GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Emerging Trends in Parallel File Systems

Abdellah Omar Adolf

MatrNr: 20623216

Supervisor: Patrick Höhn

Georg-August-Universität Göttingen
Institute of Computer Science

April 4, 2024

Abstract

In high performance computing environments with its traditional Parallel or Distributed File systems, metadata management is essential especially with the exponential growth in data. Metadata operations could account for more than half of all system access activities, particularly for large-scale deployments in modern data centres where metadata volumes can reach petabytes. Without efficient metadata management approach, this could create a persistent performance bottleneck. Traditional parallel file systems often store metadata on a single server. Managing and maintaining consistency of this centralized metadata across numerous storage nodes in large-scale environments with millions of files and directories becomes a significant challenge in terms of scalability (capacity and throughput). The use of multiple metadata servers offers a possible solution; however, efficiently indexing and distributing metadata nodes across different metadata servers presents enormous challenge in terms of effective techniques for distribution, ensuring load balance and preserving node locality. This report presents two emerging metadata management services—Duplex and AdaM—that hold the promise of addressing these aforementioned challenges. Duplex enhance scalability by augmenting capacity and throughput, while concurrently ensuring stability and minimizing latency for latency-sensitive applications. Duplex adopts novel partitioning and indexing methods grounded in flattened metadata management, effectively addressing load balancing and mitigating activity hotspots within super-directories. By employing a dedicated server for permission management, Duplex efficiently manages file permissions to enhance system performance. Evaluations shows that Duplex significantly reduces the average lookup latency by up to 84% and the 99th percentile tail latency by up to 88.2% for metadata-intensive benchmarks. Furthermore, Duplex showed an enhancement in lookup IOPS by up to $7.6\times$ /$2.3\times$ when compared to established systems such as CephFS and BeeGFS. AdaM on the other hand, uses a different approach, focussing on dynamically managing the load distribution of metadata across multiple metadata servers (MDSs) while preserving metadata locality through the use of deep reinforcement learning technique (deep deterministic policy gradient algorithm) to learn to migrate as well as minimizing migration costs when relocating and distributing hot metadata nodes. Evaluation on Amazon's cloud platform (EC2) significantly showed that AdaM outperforms other technieques (hash-based mapping, static sub-tree partition, dynamic sub-tree partition and AngleCut) in terms of how fast it can answer queries, how well it balances competing goals (balancing metadata load while preserving locality), how well it adapts to changing data access patterns, and how much it costs to migrate data to ensure load balance.

Statement on the usage of ChatGPT and similar tools in the context of examinations

In this work I have used ChatGPT or a similar AI-system as follows:

- ☑ Not at all
- ☐ In brainstorming
- ☐ In the creation of the outline
- ☐ To create individual passages, altogether to the extent of 0% of the whole text
- ☐ For proofreading
- ☐ Other, namely: -

I assure that I have stated all uses in full.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Figures

List of Abbreviations

HPC High-Performance Computing

PFS Parallel File System

DFS Distributed File System

MDS Metadata Server

PMS Permission Merging Server

CH Consistent Hashing

DCH Double Consistent Hashing

IOPS Input/Output Operations Per Second

GPFS General Parallel File Systems

PVFS Parallel Virtual File System

SAN Storage Area Network

RDMA Remote Direct Memory Access

(OSS) Object Storage Server

(OST) Object Storage Server

MDTs Metadata Targets

POSIX Portable Operating System Interface

DRL Deep Reinforcement Learning

DQN Deep Q-network

PG Policy Gradient

DPG Deterministic Policy Gradient

DDPG Deep Deterministic Policy Gradient

1 Introduction

1.1 Parallel File Systems in High Performance Computing

A Parallel File System (PFS) is a kind of Distributed File System (DFS) that allows information to be distributed among multiple servers and provides concurrent access [Dub19]. Parallel File Systems form a critical component of the infrastructure supporting High-Performance Computing (HPC) [Bor+23]. They play significant role in management of data and storage. Their design principles, scalability features and optimization for parallel input/output (I/O) make them [Car+09] indispensable for handling the immense data processing requirements of modern scientific simulations, data analytics and computational research in various domains.

Parallel file systems employed in HPC environments have progressively become more specialized, aiming to optimize performance by leveraging the underlying storage hardware for computational application workloads [Car+09]. However, these immense data processing requirements have outpaced the design and architecture of PFS. Thus, the rate of improvement in storage device capabilities lags behind, resulting in a considerable gap between the volume of data generated and the capacity of storage devices to efficiently store this data. File systems therefore "need to adopt to the needs of new applications, storage and communications, among others" [Gar+23]. This development has led to continuous evolution of parallel file systems to reflect the dynamic nature of HPC environments and the efforts to enhance performance, scalability and fault tolerance in the realm of large-scale parallel computing.

There are a number of Parallel File Systems (PFSs). However, some of the main PFSs used in HPC are: General Parallel File System (GPFS); Lustre; Parallel Virtual File System (PVFS); Expand 1.0; Gekko FS [Gar+23] and BeeGFS [BPT22]. A brief overview of GPFS, Lustre and PVFS is presented to usher the reader into the space of PFS. Lustre and GPFS are the most widely used PFS in High-Performance Computing systems [Zhe+21], with Lustre dominating in over 80% of the systems in the Top 500 list [ZL22].

1.1.1 General Parallel File System (GPFS)

GPFS—IBM's parallel, shared-disk file system— is designed for cluster computers, supporting up to 256 mounted file systems with applications accessing files through standard UNIX interfaces or specialized interfaces for parallel programs [Dub19]. Known for high performance, GPFS achieves this feat through striping data across multiple disks, high-performance metadata scans, large configurable block sizes and advanced algorithms for improved I/O functions [Dub19]. GPFS, which is used on many of the world's largest supercomputers, including ASCI White, ensures file system coherency and consistency through sophisticated locking and token management [SH02].

GPFS scales to the largest clusters globally, employing a shared-disk architecture that balances load and achieves full throughput [SH02]. It supports fully parallel access to both file data and metadata, utilizing distributed locking to synchronize read-write disk accesses across multiple nodes [SH02]. GPFS is highly scalable, supporting concurrent access from multiple Storage Area Network (SAN) or network-attached nodes, and boasts features like data replication, policy-based storage management and multi-site operations [EHM17]. Its flexibility extends to deployment across various operating systems [EHM17].

With features like distributed metadata, replication, and automatic recovery, GPFS eliminates the risk of single points of failure, providing fully POSIX semantics and ensuring workload isolation [EHM17]. By integrating different disk drives, utilizing client-side caches, enabling simultaneous file access and optimizing query languages for fast sorts, GPFS enhances scalability and efficiency [Dub19].

