The State of the Art of Metadata Managements in Large-Scale Distributed File Systems — Scalability, Performance and Availability

Hao Dai, Yang Wang, Kenneth B. Kent, *Senior Member, IEEE*, Lingfang Zeng, and Chengzhong Xu, *Fellow, IEEE*

Abstract—File system metadata is the data in charge of maintaining namespace, permission semantics and location of file data blocks. Operations on the metadata can account for up to 80% of total file system operations. As such, the performance of metadata services significantly impacts the overall performance of file systems. A large-scale distributed file system (DFS) is a storage system that is composed of multiple storage devices spreading across different sites to accommodate data files, and in most cases, to provide users with location independent access interfaces. Large-scale DFSs have been widely deployed as a substrate to a plethora of computing systems, and thus their metadata management efficiency is crucial to a massive number of applications, especially with the advent of the Big Data age, which poses tremendous pressure on underlying storage systems. This paper reports the state-of-the-art research on metadata services in large-scale distributed file systems, which is conducted from three indicative perspectives that are always used to characterize DFSs: high-scalability, high-performance, and high-availability, with special focus on their respective major challenges as well as their developed mainstream technologies. Additionally, the paper also identifies and analyzes several existing problems in the research, which could be used as a reference for related studies.

Index Terms—High-availability, high-performance, high-scalability, large-scale distributed file system, metadata management

1 Introduction

With the rapid growth of Big Data applications, current computing architectures pose new challenges to storage systems in terms of scale and performance requirements. For example, in Big Data applications such as those in health [1], traffic [2], and financial [3], the scale of data is usually in the order of terabytes (TB), petabytes (PB), or even exabytes (EB). Therefore, a large quantity of storage resources is needed to store and manage the data.

Moreover, a large number of data analysis tasks require low-latency access to the data across different storage servers, which also poses high requirements for the reading and writing speeds of the storage system. This is especially important to distributed file systems (DFSs) as it has become one of the most effective ways to manage Big Data storage by its simplicity and versatility. Some typical application scenarios of DFS are Big Data processing (e.g., *HDFS* [4], *GFS* [5]), high performance computing (HPC) (e.g., *Lustre* [6], Spectrum Scale [7], *BeeGFS* [8]), and Web applications (e.g., *CephFS* [9], *GlusterFS* [10]). Therefore, to support the massive data storage and computation requirements, as well as hardware optimization, efficient data organization and management is also one of the key technologies that must be considered.

- *Hao Dai and Yang Wang are with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China. E-mail: hao.dai@siat.ac.cn, yangwang5@msn.com.*
- *Kenneth B. Kent is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada. E-mail: ken@unb.ca.*
- *Lingfang Zeng is with ZJ Lab-Enflame Joint Innovation Research Center, Zhejiang Lab, Hangzhou, Zhejiang 310058, China. E-mail: zenglf@zhejianglab.com.*
- *Chengzhong Xu is with the Faculty of Science and Technology, University of Macau, Macau 999078, China. E-mail: czxu@um.edu.mo.*

Manuscript received 13 Sept. 2021; revised 17 Apr. 2022; accepted 22 Apr. 2022. Date of publication 4 May 2022; date of current version 26 July 2022.
This work was supported in part by Third Xinjiang Scientific Expedition Program Under Grant 2021xjkk1300, in part by the National Natural Science Foundation of China under Grant 61672513, in part by Zhejiang provincial "Ten Thousand Talents Program" under Grant 2021R52007, in part by the Center-initiated Research Project of Zhejiang Lab under Grant 2021DA0AM01, in part by the Natural Science and Engineering Research Council of Canada, Lockheed-Martin Cybersecurity Fund under Grant LMCRF2020-02, and also in part by Mitacs under Grant IT24602.
(Corresponding authors: Yang Wang and Lingfang Zeng.)
Recommended for acceptance by A. Sussman.
Digital Object Identifier no. 10.1109/TPDS.2022.3170574

1.1 Metadata Workloads

File system metadata is a special kind of system data that describes the structural characteristics of file system, not only including its type, size, state (superblocks), but also maintaining the access rights, owner, creation/update time as well as the data block information with respect to each file and directory. The play of a file system typically involves frequent operations on the metadata to facilitate its normal data access operation. Let us take the example in [11] to illustrate, in which a one-block sized file (e.g., `/foo/bar`) from an inode-based file system is opened, read and then closed. The entire process is depicted in Table 1 where a bunch of read operations caused by the open take place to locate the inode of the file `bar`, whereby the file read operation is performed by first consulting the inode, then reading the block, and finally updating the inode's last-accessed-time field with a write. Note that this example

1045-9219 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
File Read Timeline for a One-Block Sized File /foo/bar [11] (Time Increasing Downward)

	Metadata			Data		
	root inode	foo inode	bar inode	root data	foo data	bar data
open (bar)	read					
		read		read		
					read	
			read			
read()			read			
						read
			write			

is just reading in a small file from disk, which involves a relatively large number of metadata operations. The life would be even worse when writing out a file. With this example one can understand that 50% to 80% of accesses to the file system are made on metadata although it only accounts for a relatively small portion of 0.1% to 1% of the entire data space [12], [13]. As file systems have evolved, research on metadata has constantly been conducted along the way. For example, Outsterhout *et al.* [13] analyzed characteristics of the UNIX 4.2 BSD file system access patterns by recording user-level activities, and found that more than 50% of all file accesses are made to the metadata. Agrawal *et al.* [14] also conducted similar research, where they collected and analyzed the metadata access snapshots of more than 10,000 file systems from Windows Desktop machines during the period between 2000 and 2004. They found some file types have a strong temporal tendency in terms of popularity and namespace usage, as well as file size changes. A similar phenomenon has also been discovered in recent work by Leung *et al.* [15]. Based on the trace data from *NapApp*, they analyzed how the metadata is searched in large-scale storage systems from two perspectives. From the user's point of view, the behavior of a metadata search is mainly reflected in the following aspects: 1) for the refinement of results, more than 95% of searches are made for multiple metadata attributes; 2) about 33% of searches are limited to a related area of the namespace, reflecting the fact that files are usually organized by a semantic organization; and 3) nearly 25% of searches that users think are important involve multiple versions of metadata, mainly from "back-in-time" searches that users make to understand the file access trends.

Alternatively, from the perspective of the accessed metadata, its patterns also reflect the characteristics of the user's search behavior: 1) *Spatial locality:* refers to the attribute values of files gathered in the namespace. For example, John's files are mostly located under the `/home/john` subtree, rather than being scattered throughout the namespace; 2) *Skewness:* The value of metadata has a high degree of skewness, meaning that the distribution of these values is not symmetrical, and some prevalent metadata values occupy most of the value space. For example, 80% of files have the most common 20 `ext` and `size` values, and these phenomena are also documented in other pieces of literature [14], [16].

Recently, Xiao *et al.* [17] performed a more detailed analysis to characterize the skewness of the structure and operational distribution of file system namespaces on a wide range of storage platforms, including cloud storage. They found that the file system namespace shows a heavy-tailed distribution for the size of the directory, which is represented by several small directories with a small number of relatively large directories. For example, among the 60 file systems being examined, 90% of the directories contain fewer than 128 directory entries. Another valuable finding is that large directories continue to grow as the storage system capacity grows. In addition to the directory files, the normal file size also shows a similar distribution. In the mainstream file system, 64% of the media files are smaller than $64\ KB$, which means that the majority of the system is small files that are less than several hundred KB. This observation is also true even in large-scale cluster file systems for Big Data processing [18]. As for the depth of the directory tree, unlike the size of the directory, it does not grow with the increase of system capacity. In the system under consideration, the vast majority (about 90%) of the directory tree does not exceed the depth of 16. This phenomenon is observed in other studies [19], [20]. Unlike the skewness of metadata values of interest to Leung *et al.*, Xiao *et al.* studied the skewness of metadata manipulation, and they found that in all the trace files being examined, only one or two operations dominated: namely the `open` operation and the `readdir` operation.

The performance of metadata access is essential to the design and implementation of the file system [21]. As such, the exploration of the metadata and its access characteristics deepens our understanding of metadata management and drives the research related to the metadata services.

1.2 Metadata Management Architecture

With the developments of the architecture of DFS in the past decades, its corresponding metadata management has been undergoing continuous improvements and optimization. Traditionally, the metadata and data are usually placed on the same storage servers. Regarding access efficiency, the metadata was usually stored physically close to the data it describes, which implies that no concept of "metadata server (MDS)" was involved in the past [22], [23], [24]. With the rapid growth of data in DFS, researchers found that the scalability and performance of metadata services tend to be the bottleneck in such an architecture. Therefore, Gibson *et al.* [25] proposed a separate metadata server architecture, regarded as a pioneer in separating the metadata server. Since file operations, as well as data consistency across the DFS, incur massive operations performed on metadata, the performance of metadata operations is critical to the overall performance of DFS. However, this problem is not that serious in early-day DFSs as they were usually small sized, and thus the data and metadata are typically stored in the same machine, which is a pseudo non-separated architecture. Unfortunately, as the size of the DFS increases, this non-separated architecture suffers from significant performance degradation as more servers need to be involved for metadata retrieval, resulting in a tremendous amount of overhead. To address this problem, the metadata is separately stored from the data itself in modern DFSs, which leads to a new kind of separated architecture. With this architecture, we can access all the data by retrieving from the MDS separately, dramatically improving the speed of MDS operations, which in turn

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

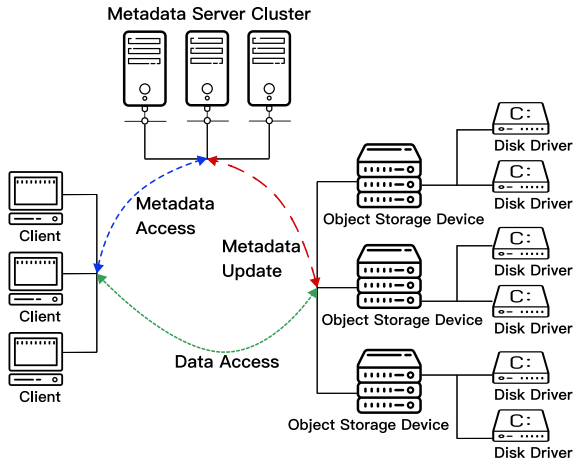

Fig. 1. A typical DFS structure. It includes three parts: MDS, OSD, and interaction with clients.

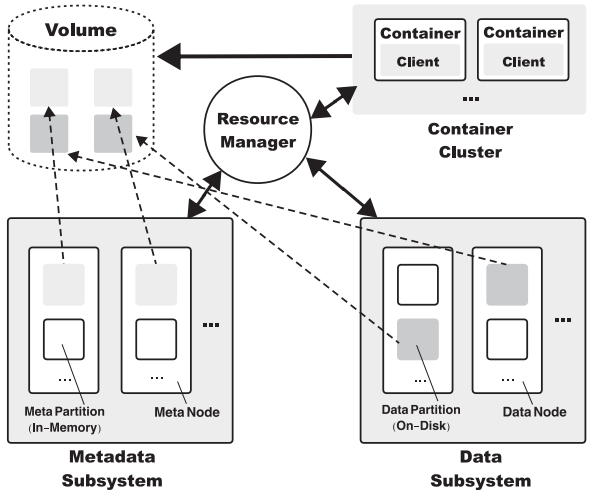

Fig. 2. The architecture of *CFS*. It consists of four components: metadata subsystem, data subsystem, container cluster, and resource manager.

enhances the scalability and performance of DFS. Given these merits, the mainstream DFS nowadays often adopts the separated architecture to manage its metadata, such as *CephFS*, *GFS*, and so on. To sum up, as opposed to the traditional structure, the new architecture makes it easier to scale up and improve the performance of the metadata server.

The current mainstream large-scale DFSs adopt a structure that separates the management of metadata from file content (data) as shown in Fig. 1. The structure mainly consists of three parts: the client file system, metadata server, and object/data storage device (OSD/DSD). The functions of each component are: the client is to provide a file system access interface; the OSD/DSD holds all data of files; and the duty of the MDS is to store and manage metadata.

The MDS system generally uses a single MDS or an MDS cluster to maintain the file system's global namespace and the physical view of the file data storage. The metadata information provides the client with services in an out-of-band manner. At the same time, the MDS system also manages the data storage devices. When a client needs to read or write data, it would first communicate with the MDS to retrieve the corresponding metadata. Afterwards, the client transfers the data from the OSD/DSD according to the metadata. This data transmission only occurs between the client and the OSD/DSD. This architecture effectively reduces the workloads of the MDS. By scale-out, the aggregated I/O bandwidth of the DSD cluster can be fully utilized to improve the overall performance of the I/O system.

Since the performance of MDS is a crucial factor of a DFS, in addition to the separate design, many other research projects, such as *CephFS* [9] and *HopsFS* [26], are proposed to optimize the MDS's performance by leveraging a distributed MDS built on the database. The exploitation of the database not only provides the MDS with a convenient and concise read/write interface, but also enables the metadata to achieve high performance by using specific index structures (e.g., log-structured merge-tree (*LSM-Tree*) [27], range-query optimized persistent ART (*ROART*) [28], etc).

To preserve the directory locality, modern mainstream POSIX-compliant DFSs usually retain inode and dentry on the same server. Some studies have also explored whether the inode and dentry can be placed on different servers, such as *CFS* [29], which is a typical DFS with a separate storage cluster to store and accommodate the metadata based on the memory usage. To efficiently achieve the utilization-based metadata placements, the inode and dentry of the same file may be distributed across different metadata nodes in the *CFS*. The architecture of *CFS* is shown in Fig. 2, where the metadata subsystem consists of a set of meta nodes, each node is composed of hundreds of partitions. The whole sub-system thus can be regarded as a distributed in-memory data store of metadata.

GlusterFS, by contrast, is an earlier, widely-used DFS without using MDS, which is proposed in 2006 [10]. The main idea of *GlusterFS* is to locate data by calculating the hash values of the file name and path instead of using the MDS for metadata management, which means the location is high-efficiency once both file name and path are given. *GlusterFS* performs efficiently with the known file name and path, while the corresponding performance is worse in other cases.

