These challenges underscore the need for a systematic and comprehensive evaluation of the Rabbit Storage System to understand its potential for accelerating scientific discovery.

To address the challenges introduced by the novel architecture, complex software stack, and dynamic

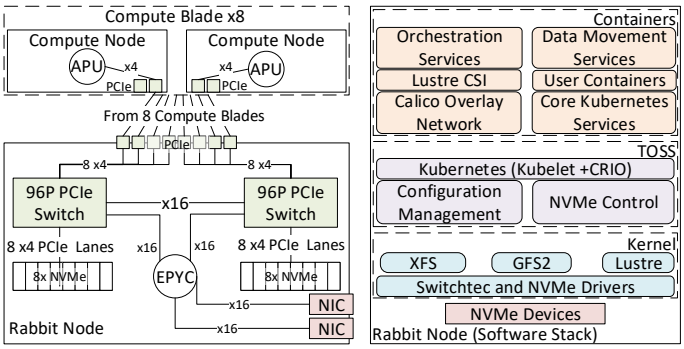


Fig. 1. The hardware architecture (left) shows the PCIe-base connection of compute nodes to a Rabbit node. The software stack (right) shows the containerized software used to manage deployment on Rabbit nodes.

provisioning of the Rabbit Storage System, we present a systematic study to investigate its capabilities and performance characteristics. Our study explores three key dimensions. First, to jointly address the challenges posed by the *novel hardware architecture* and the *complexity of the software layer*, we evaluate the ability to configure and deploy different file systems namely, XFS, GFS2, and Lustre, with their specific hardware connections. This allows us to analyze how combinations of hardware topology (e.g., PCIe and Slingshot 11 network [23]) and file system characteristics influence performance across workloads. Second, to address the *dynamic nature of the hardware*, we study the system's capability to deploy a file system to different hardware configurations, enabling flexible provisioning models that support both transient job-level and persistent campaign-level data access. To quantify the practical utility of these capabilities, we evaluate Rabbit's performance using the popular community benchmarks like IO500 [24] and DLIO benchmarks [25]. Together, this investigation lays the foundation for guiding users to efficiently utilize of Rabbit systems in production HPC environments. The key contributions of this work are:

1) We conduct the first systematic study of the Rabbit Storage System's capabilities, exploring its hardware configurations and software-defined features.
2) We demonstrate Rabbit's dynamic configurability, showing how the same hardware can expose XFS, GFS2, Lustre, or shared block semantics to match workload needs.
3) We evaluate Rabbit using IO500 and DLIO benchmarks to assess performance across diverse configurations and workload types.

II. Rabbit Storage System

The Rabbit Storage System is a software-defined, workload-aware storage accelerator designed for dynamic adaptation to diverse HPC and AI workloads (Figure 1). It achieves this flexibility through a tightly integrated hardware [15] (shown on left) and software architecture [26] (shown on right) that supports on-demand configuration and provisioning.

Figure 1 shows the hardware architecture. The Rabbit architecture connects one Rabbit node to eight compute blades. Each blade contains two compute nodes, and every node's AMD APU is connected to the Rabbit node via PCIe links (depicted in green). The two switches are interconnected via x16 PCIe Gen 4 links to ensure full bandwidth. Each switch connects directly to multiple NVMe devices using x4 lanes, enabling high-throughput access. Additionally, two PCIe links on each switch connect to the Rabbit node's EPYC processor, system memory, and NIC, which interfaces with the HPE Slingshot network. This topology ensures high storage bandwidth and low latency between compute and storage.

Figure 1 (right) illustrates the software stack, composed of three main layers. The *container layer* enables dynamic service management on Rabbit nodes, hosting components such as storage orchestration services, data movement utilities (e.g., stage-in/out), user-defined containers, overlay networks, and core Kubernetes services [27] for lifecycle management. Beneath it, the *TOSS operating system layer* includes pre-installed software such as Kubernetes (with `kubelet` and `CRIO`), configuration management tools. Finally, the kernel layer contains file system modules for XFS, GFS2, and Lustre, and core operating system drivers.

III. Methodology

Our methodology is designed to comprehensively evaluate the capabilities and performance of the Rabbit Storage System under realistic HPC and AI workloads. We first describe the testbed and then define the experimental scenarios that cover the different capabilities and performance characteristics of Rabbit Storage system.

A. Testbed

1) Hardware Platform.: Our experiments were conducted on the Tuolumne system at Lawrence Livermore National Laboratory [14], which comprises 1152 nodes; we utilized up to 256 nodes for our evaluation. Each compute node is equipped with 4 AMD MI300A APUs [28]. Each MI300A APU integrates 24 AMD Zen 4 x86 CPU cores, 228 AMD CDNA 3 GPU compute units, and 128 GB of unified HBM3 memory that presents a single shared address space to both CPU and GPU. Each node has a 4×200 Gb/s HPE Slingshot 11 NIC. As a baseline for performance comparison, the system provides a Lustre file system, referred to as Global Lustre, which serves all compute nodes. Each Rabbit node contains 16-2TB NVMe drives, of which we configured 20 TB space. These drives deliver a peak sequential read-write bandwidth of 3.5 GB/s. Thus, a single Rabbit node offers an aggregate sequential bandwidth of up to 56 GB/s. Each Rabbit node is also equipped with 256 GB of memory, 64-core AMD EPYC 7713 processor, and 2×200 Gb/s Slingshot NICs.

2) Software Configuration.: All experiments were conducted on the Tuolumne system running the TOSS 4 operating system. The software environment was configured with AMD compiler version `6.2.4`, Cray MPICH `8.1.25` for MPI support, and CMake `3.29.2` for build configuration. The ROCm stack version `6.2.4` was used to support GPU-accelerated workloads on the AMD MI300A APUs.

3) Workloads.: We evaluated the Rabbit Storage System using three classes of I/O workloads: IOR [29], community IO500 benchmark [24], and DLIO Benchmark [25]. The IOR benchmark was configured with 4 MB read and write operations per process and a total block size of 16 GB per process. The benchmark was executed using the file-per-process (FPP) strategy over the POSIX interface, with a sequential access pattern. For IO500, we modified the configuration with direct I/O, 1GB block size, and 5120 segments. Finally, in DLIO Benchmark, we tested the BERT [30] and CosmoFlow [2] models implemented in TensorFlow and training on TFRecord datasets. Each test executed ten times to capture performance variability.