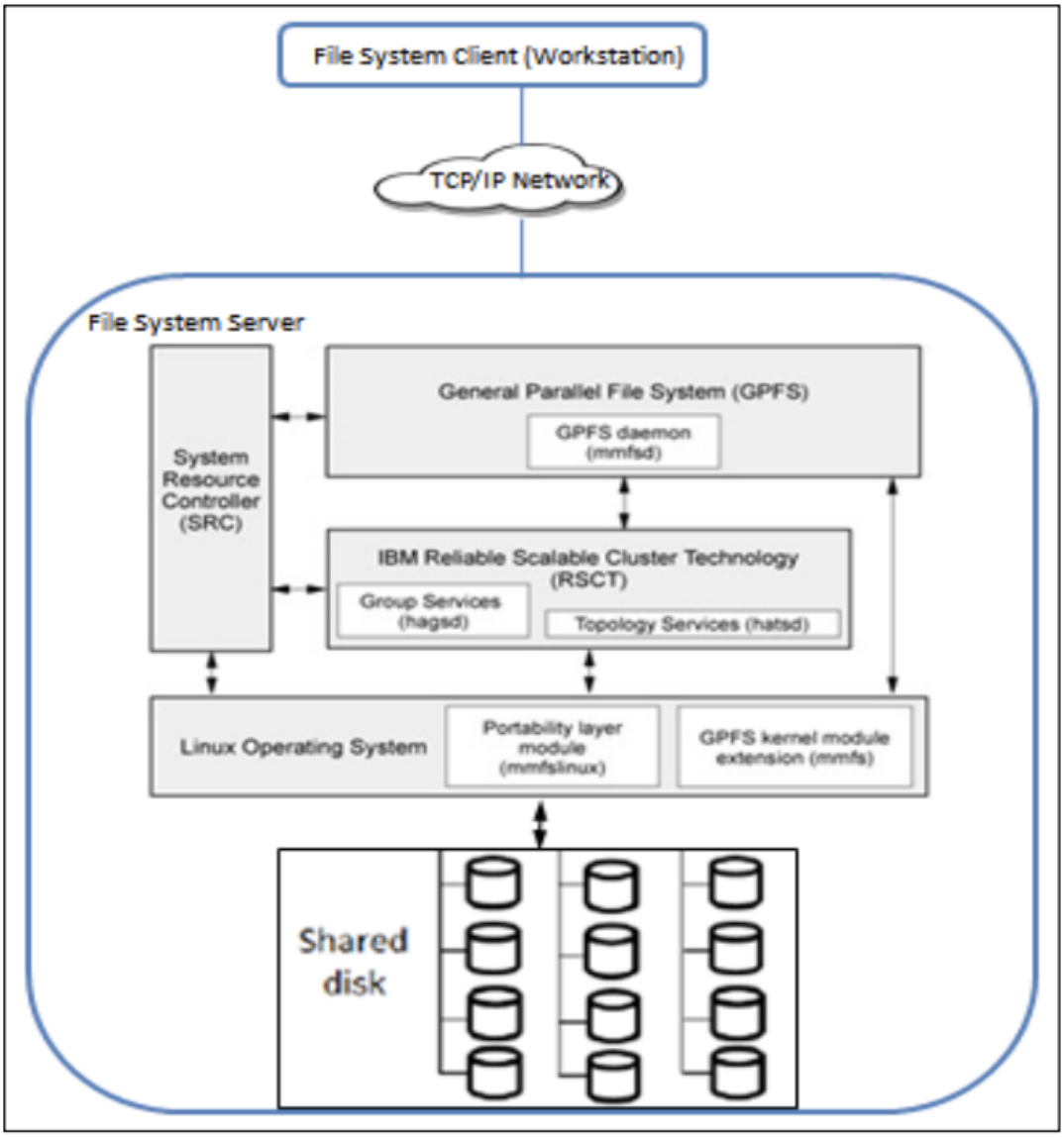

Figure 1: GPFS File System Architecture
[EHM17]

1.1.2 Lustre

Lustre, a distributed parallel filesystem, is a cornerstone in large-scale cluster computing, evolving from a 1999 research project at Carnegie Mellon University [Dub19; Gar+23]. Named by a portmanteau of 'Linux' and 'Cluster', Lustre emphasizes high bandwidth across diverse hardware, seamlessly scaling from large clusters to commercial setups [Gar+23]. Lustre leverages a modified ext4 system, called ldiskfs, to optimize performance and functionality [Dub19]. For even higher performance, Lustre supports high-

speed networking options like Remote Direct Memory Access (RDMA) over Infiniband [Dub19].

The use of Lustre in large-scale cluster computing are evident in its global recognition [ZL22; Pau+20]. Lustre has become the foremost PFS globally, known for its open-source nature and its ability to deliver high bandwidth and high availability across diverse hardware environments [Man+22]. A large number of clients can connect to Lustre thanks to its POSIX-compliant interface, which allows for concurrent file system usage.[Dub19].

Lustre's architecture consists of Metadata Server (MDS), Object Storage Server (OSS) and Clients. MDS manages metadata stored in Metadata Targets (MDTs), while OSS stores actual file data in Object Storage Targets (OSTs) . Clients mount and use the Lustre file system over the network. The client-server network architecture, combined with the separation of metadata and data services, enhances parallel file access and overall system performance [Dub19].