1.3 Challenges of Metadata Management

Some DFSs in the early stage, such as *GFS* [5], *HDFS* [4], and *BWFS* [30], adopted a single MDS architecture. In the small-scale case, a single MDS always shows advantages in reducing the communication cost of metadata access and maintaining the metadata consistency with low overhead. However, with the development of cloud computing and Big Data, the single MDS architecture is suffering from considerable challenges in terms of the scalability, performance, and availability of storage systems. For example, in a massive storage system at the EB-scale, the metadata can be increased to the PB-scale [31], which leads to a high scalability requirement of the MDS system.

In many applications, metadata is usually highly shared and accessed concurrently, and low-latency guarantees are required for high-frequency access metadata [32], which in turn requires the high performance of the MDS system. Finally, as a crucial part of file read/write, the breakdown of a single MDS greatly declines the availability of services, which is the basic guarantee for providing data services.

Considering these factors, an intuitive idea is to adopt MDS clustering technology to attain high scalability, high

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

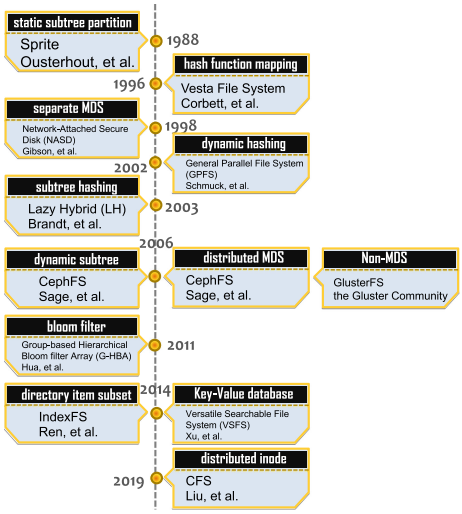

Fig. 3. Some key milestones in the development of MDS.

efficiency, and high availability. Therefore, many enterprises and research institutes conduct extensive and in-depth research on the requirements of high scalability, high performance, and high availability of MDSs based on the MDS cluster architecture (e.g., *CephFS* [9], *Lustre* [6], [33], [34], [35], *CFS* [29]). We made a timeline for some key technologies in the development of MDS. As shown in Fig. 3, with the emergence of separated MDS, significant works have been conducted to improve the performance and scalability optimization of MDS and novel optimization techniques have been proposed, accordingly. Note that only some key representative works are listed in the figure.

This article summarizes and sorts out the state-of-the-art in this area, and analyzes the existing problems to identify prospects for future development, hoping to provide related research with valuable references. It is worth noting that this article is definitely not a summary of exhaustive technologies. Some hot topics including techniques for optimizing the metadata management for new storage media (such as solid state drive (*SSD*) [36], [37], non-volatile memory express (*NVMe*) [38] and phase change memory (*PCM*) [39]) or AI (e.g., deep learning) are not covered. Since these topics can be summarized separately, interested readers can refer to relevant literature [40], [41], [42], [43], [44], [45].

The remainder of this work is organized as shown in Fig. 4, where the state-of-the-art of the metadata management technologies is surveyed in terms of scalability, performance, and availability, respectively. In particular, Section 2 studies and compares the metadata management methods from the perspective of high scalability, while Section 3 overviews the techniques regarding the performance of metadata managements, including caching and replication, index and retrieval, and value-add methods as well. Section 4 discusses the methods for the high availability of metadata in large-scale DFS. Finally, Section 5 summarizes some future research challenges, and Section 6 concludes this review.

2 Highly Scalable Technology for Metadata Services

High scalability is one of the main purposes of adopting the MDS cluster architecture, and it is also a hot issue in DFSs in recent years. The main challenge is how to divide the entire namespace of the file system into slices and distribute them evenly among multiple MDSs. In addition to an initial division, it also aims to effectively deal with the changes in access loads and the elastic increase and decrease of MDSs. The highly scalable technology of metadata management can be classified from many angles. This paper summarizes and compares the existing technologies into *static* and *dynamic* spatial partitioning methods from the perspective of the organizational query of metadata.

2.1 Static Space Division Methods

The so-called static space partitioning method refers to the case that division of the metadata space only depends on some information related to its structure, such as path name, subtree size, etc., regardless of the actual load changes. *Subtree partitioning* [46], [47] and *hash-based mapping* [6] are two common basic methods. Consequently, there are many different variants based on these two methods for different scenarios and purposes [48], [49], [50].

2.1.1 Static Subtree Partitioning Methods

The static subtree method is a relatively simple partitioning method, which is often used in early-phase DFSs, such as *NFS* [51], *AFS* [52] and Coda [53], etc., as well as some recent massively distributed file systems, such as *Hadoop Federated HDFS* [54], etc. This approach typically leaves it up to the system administrator with how to partition the hierarchy of the directory and assign each "shard" (usually a subtree) to the specified MDS. This approach is highly efficient when the metadata of a file needs to be repeatedly accessed by a relatively small number of servers. Whereas the main shortcoming is that it can not dynamically cope with the unbalanced metadata workloads. That is, the "hot spots" caused by uneven accesses are not well handled. For instance, when a group of files in the same directory subtree is frequently accessed in a short interval, the server where they are located would be extremely busy, thus becoming a performance bottleneck of the entire system. To address this issue, the subtree replication will play a particular mitigation role in some cases. Still, it may also increase the cost of storage and the overhead of maintaining copy consistency. Additionally, there are some difficulties with this division method when increasing or decreasing MDS. Because of the re-planning of namespace partitioning, some subtrees need to be migrated among MDSs, resulting in the decline of performance.

2.1.2 Hash Function Mapping Methods

A hash-based method is to apply a *hash function* to the file name to locate the file's MDS. This approach not only reduces the workload imbalance between the MDSs, distributing service requests evenly across MDS clusters, but also allows the clients to retrieve the metadata directly. For example, *Vesta* [55] and *zFS* [56] utilize a hash of file path names to locate the data and servers. The main drawback of this approach is the lack of data locality. In addition, the modification of the path name would lead to the migration of massive metadata among the MDS cluster, increasing the network workload. Like the static hashing method, access control and the path query also bring lots of network overhead due to the

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

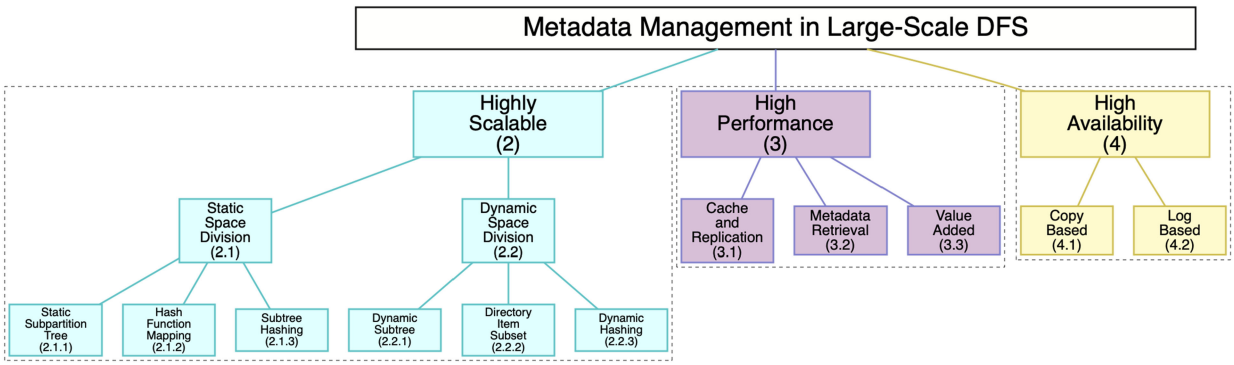

Fig. 4. The state of the art of metadata managements for large-scale distributed file systems.

prefix directory of the queried file being stored on a different server. Furthermore, Note that the hash function will vary with the number of servers, resulting in massive migrations as well.

At present, some improvements have been proposed in the literature [57] for these issues. Unlike the traditional methods based on file path name hashing, Xu *et al.* [57] suggested using a hash function to assign an entire directory subtree under a nested point to the same MDS, which is beneficial to maintain the locality. To this end, they designed a new localized hash function based on *simhash* [58], [59]. The main idea of the well-designed function is to hash each directory's sub-directory of a path (2-byte encoding). After that, the hash values of the inode are linked to the same file, forming a 48 B hash value instead of hashing the entire path name. In addition to maintaining the locality, this approach also makes it free to name new files and directories. For the migration of files among directories, this approach can swiftly update the hash value.

Notably, as far as handling data volume at ultra large scale is concerned, these hash-mapping based methods are more favored in industry. For instance, *Tectonic* – an exabyte-scale distributed file system proposed by Facebook [60] – leverages hash-partitions of each metadata layer to avoid the hotspot issue. Not surprisingly, *Tectonic* still suffers trade-offs and compromises arising from data updates. To effectively reduce the data updates and migrations caused by modifying the directory attributes, some researchers adopted an alternative strategy to separate the directory path attribute from the directory object. The advantage of this approach is to reduce the overlapping cache of the prefix directory, thus improve the utilization and hit rate of the MDS cache. Moreover, it also enhances the storage locality of the directory so as to decline the disk I/O operations. Nevertheless, owing to the increased overhead of retrieving the directory index server, this approach may negatively impact the availability of the entire system.

2.1.3 Subtree Hashing Methods

Subtree partitioning enhances support for the locality of metadata accesses, but it still suffers from the imbalance of storage loads. In comparison, the hash-based method achieves better load balancing at the expense of locality support. Therefore, the hybrid subtree hashing approach integrates these two aspects to take advantage of each other. The hybrid method usually leverages a hash function to achieve a balanced distribution of metadata among the MDSs and it supports semantic query and control of directories and files based on a hierarchical directory structure. It is generally accepted that locality and load balancing are often contradictory to each other by their very nature. However, as shown in Fig. 5, the subtree + hashing approach adopts a hybrid method to strike a good balance between these two metrics. It first achieves good virtual locality by partitioning the namespace of DFS into subtrees, and then realizes good load balancing by hashing the partitioned subtrees into different MDS servers in physics.

Based on this idea, Brandt *et al.* [48] proposed a hybrid strategy called *lazy hybrid* (*LH*). On the one hand, the method adopts a metadata lookup table to decouple the hash value of the path from the position of the metadata, so as to attain the elastic management of the server resources. On the other hand, an *access control list* (*ACL*) with dual entries makes the extra permission operations avoidable. Additionally, a unique character of *LH* is to adopt a hybrid mechanism to postpone the migration of metadata until it is visited after being modified, thus declining the burst network traffic between MDSs. However, although *LH* distributes the namespace tree over different servers by the hash function, it ignores the impact of single oversized directories. Excessively large or skewed directories seriously break down the scalability of the file system, so Swapnil *et al.* proposed *GIGA* + [61], a metadata management service based on a directory partition and hash method. This approach achieves a further

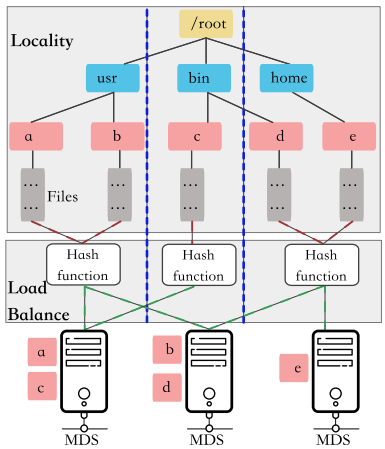

Fig. 5. The hybrid subtree hashing approach.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

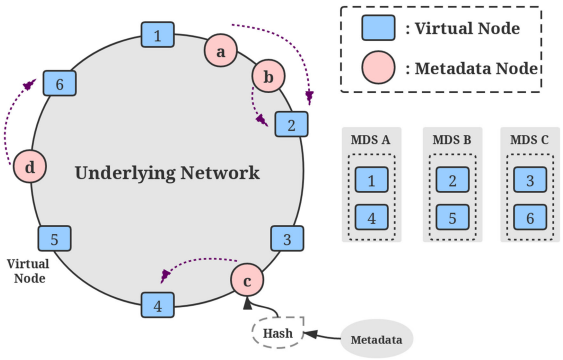

Fig. 6. The Architecture of *AngleCut*. The dotted line with an arrow represents the metadata node is assigned to the virtual node.

trade-off between locality and load balancing, providing a more fine-grained approach to namespace tree management.

Despite *LH* and *GIGA+* having many advantages, they still suffer from the network overhead caused by metadata migration. Xu *et al.* [57] proposed a similar system *DROP* for metadata management of EB-scale storage systems. *DROP* uses hash functions to distribute metadata. Meanwhile, it maintains a global directory to execute the directory attribute dependent operations, such as reading and writing permissions. As mentioned above, to achieve these advantages, they used a hash function, called *localhold*, and implemented a histogram-based dynamic load balancing strategy to overcome the load imbalance caused by the hash function.

Although *DROP* has significant improvements in both efficiency and scalability to serve EB-scale file systems, the augmented hash keys are required to be maintained in metadata storage, leading to large additional space requirements. To address this issue, Gao *et al.* presented *AngleCut* [31], [62], a ring-based metadata management policy for large-scale DFSs to partition the metadata namespace tree. For the most part, we tie a distributed file system into the namespace of the local file system by mounting the remote directory tree with a particular point in the local directory tree. Therefore, the namespace organization structure of DFS is tree-like, and we use the namespace tree to represent the whole namespace. Similar to *DROP*, a novel locality preserving hashing function is used to hash the metadata namespace tree onto a linear key-space in *AngleCut*. Subsequently, the angle value of the metadata nodes will be calculated to allocate the nodes to a Chord-like ring.