4) Rabbit Configurations.: The Rabbit Storage System supports multiple file system configurations. For local access over PCIe, both `XFS` and `GFS2` file systems are supported, where a namespace is created from each of 16 NVMe drives and mounted exclusively for XFS and concurrently for GFS2 on the compute nodes. For distributed access over the Slingshot network, `Lustre` is deployed with one object storage target (OST) across the 16 NVMe drives. The Lustre configuration is highly tunable at server and client side. Furthermore, Lustre file systems can be deployed in two modes: as a *transient* file system scoped to the lifetime of a single job, or as a *persistent* file system accessible across multiple jobs—an essential capability for long-running scientific campaigns [31].

B. Scenario Exploration

To understand the capabilities and performance characteristics of the Rabbit Storage System, we design scenarios that evaluate runtime behavior under representative workloads while highlighting adaptivity.

1) Capability to Deploy Different File Systems.: The Rabbit Storage System supports flexible deployment of multiple file systems, making it possible to tailor storage to specific workload requirements. In this scenario, we evaluate the system's ability to deploy different file systems, namely, XFS, GFS2, and transient Lustre. Importantly, the Rabbit architecture also allows deploying multiple file systems concurrently on the same Rabbit node, enabling workload-centric access pattern optimization within a single workflow. By running IOR with a consistent configuration across these file systems, we analyze their relative performance under sequential POSIX I/O, revealing trade-offs in throughput, latency, and parallel scalability. This evaluation provides insight into how different file systems can be effectively matched to workload requirements in software-defined storage environments.

2) Capability to Share Block Devices.: In this scenario, we explore exposing the 16 NVMe drives on each Rabbit node to all compute nodes in multiple configurations. First, we use GFS2, where each compute node mounts the same device through a GFS2 file system; GFS2 uses a global lock manager to maintain coherency between compute nodes. Second, we deploy a read-only XFS, where data is staged into the file system once and then mounted on all compute nodes as read-only XFS. Third, we explore a raw block device exported via device mapper to all compute nodes; in this case, the device mapper aggregates the 16 NVMe drives into a single logical block device that is presented uniformly to every compute node. Finally, we expose all 16 raw NVMe devices to each compute node. On these devices, we test the capability to read and write blocks from multiple compute nodes. For device-level access, we use POSIX locks and direct I/O to coordinate cross-node access to the block device. This exploration enables new hybrid architectures for sharing data and block devices across compute nodes, a capability unique to the Rabbit architecture.

3) Capability to Deploy Different Lustre Configurations.: In this scenario, we explore the flexibility of the Rabbit Storage System in configuring Lustre with a wide range of tunable parameters and assess how these configurations impact I/O performance using the IOR benchmark. Lustre's extensive configuration space allows tuning for various workload characteristics across four major dimensions [32]. *Resilience* features such as `raidz` and checksumming affect data protection and redundancy, while *space management* features like compression optimize capacity utilization. *Layout* parameters, including `record-size` and `raidz` level, influence how data is physically organized across drives, impacting access granularity and efficiency. Finally, *operation performance* is governed by factors such as the number of object storage targets (OSTs) and metadata targets (MDTs), autotrim feature, client core binding policies, number of inflight RPCs, and RPC page size, all of which affect concurrency and I/O throughput. We use IOR as a controlled workload to systematically explore these Lustre configurations and quantify their impact on sequential POSIX I/O performance. This exploration provides guidelines to fine-tune Rabbit's Lustre performance for a given workload beyond what is possible in fixed storage architectures.

4) Capability to Deploy Different Hardware Configurations.: The Rabbit Storage System enables flexible hardware provisioning modes that allow dynamic adaptation of storage resources to job requirements. In this scenario, we evaluate the *Transient Lustre* and *Persistent Lustre* described in Rabbit configurations. In the transient configuration, a Lustre file system is deployed dynamically at job launch, with Rabbit hardware allocated proportionally to the compute nodes in the job. This model is ideal for scaling storage bandwidth and capacity with the size of the workload. In contrast, the persistent configuration allows system administrators to pre-provision a Lustre file system on a fixed set of Rabbit nodes, independent of the compute job layout. This persistent deployment is particularly useful for workflows requiring high I/O performance with fewer compute resources or for sharing storage across multiple jobs. Using IOR, we compare performance across these two deployment models to assess how hardware configuration flexibility contributes to Rabbit's potential as a software-defined storage accelerator.

5) Performance Characteristics with IO500 and DLIO benchmark.: To assess Rabbit's end-to-end performance under realistic applications, we use two benchmarks: IO500 and DLIO. IO500 evaluates Rabbit's ability to handle mixed I/O workloads through tests such as `ior-easy`

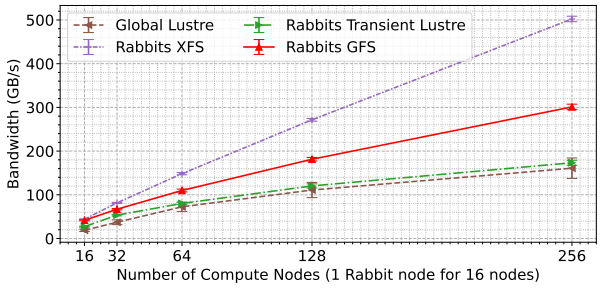


Fig. 2. XFS and GFS2 via PCIe provides better isolation leading to 3.12× and 1.87×, respectively, faster performance than Global Lustre

and `ior-hard` for data throughput, `mdtest-easy` and `mdtest-hard` for metadata performance, and `pfind` for directory traversal. These characterize Rabbit's performance for simulation and analysis workloads compared to global Lustre. Complementing this, DLIO assesses Rabbit's readiness for AI workloads using BERT (compute-heavy) and CosmoFlow (I/O-heavy) models implemented in TensorFlow. These benchmarks test data loading efficiency, scaling behavior, and variability across Rabbit configurations (XFS, transient Lustre, and persistent Lustre), demonstrating Rabbit's capability to deliver high throughput and low-latency I/O patterns for both traditional HPC and AI-driven applications.