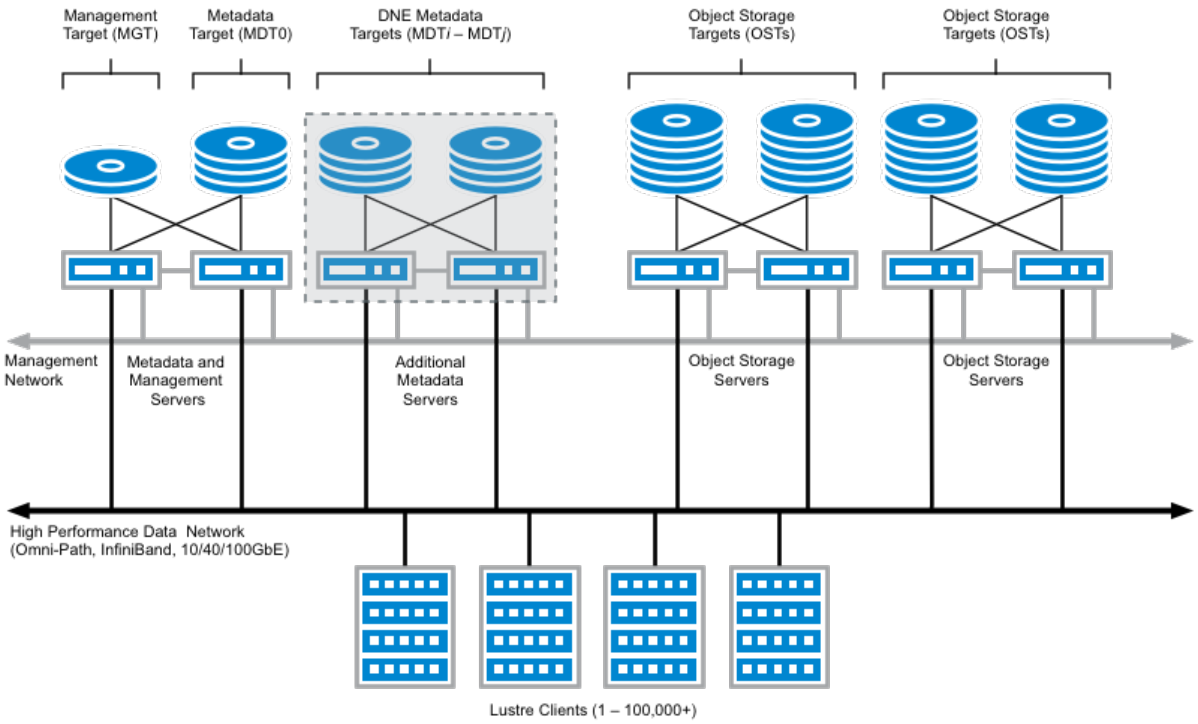

Figure 2: Lustre File System Architecture, [Wik17].

1.2 Challenges and Limitations Associated with Existing parallel file systems

Parallel File Systems handle intensive operations in terms of large-scale data storage and access in parallel computing environments. Often challenges and limitations (due to the sheer volume of data and concurrent access) arises which subsequently affects performance, reliability, consistency, access and metadata management. As data grows exponentially beyond Petabytes to Exabyte scale [Hua+23], there is the need to find emerging trends that can help resolve some of these challenges if not all. Below are some of the overarching challenges and limitations.

- Metadata management [Hua+23]
- Scalability [Dai+22]
- I/O bottlenecks [Hua+23]
- Consistency [Lia+21]
- Fault tolerance and reliability
- Storage performance bottlenecks [Zhe+22]

In this report, the first two challenges are discussed towards achieving the objective of the study outlined under subsection 1.3 on page 5.

1.2.1 Metadata management

Accessing metadata in Distributed File Systems is crucial, constituting over fifty percent of system accesses [Don+23]. This consistently poses a performance bottleneck, particularly for large-scale operations with petabyte-sized metadata volumes in contemporary data centers. PFSs often store metadata, such as file attributes and directory structures, centrally using a single metadata server [Hua+23]. Managing such metadata and ensuring its consistency across distributed storage nodes can be complex, especially in large-scale deployments with millions of files and directories. Thus, while a single metadata server streamlines design, it can become a performance bottleneck and reliability risk, making a distributed approach more efficient and resilient [Hua+23][Don+23].

Even with distributed approach the mere exponential increase of data presents a problem. In modern HPC infrastructure, application are increasingly read and are also metadata intensive [Hua+23; Dai+22] such that I/O applications—coupled with the concurrent submission of numerous applications—can quickly overwhelm shared file system metadata resources, leading to performance degradation and unfair I/O distribution. Metadata management efficiency therefore becomes crucial in improving system performance and reliability.

The natural questions that arise are: How can metadata nodes be distributed reasonably well across different Metadata Servers (MDSs) to ensure load balance and avoid hotspots? And if there is any success in achieving this, how can the distribution be done such that node locality is preserved? Moreover, can there be efficient permission management?

1.2.2 scalability and Metadata management

Achieving high scalability is the bedrock for the adoption of Metadata Servers (MDS) in design and architecture of metadata management services/technology [Dai+22] in recent years. The critical role of metadata access in DFS environments necessitates exploring scalable solutions. Addressing the bottleneck caused by frequent metadata access in large-scale DFS deployments with petabyte-scale metadata volumes is crucial. With the deployment of MDS to achieve scalability, the core challenge lies in dividing the vast filesystem namespace (all files and directories) into manageable chunks and distributing them efficiently across multiple MDS cluster [Dai+22]. In most cases distribution does not ensure load balance among MDSs and where it does may result in non-preservation of node locality [Dai+22; Don+23; Hua+23].

Hao Dai et al. (2022) [Dai+22] categorizes high-performance technologies for MDS into three main areas. The first focuses on scalability, aiming to improve overall file system performance through techniques like load balancing and eliminating hotspots. The second category tackles metadata queries by establishing new indexing methods for existing metadata attributes to enhance the efficiency of various applications that rely on metadata searches. The third category, known as value-added technologies, goes a step further by leveraging standard metadata sets along with additional application-specific information. This optimization is achieved by harnessing the application-specific data stored in the enhanced Metadata Server (MDS) to improve the performance of those applications.

1.3 Objective of the study

This study—in line with the course "Newest Trends in High-Performance Data Analytics"—aims to find emerging trends in Parallel File Systems that present solutions to metadata management challenges. To this end, two emerging trends have been thoroughly presented in subsections 2.2 and 2.3.

1. Duplex: "[A] Low-latency and scalable full-path indexing metadata service for Distributed File Systems" [Don+23].

2. AdaM: "An Adaptive metadata management scheme based on Deep Reinforcement Learning for large-scale Distributed File Systems" [Hua+23].

1.3.1 Why Emerging Trends?

The exponential growth of data into exabytes (EB) from petabytes (PB) creates a challenge for High-Performance Computing environments [Hua+23]. With millions of directories and billions of files, commensurate processing power is necessary. However, data is not the only thing growing. Metadata— the information that describes the data itself—also increases proportionally. Even though metadata itself constitutes a small fraction, typically 0.1% to 1% of the total data space, in an exabyte-scale file system dealing with enormous amounts of data, this percentage can translate to petabytes (PB) of metadata [Hua+23].

In DFS, accessing metadata sits on the critical data path, meaning it is essential for most file system operations [Don+23]. This can lead to performance bottlenecks due to the high volume of I/O (Input/Output) operations required. In fact, studies have shown that metadata can account for up to 80% of file system operations in large HPC environments [Hua+23; Don+23; Dai+22]. This emphasizes the importance of efficient metadata management because even a small percentage can become quite significant at these massive scales. Therefore, efficient management of metadata becomes crucial for optimizing system performance in such data-intensive environments.

2 Emerging Trends in Metadata Management

2.1 Overview of State of the Art Metadata Management

File system metadata refers to the information responsible for managing the namespace, permission semantics, and the locations of file data blocks [Dai+22]. State of the art metadata management is defined in this study as the evolving and innovative technologies that are becoming increasingly prominent in the field of metadata management. Thus, as data continue to grow in size and complexity, emerging practices aim to address new challenges and leverage advancements in technology. These practices often involve the adoption of novel tools, methodologies, and strategies to handle metadata more efficiently, ensuring better organization, accessibility and utilization of data. By staying abreast of emerging trends, organizations can enhance their metadata management processes to keep pace with evolving data landscapes and derive valuable insights from their information resources.