The system architecture of *AngleCut* is shown in Fig. 6 where `Node a-d` represents the metadata nodes whose positions are calculated by a specific hash function, and `Node1-6` stands for virtual MDSs, which are also mapped to their respective positions on the ring via the same hash function. According to the Chord definition [63], the metadata is allocated to the nearest MDS on the hash ring in a clockwise direction. As mentioned above, maintaining the hierarchy locality of the metadata namespace tree is an essential part of MDS, so that the locality preserving hashing (*LPH*) algorithm is used to assign "close" metadata nodes to the same MDS as frequently as possible.

Afterward, *AngleCut* assigns the virtual servers to the real MDSs through the cumulative distribution function (CDF) of metadata access frequency. As shown in Fig. 6, server `Node1` and `Node4` are managed by MDS `A`. Based on this design, *AngleCut* not only improves the system load balancing degree and the system scalability but also reduces the use of extra unnecessary space.

2.2 Dynamic Space Division Methods

As the name implies, the dynamic space partitioning method assigns the global namespace to different MDSs relative to the static space partitioning method. When the workload changes, the dynamic load balancing mechanism is used to redistribute the metadata at different granularities across the MDS clusters. Compared with the aforementioned methods, the dynamic space partitioning method exhibits a comparative advantage in maintaining access locality, the adaptive load change and the flexible use of resources, and thus it attracts wide interest in industry and academia. Currently, the mainstream large-scale DFSs, such as *Ceph* [9], *GFSII* [64], *GlusterFS* [5], [10], and *Luster* [65], support dynamic metadata management in various forms. For the dynamic subtree partitioning method, research mainly focuses on the issues such as the load balancing technology and related resource flexibility.

2.2.1 Dynamic Subtree Methods

The load balancing issue is to migrate the workloads of those heavily loaded nodes to the lightly loaded nodes. In this phase, the load balancing is usually performed in a minimum unit of the subtree to maintain the spatial locality of accesses. *Ceph* is a typical system that leverages dynamic subtree partitioning to achieve load balancing [9]. Specifically, each MDS uses an exponential time decay counter to measure the frequency of the metadata accesses according to the directory structure. Any operation on a particular metadata item would increase the access count from the root to all the directory nodes along the path to the accessed metadata node, thus each MDS would provide a weighted directory tree to characterize the most recent metadata load distribution. The load value of the MDS is compared among the MDSs by periodically sending the heartbeat information to each other, and the heavily loaded node can then migrate the identified subtree to the lightly loaded node, at the same time, a new node has become an authoritative node of the subtree. In *Ceph*, the authority node of a directory entry is defined by the path name of the directory entry or the hash value of its inode number. As the directory grows, or the access frequency increases, it can be hashed to other nodes. Otherwise, as the directory shrinks, or the access frequency decreases, different parts of a directory subtree can be aggregated from multiple servers to one server, which provides support for flexible resource management.

Ceph adopts a placement algorithm, called controlled replication under scalable hashing (*CRUSH*) [66], to conduct a structured mapping from objects to a hierarchical map, which is composed of a cluster of OSD nodes, and then distribute the objects evenly across available OSDs. Although *Ceph* has a respectable performance in load balancing, it suffers from uncontrolled data migration after expanding the cluster nodes due to the flaws of the *CRUSH* algorithm. To achieve controllable data migration in *Ceph*, Wang *et al.* [67] proposed a novel extension $M_{AP}X$ to the *CRUSH* algorithm

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

by introducing an extra time-dimension mapping from object creation time to cluster expansion time, where the object creation timestamps can be maintained as a sort of higher-level storage metadata. Experimental results show that the MAPX-based migration-free *Ceph* outperforms the *CRUSH*-based *Ceph* by up to $4.25\times$ in terms of the tail latency.

The *Blue Whale Metadata Management System* (*BWMMS*) is an MDS cluster technology developed for EB-level storage [68]. With the integration of *pNFS* [69] and the *Blue Whale device file system* (*BWFS*), *BWMMS* implements a metadata distribution strategy granular to the directory. Unlike the weighted subtree method in *Ceph*, the *BWMMS* partitions the global namespace and specifies the total number of storage directories for each partition. Once the partition is filled, the new directory will be distributed to other metadata subvolumes. The selection of the metadata sub-volume is performed in a *Round-Robin* manner to distribute the metadata to respective MDSs. This idea is somewhat similar to the way *GPFS* handles the diffuse column functions of large directories. The difference is that the contents in a perfect score are not split. The *BWMMS* does not use the hashed path name to locate the MDS. Instead, it maintains an in-memory MDS Map data structure to store the mapping relationships between the metadata sub-volume and the MDS. Although this method does not depend on the path, it introduces the control problem brought by the MDS Map. Another problem with the *BWMMS* is that partitioning of the namespace only considers the size of the directory entries without considering the frequency of accesses to the directory entries, resulting in an access hotspot problem.

Since each MDS has its own metadata balancer, a heartbeat mechanism is needed to maintain consistency in the MDS cluster. The balancer needs to send inodes to other MDS nodes when rebalancing the loads so that the trade-off of performance between distribution and locality needs to be discussed. The performance may become worse and the number of requests will increase when the metadata is distributed unnecessarily, because the MDSs need to request the remote metadata from other MDSs for file system operations, which means, the dynamic subtree leads to the complexity of deciding how to migrate the resources based on the characteristics of said resources. In terms of exploring locality vs. distribution with respect to the performance and stability, and gain insights into the true bottleneck, Sevilla *et al.* [70] presented a general programmable metadata balancer for *CephFS*, which is called *Mantle*.

2.2.2 Directory Item Subset Methods

As opposed to the systems mentioned above, for metadata access characteristics, *IndexFS* [49] adopts a hierarchical structure to design the MDS clusters, processes the metadata and small files in a unified manner, and considers their out-of-core storage optimization. Each directory is built on a random primary server, and the directory entries for all files are also stored on the same server. When the size of an increasing directory exceeds a predetermined threshold, *IndexFS* will continue to divide it incrementally and randomly store subsets in a backup MDS. To achieve load balancing, the choice of alternative MDS is based on the principle of "Power of Two Choices" [71], which detects two randomly selected servers and places the subset of directories in a server with a relatively small number of directories. It can be seen that, unlike the system based on the subtree partitioning, the metadata of the *IndexFS* system is distributed in a subset of directory entries stored in the MDS. For background storage optimization, *IndexFS* utilizes the *LSM-Tree* [27] — a technology commonly used in current Key-Value Store systems — to represent and store the metadata and small file data. Each server stores and manages part of the file system metadata and uses *LevelDB* [72] to package metadata and small file data into a flat large file. Then the flat file will be formatted as a sorted string table (*SSTable*) [73] and stored in the shared cluster file system. In this way, *IndexFS* can serve the metadata retrieving by the Key-Value interface of *LevelDB*. Despite the scalability of metadata accesses having been improved, studies have shown that this approach strongly depends on exploiting the locality of the cache to reduce the cost of path queries [17], [74]. Therefore, in the case of divergent path access, cache updates will lead to significant hotspot issues with this approach.

2.2.3 Dynamic Hashing Methods

GPFS is a distributed file system designed by IBM for shared storage [75], which implements dynamic expansion of directory organization at the block level by means of *extensible hashing*. The entries of a directory are stored on disk blocks, which are addressed by the low-order n bits of the directory name's hash value, where n depends on the size of the directory. Specifically, when a new directory entry is created for a directory disk block, the disk block is divided into two parts if it is full. The logical block number of the new disk block is implemented by the $n+1$ bit position '1' of the old block number. The directory entry is the migration of the directory entries whose hash value of the $n+1^{th}$ bit in the old disk block is '1'. The $n+1$th bit of the logical block number of the old disk block is '0,' and its directory entry is also composed of the remaining directory entries whose $n+1$th bit has a hash value of '0'. Thus, regardless of the size and structure of the catalog file, a query typically requires only one directory block query. Because it is shared storage, all nodes can participate in metadata management including the namespaces. The consistency of metadata is guaranteed by token control between the nodes. Based on the same idea, some researchers applied the column-hash function to the numbering of the MDS set to achieve load balancing in the elastic server resources. However, this method does not provide good support for the locality of metadata accesses due to the randomness of the load splitting.

In the hierarchical metadata management based on the *group* concept proposed by Hua *et al.* [76], the splitting and merging of the group also adopts a similar method. Similarly, the *AbFS2* [77] metadata model combines a hash/table and B+ trees, which is based on the same idea. However, these approaches only aim to balance storage space but lack dealing with the hotspots. To address the hotspot issue, *AngleCut* [31], [62] adopts a periodic random walk to estimate the cumulative distribution function (CDF) of metadata access frequency. As the access frequencies of nodes change over time, the hotspot caused by the unbalanced workload of MDS can be captured by CDF so that *AngleCut*

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Summary of Existing DFS With Different Methods of Namespace Slice

Method		Reference
Static Space Division	Static Subtree Partitioning	*Sprite* [79], *Panasas ActiveScale Storage* [46], *NFS* [51], *StorageTank* [80], *AFS* [52], *Coda* [53], *Hadoop Federated HDFS* [54]
	Hash Function Mapping	*Vesta* [55], *Intermezzo* [81], *RAMA* [82], *zFS* [56], *DROP* [57], *Lustre* [6]
	Subtree Hashing	*Lazy Hybrid* [48], *GIGA+* [61], *DROP* [57], *AngleCut* [31], [62]
Dynamic Space Division	Dynamic Subtree	*Ceph* [9], *Farsite* [83], *Envoy* [84], *BWMMS* [68], *pNFS* [69], *Mantle* [70]
	Directory Item Subset	*IndexFS* [49], [71] [17], [74]
	Dynamic Hashing	*GPFS* [75], *G-HBA* [76], *AbFS2* [77], *AngleCut* [31], [62], *NICE* [78]

TABLE 3
Comparison Between Static Namespace Method and Dynamic Namespace Method

Metric	Static			Dynamic		
	Subtree Partitioning	Hash Mapping	Subtree Hashing	Subtree	Directory Item Subset	Hashing
HotSpot	Yes	No	No	Yes	Yes	Yes
LoadBalance	Bad	Good	Good	Good	Good	Good
Locality	Good	Bad	Good	Good	Good	Bad
Scalability	High-Cost	High-Cost	High-Cost	Low-Cost	High-Cost	High-Cost
Name Change	High-Cost	High-Cost	High-Cost	High-Cost	High-Cost	High-Cost

can reallocate metadata on the LPH-based hash ring accordingly. In particular, Kettaneh *et al.* [78] proposed a storage system, *NICE*, which leverages a software-defined network to optimize the scalability of the DFS. The MDS in *NICE* is responsible for managing the metadata for the storage system, which includes the information of the storage nodes in the *NICE* and the range of the hash space (partition) served by each storage node. With the heartbeats of storage nodes, the MDS can detect the membership changes (say, joins and failures) and control the *OpenFlow* switches to update the forwarding tables.

2.3 Summary

In this section, we overviewed a variety of scalable technologies for MDSs in DFSs. The studied works are summarized in Table 2. And we compared these methods from five aspects: access hotspot, load balancing, support for access locality, resource elasticity, and namespace change.

According to the comparison in Table 3, each of these methods has its own merits. For instance, static hashing methods (i.e., hash function mapping and subtree hashing) can effectively avoid the hotspot issue. While as mentioned in 2.1.1, the static subtree partitioning method performs poorly in load balancing compared with other methods. As for the locality, the naive hashing methods cannot maintain good local correlation of data, so the locality performance of these methods (i.e., hash-function based mapping and dynamic hashing) are worse than that of others. In contrast, for the cost of scalability, the dynamic subtree can effectively reduce the adjustment cost when adding or removing nodes due to the dynamic tree structure. From this table, we can see that the current technology is still expensive in terms of adapting to resource flexibility and filename change.

In summary, all these methods attempt to make a trade-off between locality and load balancing. In particular, subtree-based methods focus more on locality, while hash-based techniques often concentrate on load balancing. Some others try to mix up these methods by using different strategies, but they have to suffer from degraded performance in respective aspects. As such, there is no universal solution effective for both metrics simultaneously, and how to combine those methods to strike a balance between the locality and load balancing in favor of the scalability with a reasonable cost is still an open problem worth further studying.

3 High-Performance Technology for Metadata Services

We broadly classify high-performance technologies for MDSs into three categories. The first category is those used to address the scalability of MDSs, such as load balancing, hotspot elimination, etc., to improve the performance of the entire file system. Second, with respect to the attribute set of the existing metadata, a new indexing mechanism is established for various query applications to improve their efficiency. The third category of technologies goes one step further, and we call it the *value-added technology* for MDSs. The core idea is to extend the metadata set for a variety of applications, each with different purposes. The extended metadata set is called the *value-added metadata set*, and we improve specific application performance by leveraging the application-oriented value-added MDSs.

3.1 Cache and Replication

Cache and replication are two key technologies often-used to achieve the high scalability of MDSs. Meanwhile, since they can greatly reduce response time by caching hotspot metadata in memory or organizing metadata in a more accessible manner, regardless of whether the cluster is scaled out or up, both caching and replication can dramatically reduce the latency of MDS to retrieve metadata, thereby improving the overall performance of DFS. Moreover, caching can be implemented at both server-side and client-side in attempting to address performance issues in addition to scalability.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

Therefore, both cache and replication also play key roles in improving the performance of the entire file system. For example, in *Ceph*, in order to eliminate access hotspots for load balancing, both the authority node and collaborative caching are adopted in the research, in which the authority node is defined for each metadata item to implement the serial access to the metadata at certain times, record the changes (say, write to external memory), and maintain the cache consistency and coherence when there exist multiple copies of the same data. Cooperative caching requires that when a metadata item has a copy in another MDS's cache, the authority node has the right to communicate with it to maintain the cache coherence. If a cached copy of a non-authority MDS is removed, the authority node is also notified to remove its local copy for cache consistency. The concept of authority nodes decentralizes the metadata control and it is the basis for implementing the replica technology. The cooperative cache maintains the consistency of the copy [85].