IV. Evaluation

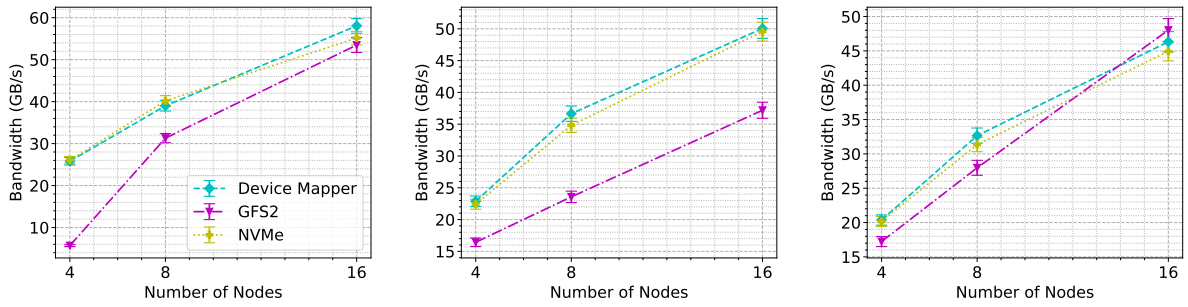
In this section, we examine five scenarios designed to capture Rabbit's performance for diverse configurations. The first four scenarios use IOR to characterize simulation and analytics workloads with controlled sequential I/O patterns. IO500 introduces random I/O and metadata stress testing to stress-test realistic mixed workloads. DLIO represents modern deep learning workloads with data-intensive training pipelines.

A. Deploy Different File Systems

We evaluate the capability of Rabbit Storage to deploy different file systems and compare their performance to the baseline of a Global Lustre file system. As described in the Methodology section, we use the IOR benchmark configured for sequential POSIX I/O and run scaling experiments from 16 to 256 compute nodes. For Rabbit-based configurations, we test three file systems: `XFS`, `GFS2`, and `Transient Lustre`, with each Rabbit node managing 16 NVMe drives and connected to 16 compute nodes. Thus, a 256-node job uses 16 Rabbit nodes. Figure 2 plots bandwidth (in GB/s) on the x-axis against the number of compute nodes on the y-axis.
Observations and Analysis: XFS demonstrates the highest performance, achieving between 65% and 3.12× higher bandwidth compared to the global Lustre baseline. This is expected because XFS uses local PCIe access to NVMe devices, minimizing network hops and metadata contention; the file-per-process pattern aligns with per-file metadata updates. GFS2 also outperforms global Lustre by 1.5× to 1.87×, but remains 5% to 66% slower than XFS due to distributed locking and coherency overheads. Rabbit's Transient Lustre provides 7% to 43% higher bandwidth than global Lustre; its shared-file semantics are less suited for file-per-process workloads, so gains diminish with scale as metadata traffic grows. This is because, GFS2 relies on a global lock manager for concurrency control, which inherently limits scalability relative to XFS. The relative ordering also reflects that XFS avoids cross-node coordination entirely, while GFS2 and Lustre pay for coherency and shared metadata. The gap between XFS and GFS2 is largest at small scales, which is consistent with lock overheads being a larger fraction of total time when concurrency is low. At larger scales, GFS2 benefits from more clients issuing concurrent requests, which helps mask lock latency. We also expect XFS to benefit from cleaner client-side caching paths that are less effective in clustered file systems. Overall, Rabbit-based configurations outperform global Lustre due to proximity to compute nodes and NVMe bandwidth, and the ordering (XFS > GFS2 > Lustre) is expected for sequential POSIX workloads.
Key Takeaway: Rabbits provide versatile hardware with job-defined software for workloads to optimize I/O performance.

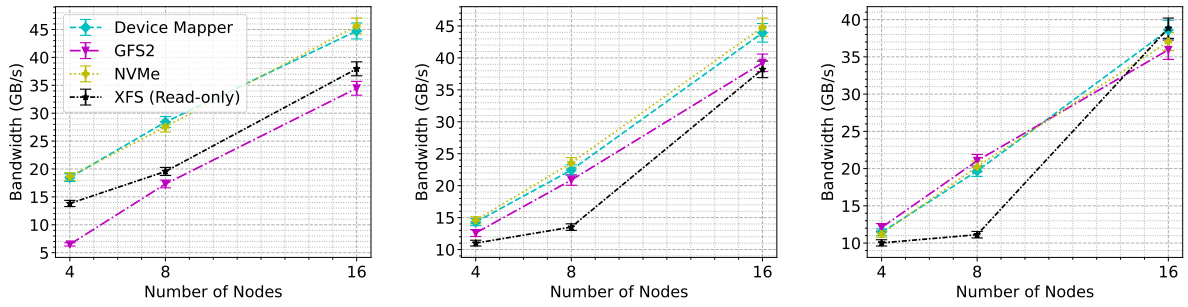
B. Share Block Devices

We evaluate Rabbit's ability to share block devices across multiple compute nodes, enabling workloads to access the same data across multiple compute nodes. We test a producer-consumer pattern where a producer benchmark writes blocks similar to the IOR configuration with varying block sizes (64 KB, 256 KB, and 1 MB), sharing the block devices across 16 compute nodes (4, 8, and 16 nodes). The consumer follows the same pattern. We evaluate three configurations: Device Mapper (presenting raw block devices), GFS2 (a shared cluster file system), and NVMe (direct raw NVMe access). For the consumer case, we also include XFS as a read-only mount. Figure 3 and 4 compares aggregate bandwidth (GB/s) for producer and consumer workloads for different scales.
Observations and Analysis:
Producer Write Performance: Device Mapper and NVMe demonstrate comparable performance across all block sizes and scales (Figure 3). At 64 KB block size, Device Mapper achieves 25.8-58.0 GB/s (4-16 nodes), while NVMe reaches 26.1-55.1 GB/s. The near-linear scaling is expected because both paths bypass file system metadata and use direct block access, so aggregate bandwidth is constrained by NVMe and PCIe throughput. GFS2 is weaker at small scales (5.7 GB/s at 4 nodes) but recovers to 53.5 GB/s at 16 nodes as lock and metadata overheads amortize with scale and block size. At 1 MB, GFS2 nearly matches Device Mapper at 16 nodes (48.0 vs. 46.3 GB/s), consistent with reduced per-operation overhead. This suggests that the clustered file system cost is mostly per-operation rather than throughput-limiting at large I/O sizes, and that large-block producers can tolerate shared semantics with minimal penalty. The device-mapper path also shows that block aggregation does not introduce a noticeable bottleneck for sequential access. The remaining gap at small blocks is dominated by coordination rather than device.
Consumer Read Performance: Device Mapper and NVMe show patterns similar to producer results (Figure 4), with Device Mapper at 18.5-44.7 GB/s and NVMe at 18.7-45.6 GB/s for 64 KB, reflecting stable raw block access. GFS2 lags at



(a) 64k Write (b) 256k Write (c) 1M Write

Fig. 3. Rabbit exposes a shared block device across 16 compute nodes, enabling exploration of new data sharing semantics for applications; Device Mapper is 4.8× faster for writing on small scales.