In the context of file systems, being able to manage data at the exabyte scale level implies an enormous storage capacity that is suitable for handling very large datasets, especially HPC environments or data-intensive applications [Xu+13]. As such, the efficiency of metadata management services can significantly enhance overall system performance.

Established metadata management strategies, such as hash-based mapping and subtree partitioning, offer solutions for workload distribution among metadata servers, balancing trade-offs between even distribution and hierarchical locality [Dai+22; Xu+13]. Thus, in distributed metadata management, strategies like hash-based mapping aim to evenly distribute the metadata workload among servers. However, this approach comes at the cost of eliminating hierarchical metadata locality [Xu+13].

Even with dynamic subtree partitioning, hotspot issues and redundant migration costs often arise when access patterns change [Hua+23]. Furthermore, subtree partitioning does not uniformly distribute the workload, and there is the need for occasional metadata migration to maintain balanced loads among servers [Hua+23]. These methods showcase the ongoing trade-offs between achieving load balance and preserving hierarchical metadata organization in distributed systems. Figure 3 shows the popular and state of the art metadata management techniques while Figure 4 shows the timeline of development of metadata techniques.

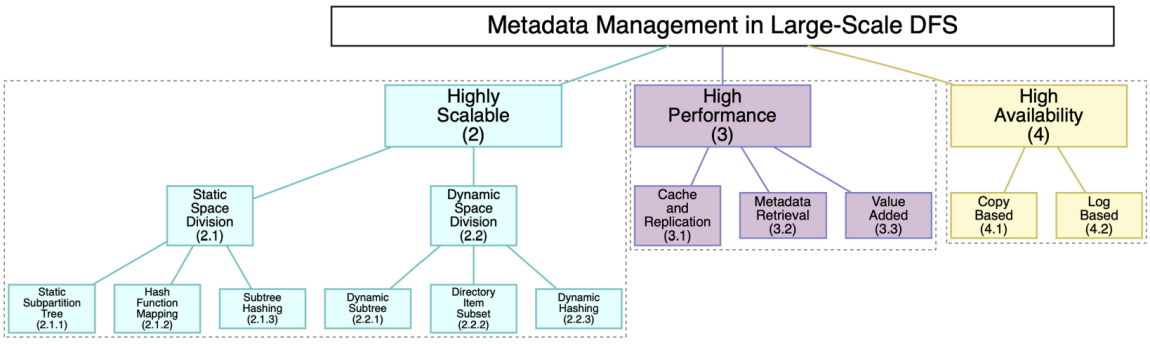

Figure 3: Popular Metadata Management Methods/Technologies [Dai+22]

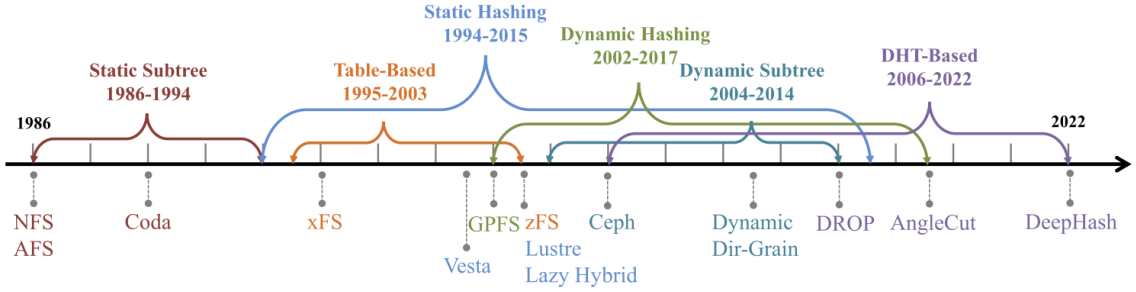

Figure 4: Development of metadata management techniques over the years. [Hua+23]

2.2 Duplex

2.2.1 Motivation

Traditional file systems use a hierarchical tree-based structure to organize metadata where directories are branch nodes, while files are represented as leaf nodes [Don+23]. As metadata enormously increases, scalability becomes important to: reduce performance bottlenecks; ensure load balance; and improve reliability of the system [Dai+22; Hua+23]. Moreover, large data analysis tasks demand swift access to data across various storage servers, necessitating high reading and writing speeds for the storage system [Dai+22].

In order to achieve extensive scalability in both capacity and throughput, contemporary metadata solutions often adopt full-path indexing within the framework of flattened metadata management [Don+23]. In other words, existing methods for storing metadata on servers are designed to handle massive amounts of data and process information quickly (high scalability in terms of capacity and throughput). However, while these techniques excel at storing large amounts of data and processing information quickly, they struggle to meet strict timeliness requirements (latency service-level objectives) for applications that demand very fast responses [Don+23]. This means they might be good at handling a lot of data overall, but they cannot guarantee retrieving specific pieces of data very quickly, which can be a problem for certain applications.

There are two main issues that prevent these approaches from achieving very fast response times: Conflict between permission checking and full-path indexing especially in POSIX-compatible file systems; and unpredictable latency in super-directories. These limitations, specifically in the context of latency, hinder the ability of these solutions to provide reliable services for applications sensitive to latency. Furthermore, these strategies face limitations such as clashes with POSIX-style permission verification and inadequate support for super-directories. As a consequence, they experience elevated and fluctuating latency, impairing their capacity to offer dependable services for latency-sensitive applications [Don+23].

Duplex is designed to address these challenges by offering Low and consistent latency which is ideal for applications that need very fast responses and high throughput and storage capacity which is important for applications that deal with massive amounts of data. According to the authors [Don+23], Duplex is a scalable DFS metadata service that leverages full-path indexing with the goal of reducing access latency and ensuring stability for latency-sensitive applications, while maintaining scalable throughput and capacity for

throughput-sensitive applications. They adopt three key design elements to address these challenges. Firstly, they incorporate a fast access path featuring a centralized permission server designed for efficient and expedited permission verification. Secondly, a tree-based permission merging algorithm is used to reduce the space footprint of the Permission Management Server (PMS) (see Figure 5). Lastly, they employ flattened metadata management based on double consistent hashing, providing a mechanism for low-latency access to super-directories. This key elements give rise to the ultimate architectural design of Duplex which reflect the three key elements.

The following section is dedicated to discussing the design and architecture of Duplex. The discussion is based on the article [Don+23] in which Duplex was authored.

2.2.2 Design and Architecture of Duplex

The architecture of Duplex comprises of three main components (Figure 5): Metadata servers (MDSs); Permission Management server (PMS); client modules.