In distributed metadata management, the path name lookup is often considered a major performance factor, limiting the system scalability and affecting the overall system performance. Xiao *et al.* [86] conducted an in-depth quantitative study on this problem by comparing *ShardFS* [87] with *IndexFS* mentioned above. In order to improve the access performance of the metadata, *IndexFS* maintains a consistent lookup cache on the client for each component of the path name and access rights. By using a lease for each directory entry, it reduces the amount of invalidation caused by changes to path names, improving the performance of the metadata query service. In contrast, *ShardFS* leverages the query state of the replication path in the MDS cluster to ensure that each file operation acts on only one node, eliminating the need for lock-based accesses to multiple servers. In addition, by classifying the metadata operations, the implementation of specialized distributed transactions further reduces the replication cost. These two methods have their own problems in spite of their respective advantages. In *IndexFS*, when some accessed directory entries are not in the cache, the cache miss would occur, and thus, the client has to experience a high query delay while in *ShardFS*, the high latency of the client is mainly from the overhead to maintain the multi-copy consistency caused by the updates to the directory metadata.

Also, the update mechanism of the cache is an important issue worth studying. Due to the different read/write speeds between the CPU cache and the computer memory, two cache update methods are often adopted in MDS: *write-through* and *write-back*. In write-through cache, data is written to the main memory simultaneously as the cache is updated. In contrast, in write-back cache, data is only written to the main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, the writes by the processor only update the cache. Compared with its write-through counterpart, the write-back cache usually achieves better performance for the update operation of the MDS. Therefore, it is widely used in various DFSs, say *Lustre*, which overcomes the link latency for modifying the metadata by using the write-back cache [88].

Caching mechanisms can be implemented at either server-side or client-side. Traditionally, it is implemented on the server-side. Client-side caching is also well worth studying as it can dramatically reduce the server-side workload even though some consistency cost is incurred. For example, the *hierarchical Persistent Client Caching* (*LPCC*) proposed by Qian *et al.* [34] is such a client-side cache mechanism designed specifically for the performance improvement of HPC file systems (e.g., *GPFS*, *Lustre* and *BeeGFS*). Furthermore, Cheng *et al.* [89] proposed a non-volatile main memory (*NVMM*) oriented client-side caching for *Lustre* based on *LPCC*. This approach, named *NVMM-LPCC*, leverages *NVMM* to attain high-speed cache retrievals, as well as classifies the cache into read/write and read-only modes to take full advantage of locality. Also, Xiao *et al.* [17] found that if the operations that change the directory query state occupy a fixed share in all the operations of the metadata, the server-side replication model of *ShardFS* is not as effective as the client-side caching in *IndexFS* in terms of scalability and performance. However, if the operations that change the status of the directory query are proportional to the number of jobs, the server-side replication model has better linear scalability than the client-side caching method. In addition to these two methods, Pineda-Morales *et al.* [90] proposed a specific technique that leverages workflow semantics and combine distribution and replication for in-memory metadata partitioning. Nevertheless, for a large number of small files, the cost of requesting metadata usually exceeds the cost of actually requesting data, so that the caching and replication of metadata may not work well for performance in this case. To address this issue, Matri *et al.* [91] proposed a novel DFS architecture, optimized for small files based on *consistent hashing* [92] and dynamic metadata replication. This design allows clients to locate the data without resorting to the metadata, while the dynamic replication replicates the metadata among the MDSs to adapt to the workload changes.

Meanwhile, based on the observation that there are usually more idempotent and fewer dependencies in the real-world data, Li *et al.* [93] and Bravo *et al.* [94] proposed *Replichard* and *Saturn*, respectively. Both of them implement *causal consistency* [95] for MDS, and classify the request operations into non-idempotent and idempotent classes. For the idempotent requests, they can be serviced by any MDS, which holds the replication while for the non-idempotent requests to the same path, all of them will be assigned to the same MDS to ensure its consistency in a simple way. This is a reasonable and flexible consistency scheme, but the handling of failures and the atomicity of operations should be taken into consideration thoughtfully [96].

Given the replication, the metadata would eventually have an unbalanced distribution. How to combine these methods to achieve the optimal performance based on the changes of workload characteristics is a problem worthy of further study. As mentioned above, some techniques [31], [57], [62], [97] (e.g., *DROP* and *AngleCut*) could rebalance the distribution through the metadata access frequency. Similarly, Cao *et al.* [98] proposed *AdaM*—an adaptive fine-grained metadata management scheme. Unlike those history-based methods, *AdaM* leverages *deep reinforcement learning* to address the load balance against the time-varying access patterns. By evaluating the system performance at each period, *AdaM* leverages an algorithm *DDPG* [99]—a

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

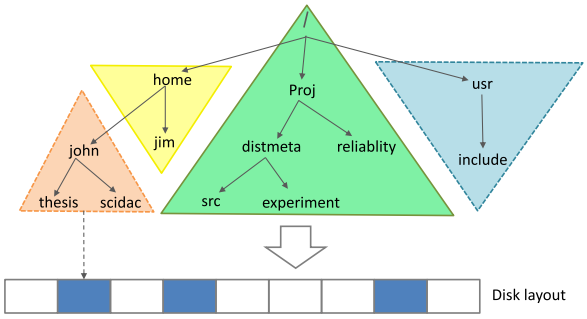

Fig. 7. *Spyglass* index tree, a namespace decomposition based sub-tree.

deterministic policy-based algorithm—to perform rebalance actions to achieve the load balance and the locality preservation timely. The experiments show that *AdaM* can achieve higher performance than the *AngleCut* approach with a lower migration cost.

3.2 Metadata Retrieval

In many cases, the operations on metadata are not just locating files through directory lookup and access control, instead they are versatile, not only including those for points, ranges, and top-k, but also having aggregated searches like the file attribute sets to fulfill specific lookup requirements. How many copies of a file are in the system? Which files consume the most space? For these problems, the aforementioned techniques have to employ a brute force search, and these requirements cannot be effectively fulfilled due to the lack of indexes on the metadata attributes. The efficient metadata indexing is usually designed based on the understanding of the characteristics of the metadata accesses as well as the metadata itself. For example, Leung *et al.* [15] proposed *Spyglass*, which exploits two main techniques to implement the aforementioned metadata and leverages its access characteristics to achieve fast and scalable metadata retrieval services: *hierarchical partitioning* and *partition versioning*.

3.2.1 Index Tree

First, the directory space of the file system is partitioned into subtrees using hierarchical partitioning, and one or more directory subtrees responsible for indexing each partition are given in the form of the *K-D tree* [100]. The partitioned index stores the metadata of the files in the partition, while the partition's own metadata retains the partition information and pointers to the child partitions. The partition includes the information used to determine whether the partition has the information related to the search condition and support for partition version control. The former is implemented by comparing the file signature representing the contents of the file. It can be seen that the indexed partition constitutes a tree, called the *Spyglass index tree*, and each index partition is sequentially stored on the disk as shown in Fig. 7. This kind of partitioning and its storage method retains the support of spatial locality for file system accesses, but at the same time, there is a problem of how to adapt to the changes in file metadata in the partition, such as addition, deletion, and change. To this end, *Spyglass* uses a version control approach—partition versioning, instead of modifying the metadata information in the field. Specifically, Spyglass batches the modifications of the file metadata in the partition at regular intervals (file system parameters). The modified partition is stored as a new indexed version and in an incremental version of the older version. Simultaneously, the old version is also retained as a support for the user's "back-in-time" retrieval and trend query.

Compared with traditional methods that establish the file metadata index based on *DBMS*, *Spyglass*'s method not only increases the retrieval speed by eight to 44 times, but also reduces the storage space by five to eight times [101]. However, this indexing also imposes certain burdens and overheads on the memory capacity and the organization of metadata. How to integrate it with the aforementioned load balancing is a complicated, yet worth exploring, problem. Replacing the k-dimensional tree (*KD-tree*) with a more efficient index tree (such as prefix trees) might be a viable way to reduce memory footprint and improve performance. For instance, Masker *et al.* [102] proposed *Hyperion*, an efficient in-memory indexing data structure, which can significantly reduce the index memory footprint and improve the performance. Base on a similar idea, *SmartCuckoo* [103] improve the *Cuckoo Hashing* method by using a directed pseudo-forest to express hash relationships, thus avoiding the occurrence of an infinite loop on the metadata index and enhancing the performance of the original *Cuckoo Hashing*.

3.2.2 Bloom Filter

For the retrieval of metadata, Hua *et al.* [76] also proposed a similar method based on the hierarchical division between groups, which is called *G-HBA*. Unlike the subtree partitioning for directory space in Spyglass, *G-HBA* organizes the MDSs into logical groups. Each MDS in the group maintains a set of *Bloom filters* [104], [105] as a *filter bank*, representing all the files whose metadata is stored on this server. This server is also known as the *home server* (home MDS) for these files whose maintained filter bank periodically passes the replication information to all the other servers. In this way, each server can accommodate a mapping of a portion of the files in the system to its primary server and each group as such to roughly maintain a mapping of all the files to their primary servers. Since the metadata query of *G-HBA* is different from the traditional path name hashing method, the query can start from the detection of the Bloom filters from any MDS, and is performed step by step from the local intra-group to the final inter-group. The organization between groups is different from the hierarchical way, in which the partitions are decomposed along the directory tree in *Spyglass*. Moreover, it also has no fixed structure. Through this method it is easy to achieve load balancing of storage space, but intra-group and inter-group queries have to use multicast methods, which increases the network traffic. Additionally, as mentioned earlier, hot issues caused by the skewness of accesses are also a factor worth considering.

Similarly, both *SoMeta* [106] and *MDBF* [107] also use a Bloom filter to accelerate the metadata search process for large-scale DFSs. Specifically, *MDBF* is a parallel metadata search method based on a Bloom filter, which unlike the grouping strategy of *G-HBA*, is based on the directory tree. *Multi-dimensional Bloom filters* (*MDBF*) are implemented as network services, which support multiple metadata attributes

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

representation, and each *MDBF* is related to a directory, respectively. As opposed to the directory tree approach used by *MDBF*, a flat namespace is used by *SoMeta* to manage metadata, which could avoid the overhead of traversing the prolonged directory path in the hierarchical namespace. Although both the search performance of *MDBF* and *SoMeta* are improved slightly, the space overhead also increased correspondingly.

Parallel computing is an alternative effective way to improve metadata retrieval, and the client can operate on the metadata in parallel by using multiple threads. For instance, in *Lustre*, a new feature is developed that allows the metadata to be manipulated by remote procedure call (*RPC*) that operates in parallel [108], which in turn improves the multi-threaded metadata performance of a single client.

3.2.3 Pre-Fetching

In addition to the techniques discussed above, the provenance of metadata is another consideration that can be used to speed up the metadata retrieval. The provenance is a type of metadata that is derived by analyzing the behavior of a user's process relative to a file. Such metadata describes which process creates the item, modifies it, and why the item is found in the current location. Therefore, the provenance analysis of the data can reflect the correlation between the file and the process, and accelerate metadata retrieval by partitioning and indexing metadata with these relationships, and pre-fetching the metadata that has not been accessed in advance according to the correlation. *PROMES* [109], *P-Index* [110] and *ProMP* [111] are all explorations that try to achieve high query accuracy and low latency through provenance analysis. *PROMES* indexes data in a relationship graph coming from provenance's analysis. It leverages correlation-aware metadata to build the index tree, and would execute the query in index subtrees when a query request was received by a storage system. While *P-index* partitions metadata into logical groups based on correlation via provenance relationships and cuts off the subtrees that do not contain the query results to reduce search scale. Moreover, it builds the index tree structure with the metadata coming from provenance, which is similar to *PROMES*. And *ProMP* uses provenance to support metadata pre-fetching. The *ProMP* mines the correlations between processes and the corresponding files then generates a Provenance Window (PW) to compute the file access correlations in the same PW, thus achieving the aggressive pre-fetching. The experimental results demonstrate that significant performance improvements in terms of metadata search occur by using provenance in indexing and pre-fetching.

The aforementioned provenance-based pre-fetching method tends to obtain file associations from the access history. Nevertheless, Chen *et al*. [112] found that the correlation of file mining from historical co-occurrence frequency is not sufficient for all the files. To supplement this correlation, they proposed a novel approach, called *SMeta*, to explore the explicit data correlations by mining the reference of hyperlinks that exist in many applications. Based on the analyzed correlations, the *SMeta* leverages a correlation-directed algorithm for pre-fetching. The experiment implements the *SMeta* atop of *Ceph* and evaluates it using synthetic and real system workloads. The results show that the metadata performance has a significant improvement.

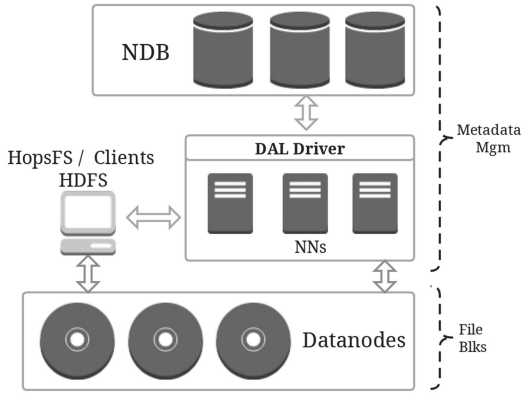

Fig. 8. The architecture of *HopsFS*. An external database named "NDB" is adopted to manage metadata in this implementation.

3.2.4 Key-Value Database

Xu *et al*. [113] gave a more detailed discussion on the metadata retrieval in Big Data high performance computing environments, from the aspects of *index manageability*, *scalability*, *performance* and *POSIX interface support*. The existing metadata index results including the above techniques are summarized, and the classification of add-on and database models is proposed, and at the same time they are compared. On the basis of combining the advantages of the file system and database, a file system level metadata management system *VSFS* [113] is proposed. *VSFS* supports transparent namespace queries and flexible file indexing. However, the proposed single master structure of the system does not have large-scale scalability.