(a) 64k Read (b) 256k Read (c) 1M Read

Fig. 4. Rabbit enables new storage semantics via a shared block device across 16 compute nodes; Device Mapper is 3.1× faster for reading on small scales.

small scales (6.5 GB/s at 4 nodes) but improves to 34.5 GB/s at 16 nodes as shared lock overheads amortize. XFS read-only mounts scale less (13.8 GB/s at 4 nodes; 38.0 GB/s at 16 nodes), 5-16% below Device Mapper, consistent with file system and metadata overheads even in read-only mode. At larger block sizes, XFS improves but generally remains below raw block access, especially at 256 KB where overheads are not fully amortized. The read-only path still pays for file system traversal and cache management, which explains the persistent gap. These trends suggest raw block access is best for high-throughput read fans, while XFS is acceptable when file semantics are needed. GFS2 becomes more attractive when concurrent readers must share a coherent namespace. The read results also show that synchronization costs dominate more than device bandwidth at small block sizes.

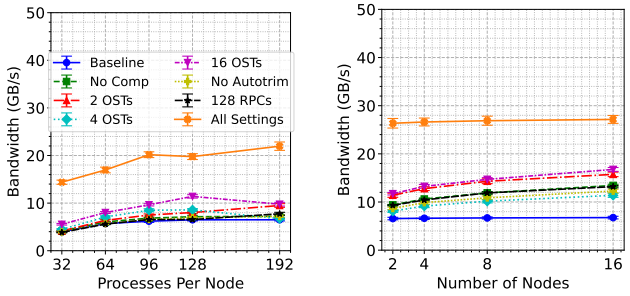
Comparative Analysis Across Configurations: Device Mapper consistently provides the best balance of performance and ease of use, delivering nearly identical throughput to raw NVMe access while offering a logical block abstraction. The marginal gains (<5%) from raw NVMe are expected, as both paths are limited by the same devices and PCIe links, and Device Mapper adds minimal overhead for sequential access. GFS2 performance converges toward Device Mapper at larger scales and block sizes, which is expected when lock and metadata overheads are amortized; it remains the best option when shared file semantics are required. XFS read-only mounts, while simpler to deploy, show performance penalties of 5-19% compared to Device Mapper, particularly for mid-range block sizes where file system overhead is not fully amortized. Scaling with node count is consistent across Device Mapper and NVMe, demonstrating good parallelization, whereas GFS2 exhibits steeper scaling curves at smaller block sizes due to increased coordination overhead, which is expected for clustered file systems.

Key Takeaway: Device Mapper offers the best balance of performance and abstraction, making it optimal for shared storage. The marginal gains (<5%) from raw NVMe don't justify the operational complexity.

C. Deploy Different Lustre Configurations

We evaluate Rabbit's capability to deploy and tune Lustre configurations to meet application-specific requirements, as described in the Methodology section. Using the IOR benchmark configured for 4 MB I/O per process with a total block size of 16 GB, we assess read-write performance across multiple Lustre tuning dimensions in server and client. We show the most impactful cases in Figure 5. For multi-node experiments, we use 192 processes per node. Here, the OST configurations are per Rabbit node and the no compression case is shown as *No Comp* in Figure 5(a). Additionally, we evaluate an "All Settings" configuration generated using a greedy approach, combining all performance-enhancing settings. Figure 5 presents bandwidth (GB/s) on the x-axis and configurations on the y-axis, comparing the baseline Lustre setup against individual optimizations and the fully tuned configuration. Results are shown for two cases: (i) scaling processes per node on a single Rabbit node (Figure 5(a)) and (ii) scaling compute nodes (1-16) performing I/O to a single Rabbit node (Figure 5(b)).

Observations and Analysis: On a single compute node, disabling compression improves performance by 6-18%



(a) One compute node (b) Multiple Compute nodes

Fig. 5. Different workload requirements can be met by tuning Lustre configuration. These effects are compounded when applied together, resulting in a speedup of 4.64× over baseline.

because compression adds CPU overhead without reducing I/O volume. Increasing OST counts results in 5-95% gains, with higher OST counts benefiting larger parallel workloads by increasing parallelism and reducing per-target contention. Disabling `autotrim` yields 5-11% improvement by avoiding background device work, and increasing inflight RPCs helps only at higher process-per-node counts where deeper queues keep OSTs busy. Combining all optimizations delivers 2.9×-4.64× speedup over the baseline. The gains are expected to be workload-sensitive, with small-block or highly concurrent workloads benefiting the most from RPC and OST tuning, while large-block workloads are more bandwidth-limited. This suggests tuning should be workload-aware rather than one-size-fits-all. The single-node results also show that server-side settings can dominate client tuning when the storage target is underutilized. For multi-node scaling (2-16 nodes writing to a single Rabbit), disabling compression improves bandwidth by 1.11×-1.98×, while increasing OST counts provides 1.23×-2.47× gains, with 16 OSTs offering 2-10% better performance than 2 OSTs. Disabling `autotrim` improves bandwidth by 11-81%, and inflight RPC tuning yields 19-95% improvement at larger scales. Combining all optimizations results in 3.37×-4.01× speedups. Maximum bandwidth of 26 GB/s is reached with just 2 compute nodes, beyond which scaling saturates due to fixed NVMe/PCIe bandwidth on a single Rabbit node. This implies adding clients without adding Rabbit nodes mainly increases contention rather than throughput, reinforcing the need to scale storage with compute. The saturation point also highlights the benefit of provisioning additional Rabbit nodes for larger jobs.