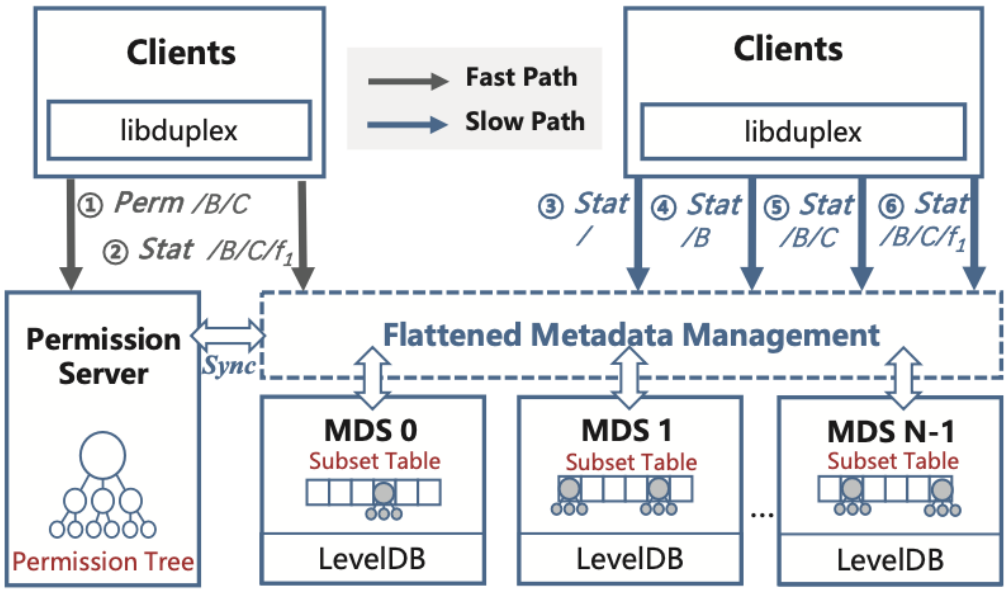

Figure 5: Duplex Architecture
[Don+23]

MDSs are the backbone of Duplex, handling the distributed storage and management of metadata. Duplex distributes the directory tree across multiple MDSs using a flattened approach that focuses on subsets of directories. This helps optimize performance. For speed, Duplex stores metadata in the memory of the MDSs. However, to ensure data reliability, updates are logged and persisted locally using a storage engine called LevelDB.

The role of the PMS is such that Duplex separates directory data into two parts: content information (like size and timestamp) and permission information (user, group, and access rights). During initialization the directory permissions are copied from the MDS cluster and stored in the single PMS. The PMS is used to centrally manage all permission information. This simplifies permission checks. By storing permissions separately and centrally, the PMS can quickly verify the permissions for a file by checking its ancestors in the directory tree. Duplex maintains consistency between the MDS cluster (where the

main data resides) and the PMS using a write-through approach. Permission updates are first applied to the MDS and then copied to the PMS. Even if the PMS crashes, hierarchical permissions can be rebuilt from the MDS cluster, ensuring reliable operation.

While the dedicated Permission Server (PMS) mitigates permission overhead and improves permission check speed, storing permissions for the entire system on a single server can limit how much permission data it can handle (scalability limitations). To address this challenge, instead of storing permissions for every single file, Duplex only stores permissions for directories since file permissions (and metadata) are retrievable from the MDSs using full-path indexing. This significantly reduces the amount of data the PMS needs to materially manage especially in large-scale DFS. For instance, the process of storing directory permissions only involves extracting directory permissions from the directory tree. Figure 6(b) on page 12 illustrates this process, where the directory tree shown in Figure 6(a) consists of 9 directories with four types of permissions and 5 files. The PMS removes unnecessary metadata, such as file metadata and directory contents, retaining only directory permissions, as depicted in Figure 6(b). Duplex identifies and merges directory permissions that are identical thus further reducing the storage requirements on the PMS.

Duplex takes advantage of the fact that sub-directories—in a typical file system—often inherit permissions from their parent directory; as such it uses this advantage to eliminate redundant permission entries in the permission tree. For instance, the process illustrated from Figure 6 (b–c) demonstrates the consolidation of the permission tree. Entries that inherit permissions from their parent directory are removed (Figure 6 (b–c)). This results in keeping only a few representative directories (only 4 directories) with unique permissions: /, /A/E, /C, and /C/F/H. Path prefixes that are identical to parent directories are removed resulting in node indexes transforming into /, A/E, C, and F/H (Figure 6(c)). This shortens the path representation for each directory to a compact prefix tree or radix tree for efficient data storing and retrieving.

With such representation, Duplex is able to handle permissions operations such as lookup, insert, remove and update seamlessly. Retrieving permissions (lookup) for a specific path involves a top-down prefix matching process, much like how a radix tree works. The PMS starts at the root and navigates through the tree based on the path components, stopping when there is no matching node for a remaining path segment.

If a path does not have a matching node in the permission tree, Duplex infers that the directory inherits permissions from its parent directory. For instance, in the directory /C/G (Figure 6(d)), G inherits permissions from C, so the PMS only needs to check permissions for / and C. When adding new directory permissions (insert operation), a top-down prefix matching process is involved where the permission tree is navigated to identify the appropriate insertion point. This process encounters three possible scenarios: If a node with a matching pathname is found, an error is raised because the directory already exists in the system; if the permissions of the current node are identical to the new permissions being added, no update is necessary as the directory inherits permissions from its parent; and finally, if the permissions differ, a new node is created as a child of the current node to represent the new directory with its unique permissions. In Figure 6(c), inserting directory /A/E/I with permissions different from /A/E would create a new node I under /A/E to store those permissions.

In terms of remove operations, when a directory is deleted in Duplex, the permission tree also removes any child directory it has. This ensures consistency between the directory structure and the permission tree. For example, removing directory /C in Figure 6(c)

requires removing nodes /C and F/H. Likewise, removing directory /C/F in Figure 6 involves removing node F/H. This recursive process ensures the complete removal of all directory components and their associated permissions.

Permission updates in Duplex can either be recursive or non-recursive. In recursive update, modification is made to permissions for an entire directory and all its sub-directories to the same value where the PMS puts all the involved directorates into a single node. This is essentially a 'remove operation' followed by an 'insert operation' (e.g., operation **(②)** in Figure 6(d) recursively alters the permissions of directory /C by first removing it and then inserting it with the new permissions). On the other hand, non-recursive updates are more complex for individual directories, especially when permissions were previously inherited from a parent but now need to be set uniquely.

Illustration of the process is depicted in Figure 6(e) where a non-recursive update to directory /A in Figure 6(c) is made. Three processes take place: Firstly, if no matching node exists for the directory pathname, a new node is created by the PMS for the directory (e.g., node A as child under node / in Figure 6(e) as shown by **(①)**). Secondly, permissions for sub-directories under the updated directory are retrieved from the MDS cluster and inserted into the permission tree—for example, retrieving permissions for D and E in Figure 6(e)—where D and E becomes children of node A **(②)**. Finally, after insertion, the permission tree is optimized using the permission merging algorithm to remove redundancy and potentially reduce the tree's depth to ensure improved performance as a result of fewer hierarchical searches—thus removing node A/E in Figure 6(e) as shown by **(③)** since it is redundant to E.