For the background storage optimization, *IndexFS* utilizes the *LSM-Tree* [27] technology commonly used in current Key-Value Store systems to represent and store the metadata and small file data. Each server stores and manages part of the file system metadata and uses *LevelDB* to package metadata and small file data into a flat large file. Coincidentally, in the architecture of *BlueStore*—a novel storage backend for *Ceph*, a key-value store, called *RocksDB* [114], [115], instead of the file system, is used to store the metadata [116], [117]. Based on *Ceph*'s experience, storing metadata in *RocksDB* allows it to leverage fast metadata operations. Given its performance improvements and some other advantages, *BlueStore* has become the default storage backend to *Ceph* within just two years.

In addition to fast metadata operations, storing metadata in a database also provides a simple option for the MDS cluster. For instance, an *Active NameNode* (*ANN*) is needed to manage the metadata in *HDFS*, and at least one *Standby NameNode* is needed for high availability. Consequently, the unique *NameNode* is now the performance bottleneck in *HDFS*. To address this issue, Niazi *et al*. [26] proposed a next generation distribution of *HDFS*, called *HopsFS*, whose architecture is shown in Fig. 8. *HopsFS* supports multiple stateless *NameNodes* and stores the metadata in an external *NewSQL* database [118].

In comparison, for both *IndexFS* and *ShardFS*, the MDSs handle the metadata stored in local *LevelDB* instances and a

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

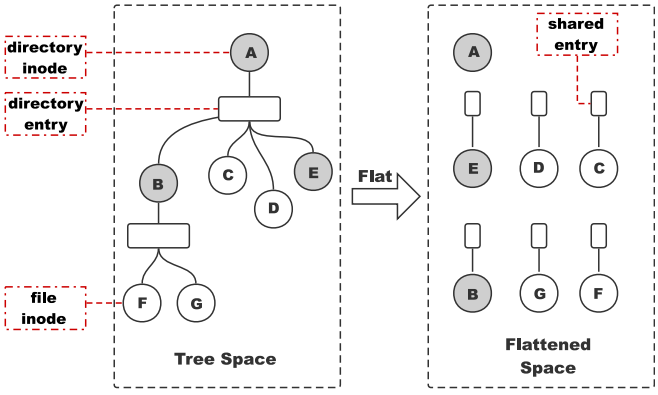

Fig. 9. An example of the flattened directory tree shows how to transition from tree space (left) to flattened space (right).

caching mechanism is exploited to improve the performance. Instead, *HopsFS* only makes load balances at the top-level directories. The experimental results show that *HopsFS* could deliver 37 times the throughput of *HDFS* for write-intensive workloads.

Nevertheless, although a few DFSs try to optimise the performance by using the key-value store, the results are not that satisfactory in empirical studies. Some researchers find that the performance gap between the key-value store and the DFS metadata is still remarkable. From these pieces of research, it shows that although some file systems gain benefits by storing metadata in a key-value store, the dependencies between directory trees still prevent the file systems from making full use of the advantages of the key-value store [119]. To improve the performance of MDS, Li *et al.* [120] studied this problem in depth and proposed a solution called *LocoMeta*—a flattened MDS for DFSs.

As shown in Fig. 9, *LocoMeta* proposes strategies to transform directory tree metadata into a flat space, which is called a *flattened directory tree*. With this design, the metadata objects are independently stored in a flat space. Evaluations show that *LocoMeta* achieves lower latency and more efficient input/output operations per second (*IOPS*), compared to *IndexFS*. However, from other research on the flat namespace, there are two challenges in the flat namespace approach, one is due to the lack of logical organization (e.g., directory tree), and the other is locating specific or related metadata [106], [121].

3.3 Value-Added Metadata

The so-called metadata value-added technology is an application-oriented technology that is built on top of existing metadata management in DFSs to integrate some specific metadata for particular applications. As such, it can combine with the aforementioned index technology to specifically improve the computational power and performance of certain types of applications. The metadata value-added technologies can cover a wide range of applications, each having a different purpose. This section focuses only on several techniques that have an impact on high-performance applications.

In order to exploit the values of some intermediate results that would, otherwise, be discarded in scientific computing, Wang *et al.* [122] added the description of the structure and semantics of the intermediate data files as the value-added metadata into the metadata management in existing DFSs. This approach can reduce redundant computational tasks by tracking and mining the originally discarded files. As a consequence, the performance of the complex scientific computation is correspondingly improved.

Based on the above research, Wang *et al.* [123], [124] further designed and implemented a scientific workload-aware file system, called *WaFS*, which could store the read and write dependency information between files in an in-memory database. This part of the added-value metadata overcomes the inabilities of the existing file system to record the dependencies between files. The workflow scheduler is an effective tool to improve the concurrency of workflow tasks and maximize resource utilization with such dependencies. In spite of this advantage, *WaFS* is still limited to the support for workflow computations, and only works for workflows that are characterized by a control-flow dependency.

Other works in this area include the distributed storage middle-layer technology proposed by Zhao *et al.* [125], [126] for metadata-intensive operations in *FusionFS* and the "rich metadata" concept proposed by Dai *et al.* [127]. It not only records some attributes of related file entities in high performance computing, but also integrates their relationships such as the generational relationships, through the proposed generic property graph, enriching various queries for metadata in high performance computing. Conceptually, "rich metadata" and the dataflow dependencies proposed in *WaFS* could complement each other. The generational relationship and data dependence characterize the logical relationships between the files in the computing process, which are not possible for traditional file systems to achieve.

In addition to the POSIX metadata, the value-added metadata could introduce extra associations between processes, tasks, file, etc. These associations are more like a graph representation than like a separate flag description. Therefore, Dai *et al.* [128] designed *GraphMeta*, a graph-based rich metadata management system for HPC platforms. Although the graph approach can facilitate the representation of metadata associations, the complexity of graph partition and graph search still makes it barely adopted in practice.

As an integrated solution, Sim *et al.* [129] combined the above methods and proposed a scalable, distributed metadata indexing framework, called *TagIt*, which facilitates a flexible tagging capability to support data discovery. *TagIt* provides a PB-level tagging function, allowing users to add their own tags to the metadata, which as a result associates rich contextual information for rapid retrieval. Moreover, *TagIt* also supports executing operation or filter on the tagged files, which can be seamlessly offloaded to storage servers in a load-aware fashion. To validate the feasibility and performance, *TagIt* has to be implemented into both *GlusterFS* [10] and *CephFS* [9]. The empirical results demonstrate that such a value-add method can efficiently expedite the data search operation.

3.4 Summary

This section introduced several high-performance techniques: cache & replication, metadata retrieval, and value-added technology. Table 4 summarizes the comparison of these technologies, where both value-added and replica technologies

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

TABLE 4
Comparison Among Cache&Replication, Metadata Retrieval and Value-Added Technology

Metric	Cache&Replication	Metadata Retrieval	Value-Added
Extra Memory	Yes	No	Yes
Universality	Good	Good	Bad
Usability	Good	Bad	Good
Pre-Optimization	No	No	Yes

are implemented by using extra memory, and hence, they are simpler and more efficient to achieve than the metadata retrieval. However, the value-added method is usually weak in generality as it requires field-specific optimization to carry out a particular design, according to the application scenarios. It is for this reason that it is often used in pre-optimizing the metadata without the requirements for real-time processing like the other two. Rather, without considering the cost of the extra space, the cache&replication is undoubtedly a worthwhile way to improve the performance without focusing on the application itself.

For high-performance optimization, both cache and replication mechanisms have been developed in relatively mature forms and found in almost all the existing DFSs [34], [89]. Additionally, given the merits of the key-value database, its introduction into metadata management is also a promising method, which has become a trend in recent mainstream DFSs [114], [117], [130]. Furthermore, as an optimization method for data administrators, a variety of DFSs also provide extended interfaces for value-added approaches, say, the secondary and tertiary indexes [128], [129]. All these methods are orthogonal and thus can be integrated to perform various purposes. We then expect the future DFS to support all these optimization methods in one shot as a natural evolution of the DFS development.

4 High Availability Technology for Metadata Services

Due to the ever-increasing scale, the MDS always faces the risk of potential faults. To ensure its uninterrupted services, the MDS cluster is required to be highly available. As mentioned earlier, in shared cluster file systems, for each partitioned directory, several candidate MDSs are randomly selected, and the directory subset will be stored in those with fewer directory entries to improve the availability of the MDSs.

4.1 Copy-Based High Availability Metadata Service

Distributed metadata management is performed through multiple MDSs to overcome the single point of centralized metadata management to improve the availability of metadata. Although a few MDS failures could not crash the entire system, it could affect to some extent the normal access of the system. To achieve higher availability of MDS clusters, we can adopt redundancy technologies for metadata management where the metadata on each MDS is backed up to other MDSs, according to the workload distribution in the cluster. Both *Hadoop* and *GFS* use copy-based metadata management. Considering the metadata recovery time, the metadata backup can be divided into a restart recovery mode and a warm standby mode.

The process to recover the metadata is mainly to backup the metadata from one MDS to other MDSs and set checkpoints periodically. When the MDS fails, the backup MDS can be enabled to load the metadata starting from the latest checkpoint, so the MDS can be restored at that point. The advantage of this mode is that development is simple and convenient, but the biggest hidden danger is the inconsistency of the metadata. Moreover, the metadata is updated after the checkpoint is not restored. To solve these problems, the *log* method is usually combined to make the recovered metadata of the standby MDS the latest version. However, the restart and recovery still require a certain amount of switching time, rendering real-time recovery impossible.

As with the method used to recover the metadata, the hot backup method is also required to select a standby MDS to become the active MDS. Both the MDSs can access the directories in the shared storage devices, and the directories store the logs for updating the namespace. The alternate MDS will detect the logs, and update all log records to its own server. The data storage server's data location information and heartbeat detection are sent to the two MDSs so that the corresponding information can be updated in real time. Once there is an MDS failure, there will be a backup MDS to replace it. *Hadoop*'s metadata management is based on this *high availability* (HA) principle.

Although *HA* is widely being used in *HDFS*, there are still some obvious shortcomings. For example, the checkpoint mechanism leads to a constraint on the size of metadata states, which is required to fit within the memory of the MDSs. To solve this problem, Stamatakis *et al.* [131] presented a novel architecture for replicated MDSs, which is called *HDFS-RMS*.

Fig. 10 illustrates the difference between *HDFS-HA* and *HDFS-RMS*. There are only two *NameNodes*, one active *NameNode* and one standby *NameNode* in *HDFS-HA* while the *HDFS-RMS* architecture consists of one master and any number of follower *NameNodes*. Whenever the master crashes, the service can be resumed by switching to a new *NameNode*, which accesses the state from the Oracle Berkeley DB (*BDB*) replicas.

In *GFS*, the standby MDS can provide read-only access to some files in certain cases. If a high-available module is appointed on the MDS, the MDS is provided to the system while the fault information is shielded, so that when the MDS fails, the metadata request is automatically switched to another candidate MDS, thereby achieving high transparency and high availability to the clients. This method can ensure better data consistency and short switching time, but it adds a certain complexity and brings a lot of pressure to the system maintenance.

The high-available solution in *BeeGFS* [8] is similar to *GFS*. The metadata mirroring with high availability in *BeeGFS* is

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

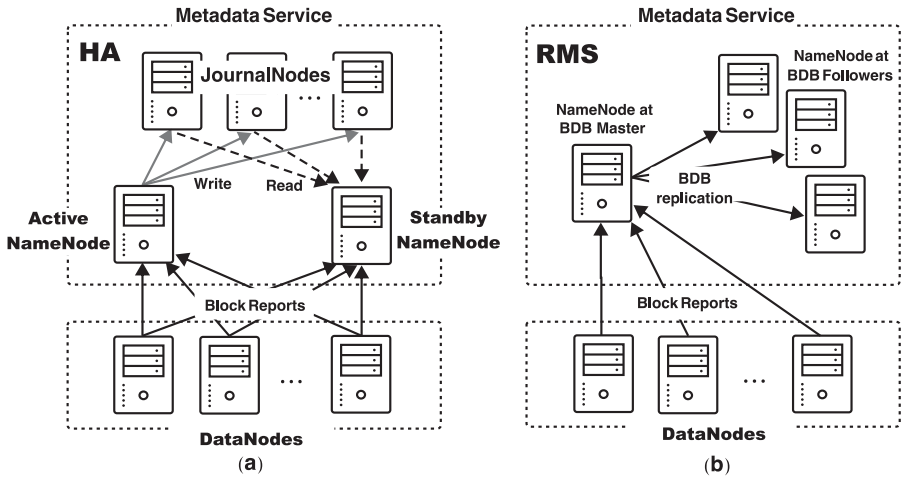

Fig. 10. Architecture comparison of *HDFS-HA* (a) and *HDFS-RMS* (b). It can be seen that the *RMS* architecture is more reliable.

called buddy groups. In a buddy group, the metadata is replicated between a pair that targets each other, which means the buddy group approach is still accessible while one-half of all servers have failed. The design of the buddy group in *BeeGFS* is illustrated in Fig. 11. As shown in this graph, each group is a replication between two targets. Normally, one of the MDSs in a buddy group is regarded as the primary, while the other is the secondary. Modifying operations will always be sent to the primary first, and the mirroring process is running in the primary. Once the primary MDS of a buddy group is unreachable, it will be marked as offline and the former secondary will become the new primary. Such a failover is transparent and happens without any loss of data for running applications.

Both *Farsite* [83] and *Archipelago* [132] leverage the replica-based method to manage the metadata. However, in order to reduce the overhead caused by the replication, *Archipelago* divides the nodes in the clusters into several small islands, and copies the metadata to each island for uninterrupted MDSs. In contrast, *Farsite* copies the metadata to several nodes. In order to reduce the resulting overhead, modifications on the metadata are recorded in logs and periodically copied to other MDSs. Chen *et al.* [133] used a similar technology to design a highly available MDS for the *Dawning Cluster file system* (*DCFS3*). The basic idea is to reduce the latency from the Packed Multi-Paxos, and at the same time, minimize the overhead of the consistency protocol incurred by the number of copies. Zhou *et al.* [134], [135] proposed a new highly reliable policy of MDS called *Multiple Actives Multiple Standbys (MAMS)*. Similar to *Archipelago*, *MAMS* divides MDSs into different replica groups and maintains more than one standby node for failover in each group. In contrast to traditional strategies, there are three states of MDS in *MAMS:* active, standby or junior. The active MDS is responsible for online management, while the standby MDS keeps up-to-date with the active node, and the junior MDS is a passive standby server. The experimental results show that the *MAMS* policy can achieve a faster transparent fault tolerance with less influence on metadata operations than the traditional HA principle.