Key Takeaways: Rabbit's Lustre enables fine-grained tuning such as disabling compression/RAID/checksum (2×), increasing OST counts (40%), client tuning (2×), that outperforms default Lustre.

D. Deploy Different Hardware Configurations

We assess Rabbit's ability to provide flexible hardware provisioning strategies by comparing two deployment models: *Transient Lustre*, where a job-specific Lustre file system is instantiated and scaled proportionally with compute nodes (maintaining a 16:1 compute-to-Rabbit ratio), and *Persistent Lustre*, where a pre-provisioned Lustre file system spans 16

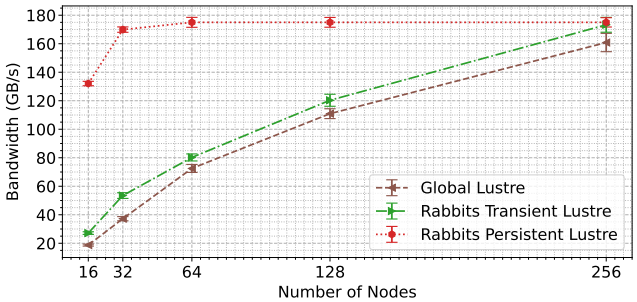


Fig. 6. Transient Lustre scales with compute nodes, while Persistent Lustre provides consistent performance (up to 47× higher than Global Lustre at smaller scales).

Rabbit nodes regardless of the compute node count. Both configurations are evaluated using the same IOR benchmark described before. Figure 6 plots aggregate bandwidth (GB/s) on the y-axis against the number of compute nodes on the x-axis.

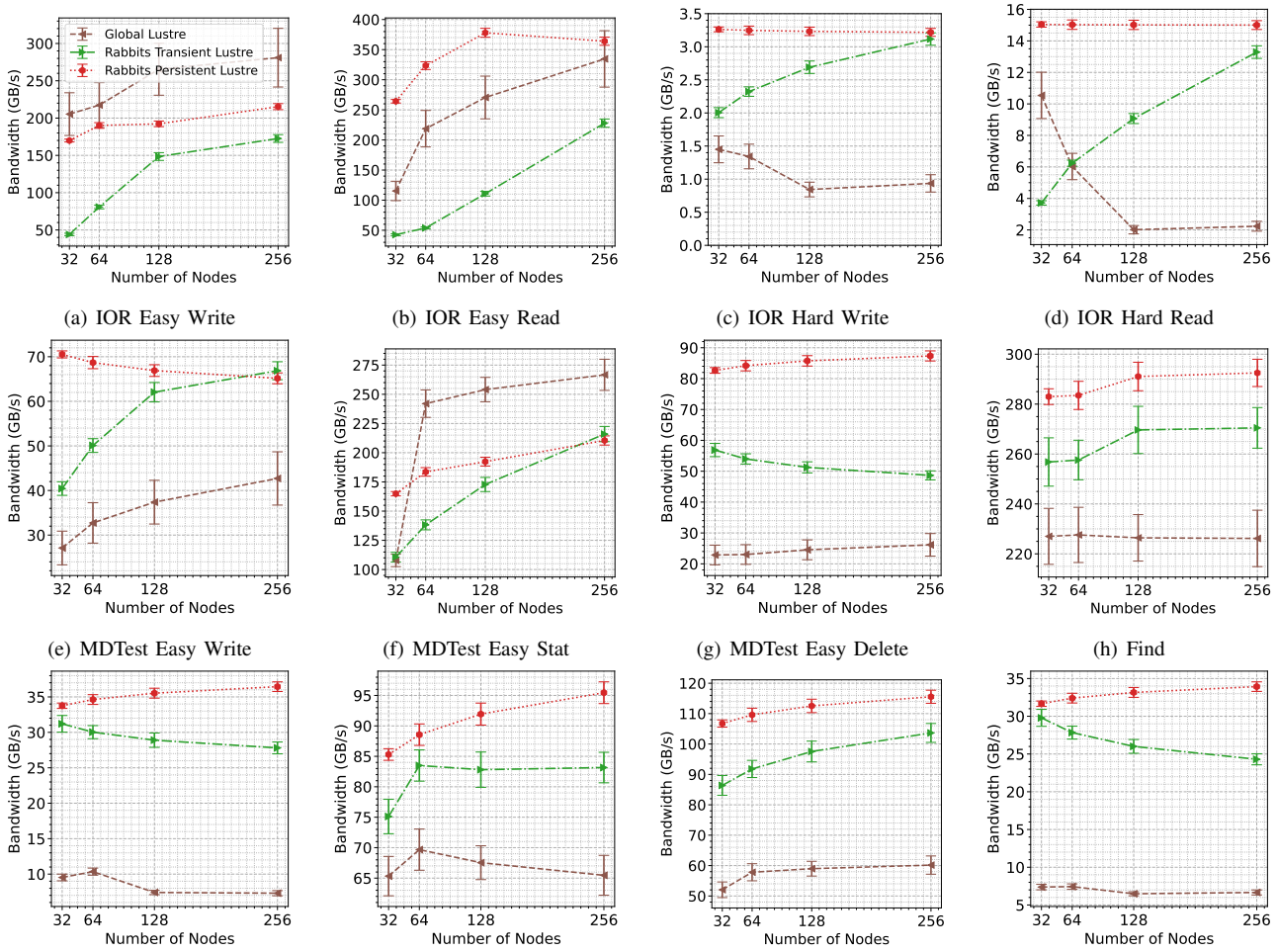
Observations and Analysis: The Global Lustre achieves ≈160 GB/s with ±15% variability across scales, expected for a shared system with external contention. Transient Lustre improves performance by 1.07× to 1.21× with ±5% variance, benefiting most at small scales where locality and NVMe dominate. Persistent Lustre is 1.08× (at larger scales) to 47× (at smaller scales) faster with ±4% variability because it uses a fixed 16-Rabbit pool. Performance scales to 32 nodes, then saturates as Rabbit bandwidth becomes the limiting factor. Transient and Persistent converge at the largest scale once they use the same Rabbit count. The lower variability for Rabbit configurations reflects dedicated hardware and reduced system noise, which is particularly valuable for performance predictability. This also indicates that transient provisioning can match persistent performance when resource counts align. The results suggest that hardware allocation, rather than software mode, is the primary driver of throughput at scale.