The role of the client module is to provide a platform of two ways to interact with the system. Firstly, a fast Path (low latency) allows clients to leverage the dedicated PMS for fast permission checks and retrieves the target file directly from the MDS cluster. This is ideal for applications requiring quick responses. For example in Figure 5 on page 8, when accessing /B/C/f_1, the PMS verifies permissions for /B/C **(①)**, and the MDS cluster verifies permissions for f_1 **(②)** with access to the desired file metadata from the MDS cluster granted only after passing both checks. Permission checks for the directory path and the file itself can happen simultaneously for even faster access.

And secondly a slow Path (high throughput) that allows clients to access applications that prioritize processing of large amounts of data and can tolerate slightly higher latency. This path ensures that a client can bypass the PMS and directly access the MDS cluster using a parallel permission check approach. This means the client checks permissions for each part of the file path simultaneously with the MDS cluster (e.g., checking permissions for /, /B, /B/C, and /B/C/f_1 all at once). In other words, when a client wants to access /B/C/f_1 on the slow path, this can be done by simultaneously accessing multiple components such as / **(③)**, /B **(④)**, /B/C **(⑤)**, and /B/C/f_1 **(⑥)**. Access to the metadata of /B/C/f_1 from the MDS cluster is permitted only after successfully passing all permission checks.

To efficiently manage large-scale DFS metadata among the MDS cluster, Duplex uses—as mentioned earlier—flattened metadata management through distribution of the directory tree among MDSs. In other words, Duplex distributes directory information across multiple Metadata Servers using a flattened approach that focuses on subsets of directories. This improves efficiency compared to traditional hierarchical structures. Three main steps are accomplished in the distribution of the directory tree:

- Partitioning: The directory tree is first split into individual directories.

- Subsetting: Each directory is further divided into smaller subsets (finer-grained units) to avoid activity hotspots in superdirectories. In Figure 7 (a) on page 12 a directory tree with 6 directories and 5 files is partitioned into 7 subsets. The decision of how many subsets to create for a directory depends on two factors:
  - Directory Size ($DirSize$): Total number of files and sub-directories within the directory.
  - Maximum Subset Size (S_{max}): Upper limit on the number of files a single subset can hold.

  A formula is used to calculate the required number of subsets (n) based on $DirSize$ and S_{max} ($n = \lfloor DirSize/S_{max} \rfloor + 1$). Each subset has three key aspects:
  - ID: Each subset has a unique identifier made of two parts: Pathname of the corresponding directory (to differentiate subsets from different directories); and unique number (starting from 0) to distinguish subsets within the same directory. (e.g., A_0 and A_1 for directory A).
  - Size: The total number of files in the directory is centrally stored in the first subset ($mastersubset$, $mSubset$). Each directory has exactly one $mSubset$. (e.g., size of directory A is stored in subset A_0).
  - Files: When a directory has multiple subsets, the files within it are distributed evenly across those subsets based on a Consistent Hashing (CH) function applied to their filenames. This ensures minimal data movement when the directory size changes and subsets need to be adjusted. (e.g., D in A_0 and f_1 and C in A_1 based on their CH values). Consistent Hashing helps distribute file metadata evenly across MDSs and minimizes the need to move data when directory sizes fluctuate.
- Hashing and Mapping: File metadata for each subset is distributed across MDSs using a CH (Figure 7 (B)). This, I think, helps to ensure even distribution and prevention of overloading of any single server. A second CH calculation is used to map these subsets to specific MDSs for storage. Following similar approach clients use what the authors call Double Consistent Hashing (DCH) to efficiently locate files from the MDS cluster. This involves applying CH calculations twice: once to identify the relevant subset and again to locate the specific MDS storing that data.

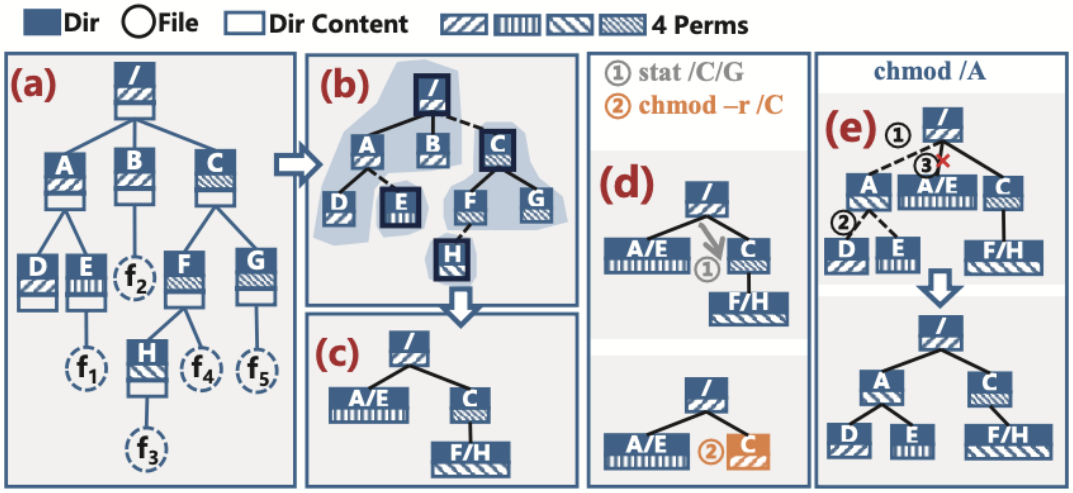

Figure 6: Permissions Management: (a) The original directory tree; (b) The initial permission tree generated from (a); (c) The merged permission tree from (b); (d) A lookup and a recursive update to (c); (e) A non-recursive update to (c).