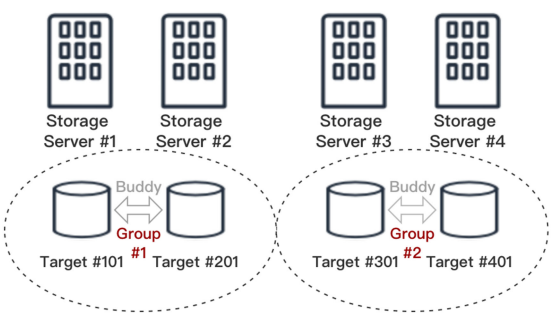

Fig. 11. The design of the buddy group in *BeeGFS*.

4.2 Log-Based High Availability Metadata Service

One of the inevitable problems of the replica-based MDSs is latency. In order to reduce the number of disk I/Os for metadata accesses, caching technology is often used to save partial metadata operations. Once the MDS is shut down, the cached metadata items are lost. Therefore, to improve the availability of the MDSs, the file system needs to be able to persist all the metadata modifications. As such, it often requires a log method to record the metadata modifications for each MDS. Even if the data in the cache is lost, it is still possible to recover the data based on the logs.

In the shared-disk cluster, the metadata and logs are stored in the shared disk array-based storage (such as storage area network (*SAN*)). In order to maintain the consistency of metadata, *GPFS* exploits the distributed locks to achieve the multi-user synchronous accesses to the metadata, and the distributed locks have better concurrency than the centralized management [75]. The update operations on the *GPFS* metadata implement the consistent updates of the file system through a write-ahead logging (*WAL*) mechanism. Although the shared-disk clusters are relatively simple to maintain the metadata consistency, they also face the "hot files" problem and cannot cope with the unstable and high-latency network traffic in wide-area network (WAN) environments across different data centres [136].

In a share-nothing cluster, the metadata is distributed across multiple independent MDSs in multiple copies. The modification operations easily involve the metadata on multiple MDSs, and the consistency of the metadata becomes more complex. To design a metadata cluster that can perform well across WAN environments, the metadata modification operations, according to the idea of *CalvinFS* [136],

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

TABLE 5
Comparison Between Copy-Based and Log-Based High Availability Method

	Copy-Based	Log-Based
Usability	Good	Bad
Consistency	Bad	Good
Switching Time	Short	Long
Latency	Low	High
Network Overhead	High	Low

are added with distributed transaction logic and a scheduler for locking management to synchronize the metadata logs via the *Paxos* protocol [137]. The normal Paxos protocol usually requires two stages when synchronizing each piece of metadata. In contrast, some research uses a Multi-Paxos protocol [138] to synchronize the logs in the study of the replica-based metadata management, reducing the number of first-stages for network overhead reduction. Although Google's *Chubby* [139] uses this metadata management based on the Paxos protocol to the synchronization log, it is well known that the Paxos protocol is complex and difficult to implement. In contrast, the Raft consistency protocol [140] is easier to understand and simpler to implement. In *Kudu* — a large open source storage engine developed by Cloudera, tablet replica management (*TRM*) is implemented through the Raft protocol [141].

Although the separation of metadata and data makes the read and write logic clearer, it increases the number of times the disks are accessed. Since each metadata occupies a small amount of storage space, the data block could be compressed, leaving a small amount of space to embed the metadata, so that the data and the metadata could be written in one shot to reduce the disk I/O overhead. *Selfie* [142] exploits this principle where the metadata of the file system can still be quickly reconstructed by scanning the data block and provide services for the system even if all the MDS clusters crashed. Additionally, this approach can also reduce network overhead and additional synchronization operations than copying the metadata to other MDSs.

4.3 Summary

We introduced two of the most popular high-availability techniques in metadata management: copy-based techniques and log-based techniques in this section. The comparisons between them are shown in Table 5 where five metrics are compared: *usability*, *consistency*, *switching time*, *latency*, and *network overhead*. As shown in the table, the copy-based method performs better in terms of usability, switching time, and latency, but worse in consistency and network overhead than the log-based methods. We can attribute these results to the fact that the copy-based methods leverage the network transmission between the primary node and the standby nodes for real-time data synchronization, thereby reducing the synchronization overheads between these nodes. Meanwhile, the standby node can be switched to the primary node immediately when the primary node crashes. However, this synchronization mechanism is a double-edged sword as it may also lead to data inconsistency when the service is crashed or network packets are lost. In contrast, the data inconsistency issue can be successfully avoided with the log-based methods, which could exploit the supplemental logs to persist all the metadata modifications, by which the MDS could be re-constructed whenever the service is crashed.

Many DFSs [34], [75], [131] provide a flexible alternative MDS through copy-based methods, which can apace switch to the secondary MDS in case of the primary failure. Meanwhile, they log the metadata operations into files to recover from these logs in the event of a failure, thus providing a more stable and reliable MDS service. Therefore, in reality, these two approaches—copy-based and log-based techniques are often combined together to achieve high system reliability. Moreover, proactive detection, localization, and failure diagnosis are also interesting and valuable research directions for boosting availability. Jha *et al.* [143] proposed a failure detection and diagnosis framework, named *Kaleidoscope*, which has been deployed on PB-Scale DFS to detect the failures from resource overload/contention issues for the improvements of the availability of DFS. By introducing the proactive detection mechanism, DFS can detect failures before crashes, thereby further improving the availability, so that detecting failures in advance is a problem worthy of research in MDS.

In today's era of rapid hardware performance and network bandwidth development, the extra overheads from the replication-based and log-based approaches could be negligible. Instead, we are more concerned with how to detect failures apace and recover in time. Therefore, proactive detection, failure location, and fast switchover are expected as the keywords in DFS availability.

5 Future Trends

The applications of metadata are multi-perspective and omni-directional [90], [119], [128]. In order to improve its performance in various applications, research on efficient and scalable technologies for the management of metadata still demands compelling needs. This trend has been becoming prominent in recent years, which is expected to manifest itself in the three following emerging areas [31], [34], [62], [102], [103], [144].

5.1 Ai-Based Mds

The current mainstream methods on high scalability and high performance usually focus on specific kinds of workloads. As AI workloads are dominating most applications nowadays, we would expect the design of *DFS* in general and its *MDS* in particular to effectively support AI applications as a trend. For example, Bae *et al.* [145] presented *FlashNeuron* which can leverage *NVMe SSD* to optimize the training process of deep learning. The empirical results show that the proposed storage optimization can significantly improve the throughput of deep learning algorithms. In addition, Kumar *et al.* [146] proposed *Quiver*, which is a caching system optimized specifically for the deep learning framework *Pytorch*. Due to the nature of iterative computation, the distributed learning frameworks have some specific properties. Some insights on how to design a customized high-performance DFS for them can be found in [147].

Reversely, as an efficient optimization method, we would also envision the applications of AI algorithms in the high-

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

performance and high-scalability technologies of MDS [98], [148]. For instance, Gao *et al.* [148] presented a machine learning-based model, called *DeepHash*, to take advantage of deep learning algorithms to design a metadata locality mapping. Moreover, from a system perspective, reinforcement learning [149] is very suitable for dynamic optimization scenarios such as load balancing. Therefore, Wu *et al.* [150] proposed *MDLB*—a RL-based mechanism for dynamic metadata load balancing—to achieve efficiency and dynamic adaptability in the load balancing of MDS.

5.2 New Medium-Based Storage

The revolution of new materials will also bring a leap in the performance and scalability of MDS. The emergence of new storage (e.g., *SSD* and *NVMe*), as well as the remote direct memory access (*RDMA*), dramatically improves the throughput of metadata server clusters. And the high-speed data I/O capabilities would undoubtedly bring a strong driving force to the performance improvement of MDS. Therefore, we believe these revolutions will become the crucial technology for highly scalable MDS clusters in the future.

Compared to hard disk drive (*HDD*), solid state drive (*SSD*) is characterized by its high storage density, low energy consumption, good read and write performance, and potential being directly used in user space without going through the kernel, rendering it greatly useful to improve the performance of the file storage system. Given these advantages, numerous recent research efforts have focused squarely on adopting *SSD* as the DFS substrates [38], [43], [45], [151], [152].

CosaFS [43] is a typical file system built on top of heterogeneous storage devices, which combines *SSD* and shingled magnetic recording (*SMR*) [153] technologies to improve the I/O performance. At present, the mainstream SSD storage is the hash-based Key-Value SSD. Although its performance is promising, maintaining a hash table in its controller DRAM usually results in inconsistent tail latency and throughput. To fully unleash the potentials of SSD, Im *et al.* [130], [154] presented *PinK*—a LSM-Tree-based KV-SSD—to avoid the impact of hash conflicts on performance. Notably, Lee *et al.* [155] proposed a new technology, called *SmartSSD*—a *SSD* with an onboard FPGA to provide computational capacities inside *SSD*. *SmartSSD* is expected to be a potential revolution to DFS.

Emerging as cutting-edge technology with higher read and write speed than *SSD*, non-volatile memory (*NVM*) contributes a significant promotion to file system performance and it is often regarded as the next-generation storage substrate. Therefore, *Ziggurat* [151] integrates *SSD* with *NVM*, instead of SMR, to design a file system that can provide multiple times performance improvement over those on *SSD* alone. Unlike *CosaFS* and *Ziggurat*, *ZoFS* [45] is completely built on top of *NVM* and leverages a new abstraction, named *Coffer*, to enable user-space libraries to take full control of *NVM* in attempting to break the performance bottleneck of the kernel-level *NVM*. Although for these systems, the results of adopting new medium are promising, it is not clear how these techniques are extended to large-scale distributed storage systems.

Unlike the above research, *PolarFS* [38] is a distributed file system advanced by the Alibaba Cloud, it takes full advantage of the emerging techniques like *RDMA*, *NVMe*, and storage performance development kit (*SPDK*) to maximize I/O performance in a large scale, which bears some similarities in spirit with *Ceph*, which is also extended with *NVM* and *RDMA* for I/O performance improvements. *Assise* [152] takes one step further to gain a new insight into how DFS performance is optimized using *NVM* on the client-side with experiments to show its performance advantage over the *NVM* version of *Ceph*.

While new technologies such as *NVM* and burst buffers [156] improve the performance of DFS, they also require researchers to rethink their approaches with respect to data management and I/O operations. Wu *et al.* [150] provides a more detailed description of the strengths and weaknesses of the new storage materials.

5.3 Non-Metadata Service

As for the availability, massive logs and/or data stream backups are required for data consistency in current MDS-based DFSs. Not only does this implementation waste a lot of resources, but the failure of the metadata servers also remains a major barrier to availability. Compared with non-MDS-based DFS, DFS with MDS would employ additional metadata servers, which in turn increases the critical path for data access. Thus, server crashes (specifically, metadata servers) would not only adversely impact the I/O performance but also potentially disrupt the availability of DFS for data access. In contrast, in non-DFS, client often adopts hash function to locate data, avoiding the unavailability caused by the breakdown of metadata servers. Therefore, as an alternative, the non-MDS architecture would become a vital exploration that tends to improve availability in the future.

Ideally, this model can also significantly increase the scalability of the system, enabling the system to achieve linear expansion and growth in terms of concurrency and performance. We believe that the main obstacles of the non-metadata service model may be complex data consistency issues and operations such as inefficient file directory traversal. In addition, the MDS model also greatly reduces the system's ability to globally monitor, and at the same time increase the client's workloads (such as the calculation of file location). How to effectively solve these problems is worthy of further study.

Additionally, for practical applications, the metadata management should not be limited to system-level information. The value-added technology of metadata still has a broad space for development in various specific applications, such as file systems with compression and reduplication, file systems with encryption and security, and so on. Under the condition of ensuring the expected functionality, how to achieve high-efficiency scalability is an important prerequisite for DFS to be deployed and applied in a large scale, and thus, it is worthy of further research and exploration.

Finally, with the advent of the Big Data era, the amount of data is growing exponentially, and the file system at PB-level is gradually failing to meet the demand for storage capacity. The current large-scale file system is designed primarily for EB-level data volumes. In this case, the management of the amount of metadata should also reach the PB-level, correspondingly, which is the level of the previous data volume. This level of metadata undoubtedly poses a huge challenge to metadata management techniques.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

6 Conclusion

In this paper, the state of the art of metadata management in large-scale distributed systems is studied from three aspects: high scalability, high performance and high availability. The advantages and disadvantages of various technologies are comprehensively compared and the future direction is also discussed and commented. It can be seen from the study that high scalability is still the most important area in DFS studies, compared to high performance and high availability. The research focuses squarely on how to balance the namespace access loads and how to achieve resource resiliency. The corresponding technologies can be extended to some other aspects as well. Although the existing technologies have achieved scalability to some extent in the face of the rapid growth of Big Data, there are still huge gaps and challenges relative to the demands on them. This forces us to consider the management of metadata from a deeper perspective so as to develop a technological framework to meet the needs of the demands today and more so—tomorrow.

References

[1] J. Sun and C. K. Reddy, "Big data analytics for healthcare," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2013, Art. no. 1525.

[2] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with Big Data: A deep learning approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 865–873, Apr. 2015.

[3] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Inf. Sci.*, vol. 275, pp. 314–347, 2014.

[4] Apache, "HDFS," [EB/OL], 2021. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. 19th ACM Symp. Oper. Syst. Princ.*, 2003, pp. 29–43.

[6] OpenSFS and EOFS, "Lustre," [EB/OL], 2021. [Online]. Available: http://lustre.org/

[7] IBM, "Ibm spectrum scale," [EB/OL], 2021. [Online]. Available: https://www.ibm.com/products/scale-out-file-and-object-storage

[8] F. C. for high performance computing, "Beegfs," [EB/OL], 2021. [Online]. Available: https://www.beegfs.io/content/

[9] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Oper. Syst. Des. Implementation*, 2006, pp. 307–320.