Key Takeaway: Rabbit enables flexible hardware configurations that enables workloads to balance performance, resource efficiency, and campaign-level requirements.

E. Performance with IO500

To assess Rabbit's performance under a community-accepted benchmark suite, we evaluate IO500 with configurations modified from the default settings to better utilize Lustre-based systems. We enable the `-posix.odirect` flag to mitigate Lustre client locking issues, set the block size to 1024 MB, and configure `segmentCount` to 5120. Each test uses 96 processes per compute node. We compare three configurations: (1) the global Lustre file system (baseline), (2) Rabbit Transient Lustre, where one Rabbit node is provisioned per 16 compute nodes, and (3) Rabbit Persistent Lustre, where 16 Rabbit nodes host a shared Lustre file system. Figure 7 compares the aggregate bandwidth and metadata rates across IO500 sub-tests, grouped into IOR (easy/hard, read/write), MDTest (easy/hard, write/read/stat/delete), and `pfind`. Each bar group represents a storage configuration (Global Lustre, Transient Lustre, Persistent Lustre) for a given test.

Observations and Analysis: For IOR easy write, Global Lustre achieves a peak bandwidth (111 GB/s to 280 GB/s), about



(a) IOR Easy Write (b) IOR Easy Read (c) IOR Hard Write (d) IOR Hard Read

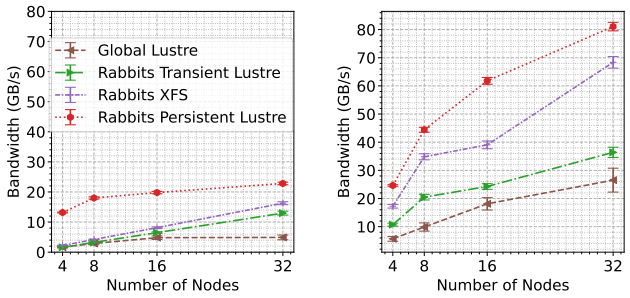
(e) MDTest Easy Write (f) MDTest Easy Stat (g) MDTest Easy Delete (h) Find

(i) MDTest Hard Write (j) MDTest Hard Read (k) MDTest Hard Stat (l) MDTest Hard Delete

Fig. 7. Persistent Rabbit improves data operations by 9× and metadata operations by 7× compared to the Global Lustre.

30% higher than Rabbit-based configurations, but with 16% variability because it aggregates a large HDD pool with many servers. Rabbit Persistent Lustre reaches up to 215 GB/s, while Rabbit's Transient Lustre delivers 172 GB/s; lower peaks are expected given the smaller Rabbit footprint per job. In contrast, IOR easy read favors Rabbit-based configurations: Persistent Lustre achieves 413 GB/s, outperforming Global Lustre's 336 GB/s and Rabbit's Transient Lustre's 228 GB/s, consistent with NVMe read bandwidth. Variability is lower for Rabbits (3-4%) due to dedicated resources. For write-intensive hard I/O patterns, Rabbit-based configurations outperform Global Lustre due to localized NVMe storage and reduced network contention. Transient Lustre delivers 1.15×-3.32× improvement, while Persistent Lustre provides an additional 1.03×-2.6× improvement over Transient Lustre. For IOR hard read, Global Lustre performance degrades with scale, whereas Rabbit Persistent Lustre maintains a consistent 15 GB/s, and Transient Lustre scales predictably with added Rabbit nodes. The overall trend matches the expectation that NVMe proximity helps most for reads and irregular access patterns, while peak write rates benefit more from large shared storage systems. The lower variance on Rabbit suggests more stable QoS, relevant for workflow pipelines that depend on repeatable I/O behavior.

Rabbit-based configurations typically outperform Global Lustre for easy metadata operations, delivering up to 7× improvements because metadata traffic stays closer to clients and benefits from NVMe latency. At high-scale `stat`, differences narrow as CPU/cache effects dominate. These results highlight Rabbit's advantage for metadata-heavy workflows, though MDT count limits restrict scaling at extreme cases. For hard metadata operations, Persistent Lustre consistently outperforms Transient Lustre, which in turn exceeds Global Lustre. Write-heavy cases see the largest gains (up to 4.2×), while read operations show smaller but still significant gains (≈1.5×). Variability remains low for Rabbit-based configurations (5%-10%). The gap is largest when metadata updates serialize on shared servers, which is common on the global file system. This indicates that Rabbit is particularly effective for workflows dominated by many small files or checkpoint directories. The remaining gap to ideal scaling suggests metadata server fan-out is the next limit to address. In `pfind`, Rabbit-based configurations show consistent gains: Transient Lustre is 13%-19% higher than Global Lustre, and Persistent is 20%-29% higher. Global Lustre shows higher variability (17%)



(a) BERT (b) CosmoFlow

Fig. 8. Rabbits with XFS provides linear scalibility but is limited by Rabbit node capacity, while Persistent Lustre scales well and achieves better bandwidth.

versus 3% for Rabbit, consistent with shared-system noise.

Across all IO500 sub-tests, Rabbit-based configurations exhibit two key trends: (1) *lower variability with improved predictability*, with performance variation typically under 5% compared to 15-17% for Global Lustre, and (2) *clear benefits for irregular and metadata-intensive workloads*, where Rabbit's NVMe-based architecture and proximity to compute eliminate shared-resource bottlenecks inherent in centralized storage systems. Rabbit-based configurations outperform in most cases, particularly for workloads that stress metadata operations or require stable performance under scaling.

Key Takeaways: Rabbit-based configurations deliver more consistent and predictable performance than Global Lustre, especially for metadata-heavy and mixed I/O patterns. Persistent Lustre generally outperforms Transient Lustre, particularly at scale, making it ideal for long-running campaigns.