[Don+23]

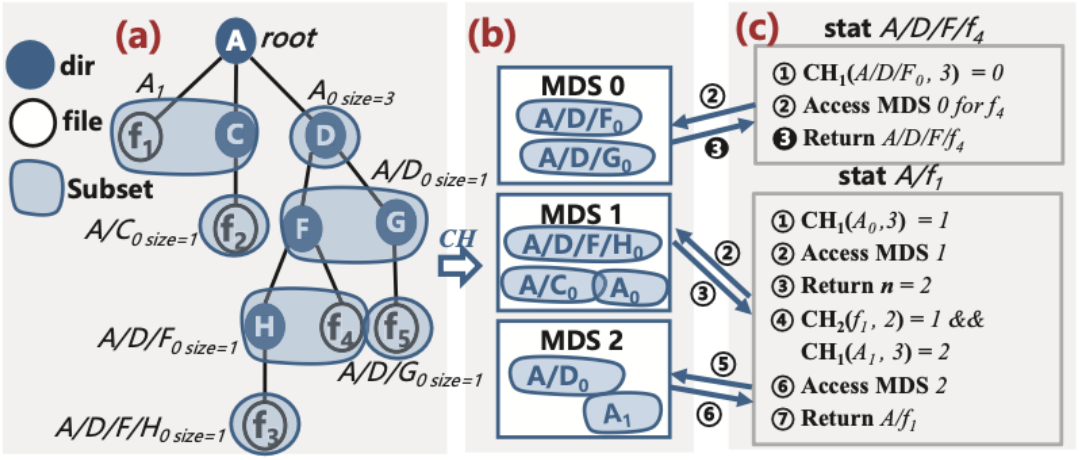

Figure 7: Flattened Metadata Management: (a) Partitioning a directory tree into 7 subsets ($Smax = 3$); (b) Distributing 7 subsets among three MDSs; (c) Accessing files from (b) through DCH.

[Don+23]

2.2.3 Evaluation of Duplex

Evaluation of Duplex against state-of-the-art metadata solutions revealed substantial improvements. Duplex, according to authors (Dong et al. 2023) [Don+23], significantly reduces the average lookup latency by up to 84% and the 99th percentile tail latency by up to 88.2% for metadata-intensive benchmarks. Furthermore, they evaluated Duplex to demonstrate an enhancement in lookup IOPS by up to 7.6 × /2.3× when compared to established systems such as CephFS and BeeGFS. This underscores Duplex's effectiveness in mitigating latency challenges and improving the overall performance of metadata-intensive operations in large-scale data processing environments.

2.3 AdaM

2.3.1 Motivation

In contemporary distributed metadata management strategies, a significant hurdle involves addressing the dynamic needs of diverse applications by efficiently mapping and migrating metadata nodes across multiple metadata servers (MDS) [Hua+23; Dai+22]. Many current approaches rely on history-based coarse-grained techniques for dynamic node reallocation, which often fall short in promptly and efficiently updating the node distribution [Hua+23]. By leveraging deep reinforcement learning, AdaM utilizes the Deep Deterministic Policy Gradient (DDPG) algorithm—a deterministic policy-based methodology—to execute timely actions, ensuring both load equilibrium and preservation of metadata locality [Hua+23; Cao+19]. AdaM, which can automatically adjust to changing access patterns, moves the most frequently accessed metadata between servers to balance load and preserve metadata locality [Hua+23; Cao+19].

The following discussions are based on the article [Hua+23] in which AdaM was authored.

2.3.2 Design and architecture of AdaM

The overarching design of Adam is to ensure metadata load balance among MDSs while preserving metadata locality through dynamic migration and subsequent distribution of hot nodes among MDSs. By considering metadata management in distributed systems as allocating metadata based on the current state for the next state, Deep Reinforcement Learning (DRL) is used to optimize the allocation process. For instance, analysis of metadata access pattern by the authors of AdaM [Hua+23] shows that most accesses are directed at small portion of metadata nodes (hot nodes). By this the authors ensured that the DRL agent act only on this nodes thereby reducing the action space.

Two key methods of DRL were employed: Deep Q-network (DQN) and Policy gradient (PG). With DQN, efficient representation of the environment (action space) from high dimensional nodes to an optimal state-action (reduced) state is achieved through deep neural network. PG is used to help strategize the agent's actions on nodes selection in different states of node environment to maximize the reward. AdaM employes DPG (deterministic PG) where the target policy is deterministic rather than stochastic. Specifically, a deterministic policy-based algorithm (Algorithm 1 [Hua+23] pg. 2846) which the authors call Deep Determinstc Policy Gradient (DDPG) is used by the agent to learn to migrate hot metadata to the appropriate location targets (MDSs).

AdaM has two main component modules (Figure 8 on page 14): Env-Monitor (EM) and Migration Manager (MM). The EM continually and dynamically monitors the environment, gathering crucial data regarding the entire metadata system. Periodically, the EM generates snapshots of the current state of the environment. Subsequently, the Migration Manager utilizes these observed states to take action by updating the mapping from active nodes to Metadata Servers (MDS).

Given that AdaM uses a machine learning model, the usual file pathnames is feature-encoded. In other words, to benefit from the DRL method to train AdaM with the pathnames as input data (metadata nodes in a tree structure), each node is represented as a vector to allow for the feature representation of the namespace tree. A simple method is to assign a position code based on the tree structure; however, this does not work well if the namespace structure keeps changing as is the case in HPC environment. Hence the

authors used network embedding technique called DeepWalk to represent/encode each node n_i in the namespace as a fixed-length vector c_i, while preserving locality (i.e nodes that are close together in the tree are such that they have similar feature encoding). The c_i acts as an identifier in the DRL process.

With this setting, EM collects two more information in addition to c_i to form an environmental statistics. That is, access frequency $f_i^{(t)}$ of hot node i at time point t and distribution/indicator $d_i^{(t)}$ which is a one-hot vector showing the MDS containing node i at time point t. This environmental statistics collected at each time point t is used to construct a state feature vector $s_i^{(t)}$ which is then used as input for the MM. The vector $s_i^{(t)}$ is a concatenation of the three environmental statistics c_i, $d_i^{(t)}$ and $f_i^{(t)}$. For a given N set of nodes the state feature vector $s_i^{(t)}$, $i \in \{1, 2, ..., N\}$ is given by $\{s_1^{(t)}, s_2^{(t)}, ..., s_N^{(t)}\}$.

The state feature vector $\{s_1^{(t)}, s_2^{(t)}, ..., s_N^{(t)}\}$ as input for the MM (the DRL agent) is designated as the state, s_t, of the system. In other words, the state of the system at time t is $s_t = \{s_1^{(t)}, s_2^{(t)}, ..., s_N^{(t)}\}$. MM's action is to map nodes to MDSs as well as indicating which MDSs are to receive hot nodes at a given time t.

The authors used a scenario where there is a distributed storage system that stores information about data (metadata) across multiple servers (M servers in total). This information is organized into distinct units called nodes (N nodes total) and with a smaller number of particularly frequently accessed nodes, called hot nodes (P hot nodes). MM dynamically shifts the P hot metadata nodes following each data access operations to the well learned[1] MDSs at time t. That is, the action of the MM is a vector $a_t = \{k_1, k_2, ..., s_P\}$ where $k_i \in \{1, 2, ..., M\}$ with $i \in \{1, 2, ..., P\}$ means that node i will be moved to MDS k_i. As mentioned earlier, the 'learn-to-migrate' process in DRL employed by MM is DDPG (Algorithm 1).