[10] T. G. Community, "Gluster inc," [EB/OL], 2021. [Online]. Available: https://www.gluster.org/

[11] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2018.

[12] E. L. Miller, K. Greenan, A. Leung, D. Long, and A. Wildani, "Reliable and efficient metadata storage and indexing using nvram," [EB/OL], 2021. [Online]. Available: http://dcslab.hanyang.ac.kr/nvramos08/EthanMiller.pdf

[13] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "A trace-driven analysis of the unix 4.2 BSD file system," in *Proc. 10th ACM Symp. Oper. Syst. Princ.*, 1985, pp. 15–24.

[14] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," *ACM Trans. Storage*, vol. 3, no. 3, pp. 9–es, Oct. 2007.

[15] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spyglass: Fast, scalable metadata search for large-scale storage systems," *FAST*, vol. 9, pp. 153–166, 2009.

[16] J. R. Douceur and W. J. Bolosky, "A large-scale study of file-system contents," in *Proc. ACM Sigmetrics Int. Conf. Meas. Model. Comput. Syst.*, 1999, pp. 59–70.

[17] L. Xiao *et al.*, "File creation optimization for metadata-intensive application in file systems," in *Proc. ICA3PP Int. Workshops Symp. Algorithms Architectures Parallel Process.*, 2015, pp. 353–363.

[18] B. Welch and G. Noer, "Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions," in *Proc. IEEE 29th Symp. Mass Storage Syst. Technol.*, 2013, pp. 1–12.

[19] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer, "Passive nfs tracing of email and research workloads," in *Proc. USENIX File Storage Technol. Conf.*, 2003, pp. 203–216.

[20] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *Proc. IEEE Int. Symp. Workload Characterization*, 2008, pp. 119–128.

[21] S. R. Alam, H. N. El-Harake, K. Howard, N. Stringfellow, and F. Verzelloni, "Parallel I/O and the metadata wall," in *Proc. 6th Workshop Parallel Data Storage*, 2011, pp. 13–18.

[22] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A fast file system for unix," *ACM Trans. Comput. Syst.*, vol. 2, no. 3, pp. 181–197, Aug. 1984.

[23] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, "Serverless network file systems," *ACM Trans. Comput. Syst.*, vol. 14, no. 1, p. 41–79, Feb. 1996.

[24] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A scalable distributed file system," in *Proc. 16th ACM Symp. Oper. Syst. Princ.*, 1997, pp. 224–237.

[25] G. A. Gibson *et al.*, "A cost-effective, high-bandwidth storage architecture," in *Proc. 8th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 1998, pp. 92–103.

[26] S. Niazi, M. Ismail, S. Haridi, J. Dowling, S. Grohsschmiedt, and M. Ronström, "HopsFS: Scaling hierarchical file system metadata using NewSQL databases," in *Proc. 15th USENIX Conf. File Storage Technol.*, 2017, pp. 89–104.

[27] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The log-structured merge-tree (LSM-tree)," *Acta Inf.*, vol. 33, no. 4, pp. 351–385, Jun. 1996.

[28] S. Ma *et al.*, "ROART: Range-query optimized persistent ART," in *19th USENIX Conf. File Storage Technol. (FAST 21). USENIX Assoc.*, 2021, pp. 1–16.

[29] H. Liu *et al.*, "CFS: A distributed file system for large scale container platforms," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 1729–1742.

[30] J. Zhang, C. Si, Y. Jia, J. Zhang, X. Han, and L. Xu, "Volume based metadata isolation in blue whale cluster file system," in *Proc. 11th IEEE Int. Conf. High Perform. Comput. Commun.*, 2009, pp. 654–658.

[31] J. Liu, R. Wang, X. Gao, X. Yang, and G. Chen, "AngleCut: A ring-based hashing scheme for distributed metadata management," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2017, pp. 71–86.

[32] D. Roselli, J. R. Lorch, and T. E. Anderson, "A comparison of file system workloads," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2000, Art. no. 4.

[33] Y. Qian *et al.*, "A configurable rule based classful token bucket filter network request scheduler for the lustre file system," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2017, pp. 1–12.

[34] Y. Qian *et al.*, "LPCC: Hierarchical persistent client caching for lustre," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2019, pp. 1–14.

[35] M. Sato *et al.*, "Co-design for A64FX manycore processor and "fugaku,"" in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2020, pp. 1–15.

[36] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70.

[37] J. Zhou, Y. Chen, W. Xie, D. Dai, S. He, and W. Wang, "PRS: A pattern-directed replication scheme for heterogeneous object-based storage," *IEEE Trans. Comput.*, vol. 69, no. 4, pp. 591–605, Apr. 2020.

[38] W. Cao *et al.*, "PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database," *Proc. Very Large Data Base Endowment*, vol. 11, no. 12, pp. 1849–1862, Aug. 2018.

[39] H.-S. P. Wong *et al.*, "Phase change memory," *Proc. IEEE Proc. IRE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[40] J. Chen, Q. Wei, C. Chen, and L. Wu, "FSMAC: A file system metadata accelerator with non-volatile memory," in *Proc. IEEE 29th Symp. Mass Storage Syst. Technol.*, 2013, pp. 1–11.

[41] S. He, X.-H. Sun, Y. Wang, A. Kougkas, and A. Haider, "A heterogeneity-aware region-level data layout for hybrid parallel file systems," in *Proc. IEEE 44th Int. Conf. Parallel Process.*, 2015, pp. 340–349.

[42] H.-S. Chang, Y.-H. Chang, P.-C. Hsiu, T.-W. Kuo, and H.-P. Li, "Marching-based wear-leveling for PCM-based storage systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 2, pp. 1–22, Mar. 2015.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

[43] L. Zeng, Z. Zhang, Y. Wang, D. Feng, and K. B. Kent, "COSAFS: A cooperative shingle-aware file system," *ACM Trans. Storage*, vol. 13, no. 4, pp. 1–23, Nov. 2017

[44] E. Kakoulli and H. Herodotou, "OctopusFS: A distributed file system with tiered storage management," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 65–78.

[45] M. Dong, H. Bu, J. Yi, B. Dong, and H. Chen, "Performance and protection in the zoFS user-space NVM file system," in *Proc. 27th ACM Symp. Oper. Syst. Princ.*, 2019, pp. 478–493.

[46] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *Proc. ACM/IEEE Conf. Supercomput.*, 2004, Art. no. 53.

[47] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in *Proc. ACM/IEEE Conf. Supercomput.*, 2004, Art. no. 4.

[48] S. A. Brandt, E. L. Miller, D. D. Long, and L. Xue, "Efficient metadata management in large distributed storage systems," in *Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Syst. Technol.*, 2003, pp. 290–298.

[49] K. Ren, Q. Zheng, S. Patil, and G. Gibson, "IndexFS: Scaling file system metadata performance with stateless caching and bulk insertion," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2014, pp. 237–248.

[50] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, p. 158–171, Jul. 2013.

[51] C. Juszczak, "Improving the performance and correctness of an NFS server," in *Proc. Winter USENIX Tech. Conf.*, 1989, pp. 52–64.

[52] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith, "Andrew: A distributed personal computing environment," *Communication ACM*, vol. 29, no. 3, p. 184–201, Mar. 1986.

[53] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Trans. Comput.*, vol. 39, no. 4, p. 447–459, Apr. 1990.

[54] Apache, "HDFS federation," [EB/OL], 2021. [Online]. Available http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html

[55] P. F. Corbett and D. G. Feitelson, "The vesta parallel file system," *ACM Trans. Comput. Syst.*, vol. 14, no. 3, p. 225–264, Aug. 1996.

[56] O. Rodeh and A. Teperman, "ZFS - a scalable distributed file system using object disks," in *Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Syst. Technol.*, 2003, 2003, pp. 207–218.

[57] Q. Xu, R. V. Arumugam, K. L. Yong, and S. Mahadevan, "Efficient and scalable metadata management in EB-scale file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2840–2850, Nov. 2014.

[58] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. 34th Annu. ACM Symp. Theory Comput.*, 2002, pp. 380–388.

[59] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 141–150.

[60] S. Pan *et al.*, "Facebook's tectonic filesystem: Efficiency from exascale," in *Proc. 19th USENIX Conf. File Storage Technol.*, 2021, pp. 217–231.

[61] S. Patil and G. Gibson, "Scale and concurrency of GIGA: File system directories with millions of files," in *Proc. 9th USENIX Conf. File Stroage Technol.*, 2011, Art. no. 13.

[62] Y. Gao, X. Gao, X. Yang, J. Liu, and G. Chen, "An efficient ring-based metadata management policy for large-scale distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 1962–1974, Sep. 2019.

[63] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. Conf. Appl. Technol. Archit. Protoc. Comput. Commun.*, 2001, pp. 149–160.

[64] K. Mckusick and S. Quinlan, "GFS: Evolution on fast-forward," *ACM Queue*, vol. 53, no. 3, 2009, Art. no. 10.

[65] J. Palencia, R. Budden, and K. Sullivan, "Kerberized lustre 2.0 over the wan," in *Proc. TeraGrid Conf.*, 2010, pp. 1–5.

[66] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, scalable, decentralized placement of replicated data," in *Proc. ACM/IEEE Conf. Supercomput.*, 2006, pp. 122–es.

[67] L. Wang, Y. Zhang, J. Xu, and G. Xue, MAPX: Controlled data migration in the expansion of decentralized object-based storage systems, in *Proc. USA: USENIX Assoc.*, 2020, p. 1–12.

[68] J. Liu, J. Zhang, B. Shao, H. Dong, Z. Liu, and L. Xu, "Metadata server clustering system for eb-scale storage," *Scientia Sinica Informationis*, vol. 45, no. 6, pp. 721–738, 2015.

[69] T. Kim and S. H. Noh, "pNFS for everyone: An empirical study of a low-cost, highly scalable networked storage," *Int. J. Comput. Sci. Netw. Secur.*, vol. 14, 2014, Art. no. 52.

[70] M. A. Sevilla *et al.*, "Mantle: A programmable metadata load balancer for the ceph file system," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–12.

[71] M. Mitzenmacher, A. W. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*. Norwell, MA, USA: Kluwer, 2000, pp. 255–312.

[72] Google, "Leveldb," [EB/OL], 2021. [Online]. Available: https://github.com/google/leveldb

[73] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, Jun. 2008.

[74] K. Hiraga, O. Tatebe, and H. Kawashima, "PPMDS: A distributed metadata server based on nonblocking transactions," in *Proc. 5th Int. Conf. Social Netw. Anal.*, 2018, pp. 202–208.

[75] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *Proc. 1st USENIX Conf. File Storage Technol.*, 2002, Art. no. 16.

[76] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Supporting scalable and adaptive metadata management in ultralarge-scale file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 4, pp. 580–593, Apr. 2011.

[77] C. Rodríguez-Quintana, A. F. Díaz, J. Ortega, R. H. Palacios, and A. Ortiz, "A new scalable approach for distributed metadata in HPC," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2016, pp. 106–117.

[78] I. Kettaneh *et al.*, "The network-integrated storage system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 486–500, Feb. 2020.

[79] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch, "The sprite network operating system," *Computer*, vol. 21, no. 2, pp. 23–36, Feb. 1988.

[80] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg, "Ibm storage tank-a heterogeneous scalable san file system," *IBM Syst. J.*, vol. 42, no. 2, pp. 250–267, 2003.

[81] P. Braam *et al.*, "The intermezzo file system," in *Proc. 3rd Perl Conf.*, 1999, pp. 1–10.

[82] E. L. Miller and R. H. Katz, "RAMA: An easy-to-use, high-performance parallel file system," *Parallel Comput.*, vol. 23, no. 4–5, p. 419–446, Jun. 1997.

[83] J. R. Douceur and J. Howell, "Distributed directory service in the farsite file system," in *Proc. 7th Symp. Oper. Syst. Des. Implementation*, 2006, pp. 321–334.

[84] R. G. Ross, "Cluster storage for commodity computation," University of Cambridge, Computer Laboratory, Cambridge, U.K., Tech. Rep. 690, 2007.

[85] Z. Cao, H. Wen, X. Ge, J. Ma, J. Diehl, and D. H. C. Du, "TDDFS: A tier-aware data deduplication-based file system," *ACM Trans. Storage*, vol. 15, no. 1, pp. 1–26, Feb. 2019.

[86] L. Xiao, K. Ren, Q. Zheng, and G. A. Gibson, "Shardfs vs. indexfs: Replication vs. caching strategies for distributed metadata management in cloud storage systems," in *Proc. Sixth ACM Symp. Cloud Comput.*, 2015, pp. 236–249.

[87] PDL.CMU, "Shardfs," [EB/OL], 2021. [Online]. Available: https://www.pdl.cmu.edu/ShardFS/index.shtml

[88] D. Oleg, "Writeback cache for lustre," [EB/OL], 2020. [Online]. Available: https://www.eofs.eu/_media/events/lad18/17_oleg_drokin_wbcache-lad18.pdf

[89] W. Cheng, C. Li, L. Zeng, Y. Qian, X. Li, and A. Brinkmann, "Nvmm-oriented hierarchical persistent client caching for lustre," *ACM Trans. Storage*, vol. 17, no. 1, pp. 1–22, Jan. 2021

[90] L. Pineda-Morales, A. Costan, and G. Antoniu, "Towards multi-site metadata management for geographically distributed cloud workflows," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2015, pp. 294–303.

[91] P. Matri, M. S. Pérez, A. Costan, and G. Antoniu, "Týrfs: Increasing small files access performance with dynamic metadata replication," in *Proc. 18th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2018, pp. 452–461.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

[92] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 654–663.

[93] Z. Li, R. Xue, and L. Ao, "Replichard: Towards tradeoff between consistency and performance for metadata," in *Proc. Int. Conf. Supercomput.*, 2016, pp. 1–11.