F. Performance with DLIO Benchmark

We evaluate Rabbit's suitability for AI workloads using the DLIO benchmark, which captures I/O behaviors of deep learning pipelines. Two representative models are tested: BERT, a compute-heavy workload with moderate I/O requirements, and CosmoFlow, an I/O-intensive workload dominated by data ingestion. Both workloads use TensorFlow with TFRecord datasets, staged-in before execution with dataset and sample shuffling enabled. For BERT, the dataset size is 452 GB (500 files, 313k samples), while CosmoFlow uses a 1.1 TB dataset (500k single-sample files). Rabbit-based configurations tested include XFS (data copied into local partitions), Rabbit's Transient Lustre, and Persistent Lustre (spanning 16 Rabbit nodes), compared against the global Lustre baseline. Figure 8 plots bandwidth (GB/s) versus compute scale for BERT and CosmoFlow across all configurations.

Observations and Analysis: For BERT, Global Lustre achieves up to 4.8 GB/s, while Rabbit XFS reaches 16 GB/s with near-linear scaling as more Rabbit nodes are added. This is expected because BERT is compute-heavy and does not saturate storage, so reduced latency and local NVMe reads mainly reduce stalls. Rabbit's Transient Lustre scales similarly, achieving 13 GB/s at 32 nodes, consistent with shared file system overheads versus XFS. Persistent Lustre delivers the highest throughput, up to 23 GB/s, but gains are sub-linear because I/O is not the dominant bottleneck. The staged-in datasets emphasize read behavior, so improvements largely reflect read-side locality. The remaining gap to peak NVMe bandwidth indicates compute is still the limiting factor. This also suggests that further storage scaling would bring diminishing returns without more computation. For CosmoFlow, Global Lustre reaches 150 GB/s, while Rabbit's Transient Lustre achieves 102 GB/s with two Rabbit nodes at 32 nodes. Rabbit XFS reaches up to 68 GB/s with linear scaling, while Persistent Lustre achieves 83-175.8 GB/s, peaking with full utilization of all 16 Rabbit nodes. The higher sensitivity to Rabbit allocation is expected because CosmoFlow is I/O-bound and can exploit additional NVMe bandwidth; allocating more Rabbit nodes directly increases available bandwidth. Persistent Lustre consistently provides the highest performance under heavy I/O demands. The two workloads highlight contrasting stress patterns: BERT benefits from moderate I/O improvements, while CosmoFlow fully exploits Rabbit bandwidth. Persistent Lustre consistently leads; XFS scales linearly; Transient Lustre balances performance and resource use.

Key Takeaways: Rabbit's flexibility enables workload-aware optimization. Compute-bound workloads (BERT) scale efficiently with moderate resources, while I/O-bound workloads (CosmoFlow) benefit from Persistent Lustre across multiple nodes. Read-only XFS and tuned Transient Lustre offer throughput-resource trade-offs.

V. Conclusion

This work presents the first systematic study of the Rabbit Storage System, evaluating its capabilities and performance under diverse configurations and workloads using benchmarks such as IOR, IO500, and DLIO. Our study uncovered three key insights into Rabbit's architectural flexibility and its impact on HPC and AI workloads. First, Rabbit enables software-defined configurability that allows users to deploy different file systems (XFS, GFS2, and Lustre) and tune parameters for workload-specific optimization, achieving up to 3.12× higher bandwidth than global Lustre in IOR tests. Second, tuning Lustre configurations (e.g., OST counts, compression, RPC settings) improves performance by up to 4.6× over the baseline with combined optimizations reaching 26 GB/s per Rabbit node. Third, benchmark-based evaluation shows that Rabbit achieves strong end-to-end performance: IO500 tests delivered a peak sequential read bandwidth of 413 GB/s (Persistent Lustre), metadata performance improved by 3.3×–7× over global Lustre, and DLIO workloads achieved up to 22.83 GB/s for BERT and 175.8 GB/s for CosmoFlow using Persistent Lustre. These results demonstrate that Rabbit is a versatile accelerator for modern workloads, bridging performance gaps in I/O-intensive simulations, analytics, and AI pipelines.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-JRNL-2014051). This material is based upon work supported by LLNL LDRD 25-ERD-042 and 24-ERD-047.

REFERENCES

[1] S. Gathu, "High-Performance Computing and Big Data: Emerging Trends in Advanced Computing Systems for Data-Intensive Applications," *Journal of Advanced Computing Systems*, vol. 4, no. 8, pp. 22–35, Aug. 2024, number: 8. [Online]. Available: https://scipublication.com/index.php/JACS/index

[2] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook, K. Maschhoff, J. Sewall, N. Kumar, S. Ho, M. F. Ringenburg, P. Prabhat, and V. Lee, "CosmoFlow: Using Deep Learning to Learn the Universe at Scale," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp. 819–829. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8665771

[3] N. Siva, "1000 genomes project," *Nature Biotechnology*, vol. 26, no. 3, pp. 256–257, Mar. 2008, publisher: Nature Publishing Group. [Online]. Available: https://github.com/pegasus-isi/1000genome-workflow

[4] B. L. DeCost, J. R. Hattrick-Simpers, Z. Trautt, A. G. Kusne, E. Campo, and M. L. Green, "Scientific AI in materials science: a path to a sustainable and scalable paradigm," *Machine Learning: Science and Technology*, vol. 1, no. 3, p. 033001, Jul. 2020, publisher: IOP Publishing. [Online]. Available: https://dx.doi.org/10.1088/2632-2153/ab9a20

[5] M. Rynge, G. Juve, J. Kinney, J. Good, B. Berriman, A. Merrihew, and E. Deelman, "Producing an Infrared Multiwavelength Galactic Plane Atlas Using Montage, Pegasus, and Amazon Web Services," *Annual Astronomical Data Analysis Software and Systems (ADASS)*, vol. 485, p. 211, May 2014, conference Name: Astronomical Data Analysis Software and Systems XXIII ADS Bibcode: 2014ASPC..485..211R. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2014ASPC..485..211R

[6] H. Devarajan and K. Mohror, "Extracting and characterizing I/O behavior of HPC workloads." Heidelberg, Germany: IEEE, Sep. 2022.