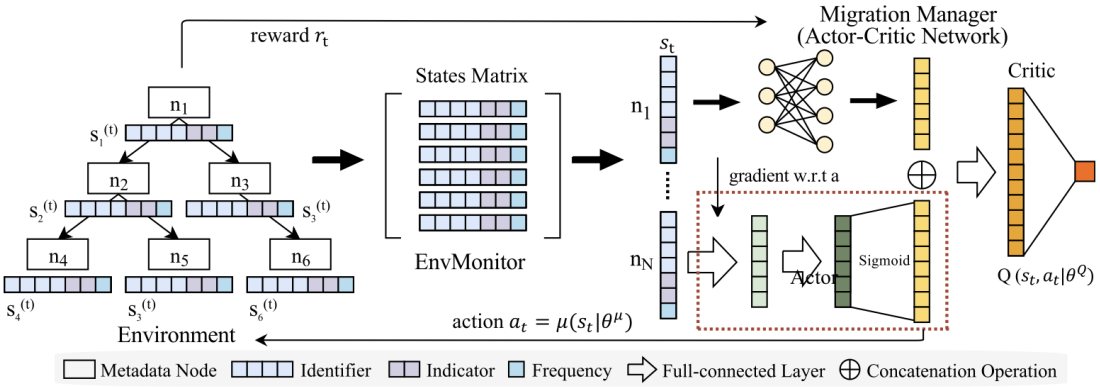

Figure 8: "A graphical illustration of AdaM.
[Hua+23]

2.3.3 Evaluation of AdaM

Real-world experiments on Amazon's cloud platform (EC2) show that AdaM outperforms other systems (hash-based mapping, static sub-tree partition, dynamic sub-tree partition and AngleCut) in terms of how fast it can answer queries, how well it balances these

[1]MDSs deemed appropriate by MM to receive a hot nodes

competing goals (balancing metadata load while preserving locality), how well it adapts to changing data access patterns, and how much it costs to migrate data to ensure load balance. AdaM was implemented using Python 3.6 with Keras on Tensorflow 1.8 as the backend during its deployment on Amazon EC2 for its evaluation.

3 Summary and Conclusion

The main objective of this report is to identify emerging trends in Parallel File Systems. The report was centred on identifying some of the recent metadata management strategies. As such Duplex and AdaM have been presented and discussed. Duplex employs innovative technique to enhance high scalability in terms of capacity and throughput while ensuring stable and low latency for latency-sensitive applications. Duplex also employs innovative partitioning and indexing technique based on flattened metadata management that ensures load balance and eliminates issues of activity hotspots in super-directories. Employing a dedicated server, Duplex ensures file permissions are handled efficiently. AdaM on the other hand is designed to offer services for managing metadata load balance while preserving metadata locality by employing Deep Deterministic Policy Gradient method to migrate hot metadata nodes across multiple MDSs efficiently (minimizing migration cost). Evaluation of both Duplex and AdaM against notable file systems shows significant metadata performance metrics over existing services. Thus this report highlights the critical role of efficient metadata management in optimizing system performance.

References

[Bor+23] Arnav Borkar et al. "Does Varying BeeGFS Configuration Affect the I/O Performance of HPC Workloads?" In: *2023 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*. IEEE. 2023, pp. 5–7.

[BPT22] Francieli Boito, Guillaume Pallez, and Luan Teylo. "The role of storage target allocation in applications' I/O performance with BeeGFS". In: *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2022, pp. 267–277.

[Cao+19] Shiyi Cao et al. "AdaM: An adaptive fine-grained scheme for distributed metadata management". In: *Proceedings of the 48th International Conference on Parallel Processing*. 2019, pp. 1–10.

[Car+09] Philip Carns et al. "Small-file access in parallel file systems". In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE. 2009, pp. 1–11.

[Dai+22] Hao Dai et al. "The state of the art of metadata managements in large-scale distributed file systems—Scalability, performance and availability". In: *IEEE Transactions on Parallel and Distributed Systems* 33.12 (2022), pp. 3850–3869.

[Don+23] Chao Dong et al. "Low-Latency and Scalable Full-path Indexing Metadata Service for Distributed File Systems". In: *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE. 2023, pp. 283–290.

[Dub19] Viacheslav Dubeyko. "Comparative Analysis of Distributed and Parallel File Systems' Internal Techniques". In: *arXiv preprint arXiv:1904.03997* (2019).

[EHM17] Akram Elomari, Larbi Hassouni, and Abderrahim Maizate. "The main characteristics of five distributed file systems required for big data: A comparative study". In: *Adv. Sci. Technol. Eng. Syst* 2 (2017), pp. 78–91.

[Gar+23] Felix Garcia-Carballeira et al. "A new Ad-Hoc parallel file system for HPC environments based on the Expand parallel file system". In: *2023 22nd International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE. 2023, pp. 69–76.

[Hua+23] Xiuqi Huang et al. "An Adaptive Metadata Management Scheme Based on Deep Reinforcement Learning for Large-Scale Distributed File Systems". In: *IEEE/ACM Transactions on Networking* (2023).

[Lia+21] Xiaojian Liao et al. "Crash consistent non-volatile memory express". In: *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 2021, pp. 132–146.

[Man+22] Nicolau Manubens et al. "Performance Comparison of DAOS and Lustre for Object Data Storage Approaches". In: *2022 IEEE/ACM International Parallel Data Systems Workshop (PDSW)*. IEEE. 2022, pp. 7–12.

[Pau+20] Arnab K Paul et al. "Understanding hpc application i/o behavior using system level statistics". In: *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE. 2020, pp. 202–211.

[SH02] Frank Schmuck and Roger Haskin. "GPFS: A Shared-Disk file system for large computing clusters". In: *Conference on file and storage technologies (FAST 02)*. 2002.

[Wik17] Lustre Wiki. *Lustre File System Overview (DNE) lowres v1.png*. `https://wiki.lustre.org/File:Lustre_File_System_Overview_(DNE)_lowres_v1.png`. Accessed: 2024-01-01. 2017.

[Xu+13] Quanqing Xu et al. "Efficient and scalable metadata management in EB-scale file systems". In: *IEEE Transactions on Parallel and Distributed Systems* 25.11 (2013), pp. 2840–2850.

[Zhe+21] Qing Zheng et al. "DeltaFS: A scalable no-ground-truth filesystem for massively-parallel computing". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021, pp. 1–15.

[Zhe+22] Huihuo Zheng et al. "HDF5 Cache VOL: Efficient and Scalable Parallel I/O through Caching Data on Node-local Storage". In: *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2022, pp. 61–70.

[ZL22] Wei Zhang and Yunxiao Lv. "A Connectivity Reduction Based Optimization Method for Task Level Parallel File System". In: *2022 IEEE 10th International Conference on Computer Science and Network Technology (ICCSNT)*. IEEE. 2022, pp. 85–88.