[94] M. Bravo, L. Rodrigues, and P. Van Roy, "Saturn: A distributed metadata service for causal consistency," in *Proc. 12th Eur. Conf. Comput. Syst.*, 2017, pp. 111–126.

[95] M. Raynal and A. Schiper, "From causal consistency to sequential consistency in shared memory systems," in *Proc. Int. Conf. Found. Softw. Technol. Theor. Comput. Sci.*, 1995, pp. 180–194.

[96] A. Ganesan, R. Alagappan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Redundancy does not imply fault tolerance: Analysis of distributed storage reactions to file-system faults," *ACM Trans. Storage*, vol. 13, no. 3, pp. 1–33, Sep. 2017.

[97] M.-H. Cha, D.-O. Kim, H.-Y. Kim, and Y.-K. Kim, "Adaptive metadata rebalance in exascale file system," *J. Supercomput.*, vol. 73, no. 4, pp. 1337–1359, 2017.

[98] S. Cao, Y. Gao, X. Gao, and G. Chen, "AdaM: An adaptive fine-grained scheme for distributed metadata management," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–10.

[99] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. 4th Int. Conf. Learn. Representations*, 2016, pp. 1–14.

[100] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communication ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[101] A. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. Miller, "High-performance metadata indexing and search in petascale data storage systems," in *Proc. J. Phys. Conf. Ser.*, vol. 125, no. 1, 2008, Art. no. 012069.

[102] M. Mäsker, T. Süß, L. Nagel, L. Zeng, and A. Brinkmann, "Hyperion: Building the largest in-memory search tree," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 1207–1222.

[103] Y. Sun, Y. Hua, S. Jiang, Q. Li, S. Cao, and P. Zuo, "SmartCuckoo: A fast and cost-efficient hashing index scheme for cloud storage systems," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2017, pp. 553–565.

[104] J. Bruck, J. Gao, and A. Jiang, "Weighted bloom filter," in *Proc. IEEE Int. Symp. Inf. Theory Proc.*, 2006, pp. 2304–2308.

[105] J. K. Mullin, "A second look at bloom filters," *Commun. ACM*, vol. 26, no. 8, pp. 570–571, Aug. 1983.

[106] H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol, "SoMeta: Scalable object-centric metadata management for high performance computing," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2017, pp. 359–369.

[107] Z. Huo *et al.*, "Mbfs: A parallel metadata search method based on bloomfilters using MapReduce for large-scale file systems," *J. Supercomput.*, vol. 72, no. 8, pp. 3006–3032, Aug. 2016.

[108] P. Grégoire, "Lustre 2.8 feature: Multiple metadata modify RPCS in parallel," [EB/OL], 2020. [Online]. Available: https://www.eofs.eu/_media/events/lad15/14_gregoire_pichon_lad2015_lustre_2_8_multiple_modify_rpcs.pdf

[109] J. Liu, D. Feng, Y. Hua, B. Peng, and Z. Nie, "Using provenance to efficiently improve metadata searching performance in storage systems," *Future Gener. Comput. Syst.*, vol. 50, pp. 99–110, 2015.

[110] J. Liu, D. Feng, Y. Hua, B. Peng, P. Zuo, and Y. Sun, "P-index: An efficient searchable metadata indexing scheme based on data provenance in cold storage," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2015, pp. 597–611.

[111] G. Wu, Y. Deng, and X. Qin, "Using provenance to boost the metadata prefetching in distributed storage systems," in *Proc. IEEE 34th Int. Conf. Comput. Des.*, 2016, pp. 80–87.

[112] Y. Chen, C. Li, M. Lv, X. Shao, Y. Li, and Y. Xu, "Explicit data correlations-directed metadata prefetching method in distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2692–2705, Dec. 2019.

[113] L. Xu, Z. Huang, H. Jiang, L. Tian, and D. Swanson, "VSFS: A searchable distributed file system," in *Proc. 9th Parallel Data Storage Workshop*, 2014, pp. 25–30.

[114] C. Kulkarni, A. Kesavan, T. Zhang, R. Ricci, and R. Stutsman, "Rocksteady: Fast migration for low-latency in-memory storage," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 390–405.

[115] S. Dong, A. Kryczka, Y. Jin, and M. Stumm, "Evolution of development priorities in key-value stores serving large-scale applications: The RocksDB experience," in *Proc. 19th USENIX Conf. File Storage Technol.*, 2021, pp. 33–49.

[116] D.-Y. Lee, K. Jeong, S.-H. Han, J.-S. Kim, J.-Y. Hwang, and S. Cho, "Understanding write behaviors of storage backends in ceph object store," in *Proc. IEEE Int. Conf. Massive Storage Syst. Technol.*, 2017, pp. 1–10.

[117] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "File systems unfit as distributed storage backends: Lessons from 10 years of ceph evolution," in *Proc. 27th ACM Symp. Oper. Syst. Princ.*, 2019, pp. 353–369.

[118] A. B. M. Moniruzzaman, "NewSQL: Towards next-generation scalable RDBMS for online transaction processing (OLTP) for Big Data management," *Int. J. Database Theory Appl.*, vol. 7, no. 6, pp. 121–130, 2014.

[119] T. Wang *et al.*, "MetaKV: A key-value store for metadata management of distributed burst buffers," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2017, pp. 1174–1183.

[120] S. Li, F. Liu, J. Shu, Y. Lu, T. Li, and Y. Hu, "A flattened metadata service for distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2641–2657, Dec. 2018.

[121] D. Zhao, K. Qiao, Z. Zhou, T. Li, Z. Lu, and X. Xu, "Toward efficient and flexible metadata indexing of Big Data systems," *IEEE Trans. Big Data*, vol. 3, no. 1, pp. 107–117, Mar. 2017.

[122] Y. Wang, H. Li, and M. Hu, "Reusing garbage data for efficient workflow computation," *Comput. J.*, vol. 58, no. 1, pp. 110–125, 2015.

[123] Y. Wang and P. Lu, "Dataflow detection and applications to workflow scheduling," *Concurrency Comput. Pract. Experience*, vol. 23, no. 11, pp. 1261–1283, 2011.

[124] Y. Wang, P. Lu, and K. B. Kent, "WaFS: A workflow-aware file system for effective storage utilization in the cloud," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2716–2729, Sep. 2015.

[125] D. Zhao *et al.*, "FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems," in *Proc. IEEE Int. Conf. Big Data*, 2014, pp. 61–70.

[126] D. Zhao, N. Liu, D. Kimpe, R. Ross, X.-H. Sun, and I. Raicu, "Towards exploring data-intensive scientific applications at extreme scales through systems and simulations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 6, pp. 1824–1837, Jun. 2016.

[127] D. Dai, R. B. Ross, P. Carns, D. Kimpe, and Y. Chen, "Using property graphs for rich metadata management in HPC systems," in *Proc. 9th Parallel Data Storage Workshop*, 2014, pp. 7–12.

[128] D. Dai, Y. Chen, P. Carns, J. Jenkins, W. Zhang, and R. Ross, "Managing rich metadata in high-performance computing systems using a graph model," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1613–1627, Jul. 2019.

[129] H. Sim, A. Khan, S. S. Vazhkudai, S.-H. Lim, A. R. Butt, and Y. Kim, "An integrated indexing and search service for distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, pp. 2375–2391, Oct. 2020.

[130] J. Im, J. Bae, C. Chung, Arvind, and S. Lee, "PinK: High-speed in-storage key-value store with bounded tails," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 173–187.

[131] D. Stamatakis, N. Tsikoudis, E. Micheli, and K. Magoutis, "A general-purpose architecture for replicated metadata services in distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2747–2759, Oct. 2017.

[132] M. Ji, E. W. Felten, R. Wang, and J. P. Singh, "Archipelago: An island-based file system for highly available and scalable Internet services," in *Proc. 4th Conf. USENIX Windows Syst. Symp.*, 2000, Art. no. 1.

[133] Z. Chen, J. Xiong, and D. Meng, "Replication-based highly available metadata management for cluster file systems," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2010, pp. 292–301.

[134] J. Zhou, Y. Chen, W. Wang, and D. Meng, "MAMS: A highly reliable policy for metadata service," in *Proc. 44th Int. Conf. Parallel Process.*, 2015, pp. 729–738.

[135] J. Zhou, Y. Chen, W. Wang, S. He, and D. Meng, "A highly reliable metadata service for large-scale distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 374–392, Feb. 2020.

[136] A. Thomson and D. J. Abadi, "CalvinFS: Consistent WAN replication and scalable metadata management for distributed file systems," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 1–14.

[137] L. Lamport, "Paxos made simple," *ACM SIGACT News Distrib. Comput. Column* vol. 32, no. 4, pp. 51–58, 2001.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.

[138] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998.
[139] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proc. 7th Symp. Oper. Syst. Des. Implementation*, 2006, pp. 335–350.
[140] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 305–320.
[141] Apache, "Kudu," [EB/OL], 2021. [Online]. Available: https://kudu.apache.org/
[142] X. Wu and Z. Shao, "Selfie: Co-locating metadata and datato enable fast virtual block devices," in *Proc. 8th ACM Int. Syst. Storage Conf.*, 2015, pp. 1–11.
[143] S. Jha *et al.*, "Live forensics for HPC systems: A case study on distributed storage systems," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Analy.*, 2020, pp. 1–16.
[144] Z. Liu *et al.*, "DistCache: Provable load balancing for large-scale storage systems with distributed caching," in *Proc. 17th USENIX Conf. File Storage Technol.*, 2019, pp. 143–157.
[145] J. Bae *et al.*, "FlashNeuron: SSD-Enabled large-batch training of very deep neural networks," in *Proc. 19th USENIX Conf. File Storage Technol.*, 2021, pp. 387–401.
[146] A. V. Kumar and M. Sivathanu, "Quiver: An informed storage cache for deep learning," in *Proc. 18th USENIX Conf. File Storage Technol.*, 2020, pp. 283–296.
[147] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 1–33, Mar. 2020.
[148] Y. Gao, X. Gao, and G. Chen, "Deephash: An end-to-end learning approach for metadata management in distributed file systems," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–10.
[149] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *Trans. Neural Netw.*, vol. 9, no. 5, Sep. 1998, Art. no. 1054.
[150] Z.-Q. Wu, J. Wei, F. Zhang, W. Guo, and G.-W. Xie, "MDLB: A metadata dynamic load balancing mechanism based on reinforcement learning," *Front. Inf. Technol. Electron. Eng.*, vol. 21, no. 7, pp. 1034–1046, 2020.
[151] S. Zheng, M. Hoseinzadeh, and S. Swanson, "Ziggurat: A tiered file system for non-volatile main memories and disks," in *Proc. 17th USENIX Conf. File Storage Technol.*, 2019, pp. 207–219.
[152] T. E. Anderson *et al.*, "Assise: Performance and availability via client-local NVM in a distributed file system," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implementation*, 2020, pp. 1011–1027.
[153] W. He and D. H. Du, "SMaRT: An approach to shingled magnetic recording translation," in *Proc. 15th USENIX Conf. File Storage Technol.* 2017, pp. 121–134.
[154] J. Im, J. Bae, C. Chung, Arvind, and S. Lee, "Design of LSM-tree-based key-value SSDs with bounded tails," *ACM Trans. Storage*, vol. 17, no. 2, pp. 1–27, May 2021.
[155] J. H. Lee, H. Zhang, V. Lagrange, P. Krishnamoorthy, X. Zhao, and Y. S. Ki, "SmartSSD: FPGA accelerated near-storage data analytics on SSD," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 110–113, Jul. 2020.
[156] N. Liu *et al.*, "On the role of burst buffers in leadership-class storage systems," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, 2012, pp. 1–11.

Hao Dai received the MSc degree in communication and electronic technology from the Wuhan University of Technology, in 2017. He is currently working toward the PhD degree with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interests include cloud computing, Big Data processing, mobile edge computing systems.

Yang Wang received the BSc degree in applied mathematics from the Ocean University of China, in 1989, the MSc degree in computer science from Carlton University, in 2001, and the PhD degree in computer science from the University of Alberta, Canada, in 2008. He is currently with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, as a full professor and with Xiamen University, China as an adjunct professor. His research interests include service and cloud computing, programming language implementation, and software engineering. He is an Alberta Industry R&D Associate (2009-2011), and a Canadian Fulbright Scholar (2014-2015).

Kenneth B. Kent (Senior Member, IEEE) received the BSc degree from the Memorial University of Newfoundland, Canada, in 1996, and the MSc and PhD degrees from the University of Victoria, Canada, in 1999 and 2003, respectively. He is currently a professor with the Faculty of Computer Science, University of New Brunswick, and the director of IBM Centre for Advanced Studies-Atlantic, Canada. His research interests include hardware/software co-design, virtual machines, reconfigurable computing, and embedded systems.

Lingfang Zeng received the BS degree in computer application from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2000, the MS degree in computer application from the China University of Geoscience, China, in 2003, and the PhD degree in computer architecture from HUST, in 2006. He was research fellow with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore, during 2007-2008 and 2010-2013, and a visiting professor with the Johannes Gutenberg University Mainz, Germany (2016-2018). He is currently with Zhejiang Lab as a PI. He publishes more than 60 papers in major journals and conferences.

Chengzhong Xu (Fellow, IEEE) received the PhD degree from the University of Hong Kong, in 1993. He is currently the dean of the Faculty of Science and Technology, University of Macau, China, and the director of the Institute of Advanced Computing and Data Engineering, Shenzhen Institutes of Advanced Technology of Chinese Academy of Sciences. His research interests include parallel and distributed systems, service and cloud computing, and software engineering. He has published more than 200 papers in journals and conferences. He serves on a number of journal editorial boards, including the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *Journal of Parallel and Distributed Computing* and *China Science Information Sciences*.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Shenzhen Institute of Advanced Technology CAS. Downloaded on September 15,2023 at 07:29:40 UTC from IEEE Xplore. Restrictions apply.