[7] B. R. Landsteiner, D. Henseler, D. Petesch, and N. J. Wright, "Architecture and Design of Cray DataWarp," 2016. [Online]. Available: https://shorturl.at/5pQ2Q

[8] W. Schenck, S. El Sayed, M. Foszczynski, W. Homberg, and D. Pleiter, "Evaluation and Performance Modeling of a Burst Buffer Solution," *ACM SIGOPS Operating Systems Review*, vol. 50, no. 2, pp. 12–26, Jan. 2017. [Online]. Available: https://doi.org/10.1145/3041710.3041714

[9] T. Wang, A. Moody, W. Yu, K. Sato, and K. Mohror, "An Ephemeral Burst-Buffer File System for Scientific Applications," Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), Tech. Rep. BurstFS; 005217WKSTN00, Apr. 2017. [Online]. Available: https://www.osti.gov/biblio/1351607-ephemeral-burst-buffer-file-system-scientific-applications

[10] M. J. Brim, A. T. Moody, S.-H. Lim, R. Miller, S. Boehm, C. Stanavige, K. M. Mohror, and S. Oral, "UnifyFS: A User-level Shared File System for Unified Access to Distributed Local Storage," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2023, pp. 290–300, iSSN: 1530-2075.

[11] A. Kougkas, H. Devarajan, and X.-H. Sun, "Hermes: a heterogeneous-aware multi-tiered distributed I/O buffering system," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '18. New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 219–230.

[12] S. Byna, Q. Koziol, V. Vishwanath, J. Soumagne, H. Tang, J. Mu, B. Dong, R. A. Warren, F. Tessier, T. Wang, and J. Liu, "Proactive Data Containers (PDC): An Object-centric Data Store for Large-scale Computing Systems," *AGU Fall Meeting Abstracts*, vol. 2018, pp. IN34B–09, Dec. 2018.

[13] "El Capitan | HPC @ LLNL." [Online]. Available: https://hpc.llnl.gov/hardware/compute-platforms/el-capitan

[14] "Tuolumne | HPC @ LLNL." [Online]. Available: https://hpc.llnl.gov/hardware/compute-platforms/tuolumne

[15] "Using El Capitan Systems: File Systems and Rabbits | HPC @ LLNL." [Online]. Available: https://hpc.llnl.gov/documentation/user-guides/using-el-capitan-systems/file-systems-rabbits

[16] Y. Fan, Z. Lan, P. Rich, W. E. Allcock, M. E. Papka, B. Austin, and D. Paul, "Scheduling Beyond CPUs for HPC," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '19. New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 97–108. [Online]. Available: https://dl.acm.org/doi/10.1145/3307681.3325401

[17] H. Sung, J. Bang, C. Kim, H.-S. Kim, A. Sim, G. K. Lockwood, and H. Eom, "BBOS: Efficient HPC Storage Management via Burst Buffer Over-Subscription," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, May 2020, pp. 142–151. [Online]. Available: https://ieeexplore.ieee.org/document/9139729

[18] IBM, "CSMD Configuration — CAST 1.5.0 documentation." [Online]. Available: https://tinyurl.com/y6cwj782

[19] A. Sweeney, "Scalability in the {XFS} File System," 1996. [Online]. Available: https://www.usenix.org/conference/usenix-1996-annual-technical-conference/scalability-xfs-file-system

[20] "Global File System 2 | Red Hat Enterprise Linux | 7 | Red Hat Documentation." [Online]. Available: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html-single/global_file_system_2/index

[21] P. Braam, "The Lustre Storage Architecture," *arXiv:1903.01955 [cs]*, Mar. 2019, arXiv: 1903.01955. [Online]. Available: http://arxiv.org/abs/1903.01955

[22] F. Di Natale, H. Bhatia, T. S. Carpenter, C. Neale, S. Kokkila-Schumacher, T. Oppelstrup, L. Stanton, X. Zhang, S. Sundram, T. R. W. Scogland, G. Dharuman, M. P. Surh, Y. Yang, C. Misale, L. Schneidenbach, C. Costa, C. Kim, B. D'Amora, S. Gnanakaran, D. V. Nissley, F. Streitz, F. C. Lightstone, P.-T. Bremer, J. N. Glosli, and H. I. Ingólfsson, "A massively parallel infrastructure for adaptive multiscale simulations: modeling RAS initiation pathway for cancer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 1–16.

[23] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An In-Depth Analysis of the Slingshot Interconnect," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2020, pp. 1–14. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9355230

[24] I. F. , "IO500." [Online]. Available: https://io500.org

[25] H. Devarajan, H. Zheng, A. Kougkas, X.-H. Sun, and V. Vishwanath, "DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2021, pp. 81–91.

[26] "NearNodeFlash/NearNodeFlash.github.io," May 2025, original-date: 2022-09-30T18:00:20Z. [Online]. Available: https://github.com/NearNodeFlash/NearNodeFlash.github.io

[27] A. M. Beltre, P. Saha, M. Govindaraju, A. Younge, and R. E. Grant, "Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms," in *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, Nov. 2019, pp. 11–20. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8950981

[28] A. Smith, G. H. Loh, M. J. Schulte, M. Ignatowski, S. Naffziger, M. Mantor, M. F. N. Kalyanasundharam, V. Alla, N. Malaya, J. L. Greathouse, E. Chapman, and R. Swaminathan, "Realizing the AMD Exascale Heterogeneous Processor Vision : Industry Product," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2024, pp. 876–889. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10609636

[29] J. Kunkel, "HPC IO Benchmark v 4.0.0," Apr. 2025, original-date: 2015-06-23T15:56:39Z. [Online]. Available: https://github.com/hpc/ior/releases/tag/4.0.0

[30] "bert in mlcommons training benchmarks." [Online]. Available: https://github.com/mlcommons/training/tree/master/retired_benchmarks/bert

[31] J. Y. Moon, F. Di Natale, H. I. Ingolfsson, H. Bhatia, and J. R. Chavez, "Multiscale Machine-Learned Modeling Infrastructure RAS," Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), Tech. Rep. MuMMI RAS, Sep. 2021.

[32] A. George, A. Dilger, M. J. Brim, R. Mohr, A. Shehata, J. Y. Choi, A. M. Karimi, J. Hanley, J. Simmons, D. Manno, V. M. Vergara, S. Oral, and C. Zimmer, "Lustre Unveiled: Evolution, Design, Advancements, and Current Trends," *ACM Trans. Storage*, vol. 21, no. 3, pp. 21:1–21:109, Jun. 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3736583