LocoFS: A Loosely-Coupled Metadata Service for Distributed File Systems

Siyang Li*
Tsinghua University
lisiyang@tsinghua.edu.cn

Youyou Lu
Tsinghua University
luyouyou@tsinghua.edu.cn

Jiwu Shu†
Tsinghua University
shujw@tsinghua.edu.cn

Tao Li
University of Florida
taoli@ece.ufl.edu

Yang Hu
University of Texas, Dallas
huyang.ece@ufl.edu

ABSTRACT

Key-Value stores provide scalable metadata service for distributed file systems. However, the metadata's organization itself, which is organized using a directory tree structure, does not fit the key-value access pattern, thereby limiting the performance. To address this issue, we propose a distributed file system with a loosely-coupled metadata service, LocoFS, to bridge the performance gap between file system metadata and key-value stores. LocoFS is designed to decouple the dependencies between different kinds of metadata with two techniques. First, LocoFS decouples the directory content and structure, which organizes file and directory index nodes in a flat space while reversely indexing the directory entries. Second, it decouples the file metadata to further improve the key-value access performance. Evaluations show that LocoFS with eight nodes boosts the metadata throughput by 5 times, which approaches 93% throughput of a single-node key-value store, compared to 18% in the state-of-the-art IndexFS.

KEYWORDS

Key-value stores, File systems management, Distributed storage, Distributed architectures

ACM Reference format:
Siyang Li, Youyou Lu, Jiwu Shu, Tao Li, and Yang Hu. 2017. LocoFS: A Loosely-Coupled Metadata Service for Distributed File Systems. In *Proceedings of SC17, Denver, CO, USA, November 12–17, 2017,* 12 pages.
https://doi.org/10.1145/3126908.3126928

*Also with State Key Laboratory of Mathematical Engineering and Advanced Computing.
†Jiwu Shu is the corresponding author.

SC17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5114-0/17/11…$15.00
https://doi.org/10.1145/3126908.3126928

1 INTRODUCTION

As clusters or data centers are moving from Petabyte level to Exabyte level, distributed file systems are facing challenges in metadata scalability. The recent work IndexFS [38] uses hundreds of metadata servers to achieve high-performance metadata operations. However, most of the recent active super computers only deploy 1 to 4 metadata servers to reduce the complexity of management and guarantee reliability. Besides, previous work [24, 39] has also revealed that metadata operations consume more than half of all operations in file systems. The metadata service plays a major role in distributed file systems. It is important to support parallel processing large number of files with a few number of metadata servers. Unfortunately, inefficient scalable metadata service cannot utilize the performance of key-value (KV) stores in each metadata server node, thus degrades throughput.

On the other hand, KV stores have been introduced to build file systems [20, 37, 38, 43, 53]. They not only export a simple interface (i. e. , `get` and `put`) to users, but also use efficient data organization (e. g. , Log-Structured Merge Tree [34]) in the storage. Since data values are independent and are organized in such a simple way, KV stores enable small objects to be accessed efficiently and provide excellent scalability, which makes them a promising technique for file system metadata servers. The advantage of KV stores have been leveraged in file system metadata (e. g. , `inode` and `dirent`) for small objects [18, 38].

However, we observe that there is a huge performance gap between KV stores and file system metadata, even for those file systems that have been optimized using KV stores. For example, in a single server, the LevelDB KV store [3] can achieve performance at 128K IOPS for random put operations and 190K IOPS for random get operations [16]. Nevertheless, IndexFS [38], which stores file system metadata using LevelDB and show much better scalability than traditional distributed file systems, only achieves 6K IOPS that is 1. 7% of LevelDB for create operations per node.

We identify that file metadata accesses have strong dependencies due to the semantics of a directory tree. The limitation is transferred from local KV store to network latency because of the complicated communication within metadata operation. For instance, a file `create` operation needs to write at least three locations in the metadata: its file `inode`, its `dirent` and `inode`. In a local file system, these update operations occur in one node, the cost of the file operation in software layer is mainly in the data organization itself. However, in distributed file system, the main

performance bottleneck is caused by the network latency among different nodes. Recent distributed file systems distribute metadata either to data servers [13, 26, 40, 47] or to a metadata server cluster (MDS cluster) [11, 31, 40, 52] to scale the metadata service. In such distributed metadata services, a metadata operation may need to access multiple server nodes. Considering these accesses have to be atomic [9, 26, 48, 51] or performed in correct order [27] to keep consistency, a file operation needs to traverse different server nodes or access a single node many times. Under such circumstance, the network latency can severely impact the inter-node access performance of distributed metadata service.

Our goal in this paper is two-fold, (1) to reduce the network latency within metadata operation; (2) to fully utilize KV-store's performance benefits. Our key idea is to reduce the dependencies among file system metadata (i. e. , the logical organization of file system metadata), ensuring that important operation only communicates with one or two metadata servers during its life cycle.

To such an end, we propose *LocoFS* a loosely-coupled metadata service in a distributed file system, to reduce network latency and improve utilization of KV store. LocoFS first cuts the file metadata (i. e. , file `inode`) from the directory tree. These file metadata are organized independently and they form a flat space, where the `dirent-inode` relationship for files are kept with the file `inode` using the form of reverted index. This *flattened directory tree* structure matches the KV access patterns better. LocoFS also divides the file metadata into two parts: access part and content part. This partition in the file metadata further improves the utilization of KV stores for some operations which only use part of metadata. In such ways, LocoFS reorganizes the file system directory tree with reduced dependency, enabling higher efficiency in KV based accesses. Our contributions are summarized as follows:

(1) We propose a flattened directory tree structure to decouple the file metadata and directory metadata. The flattened directory tree reduces dependencies among metadata, resulting in lower latency.
(2) We also further decouple the file metadata into two parts to make their accesses in a KV friendly way. This separation further improves file system metadata performance on KV stores.
(3) We implement and evaluate LocoFS. Evaluations show that LocoFS achieves 100K IOPS for file `create` and `mkdir` when using one metadata server, achieve 38% of KV-store's performance. LocoFS also achieves low latency and maintains scalable and stable performance.

The rest of this paper is organized as follows. Section 2 discusses the implication of directory structure in distributed file system and the motivation of this paper. Section 3 describes the design and implementation of the proposed loosely-coupled metadata service, LocoFS. It is evaluated in Section 4. Related work is given in Section 5, and the conclusion is made in Section 6.

2 MOTIVATION

In this section, we first demonstrate the huge performance gap between distributed file system (DFS) metadata and key-value (KV) stores. We then explore the design of current DFS directory tree to identify the performance bottlenecks caused by the latency and scalability.

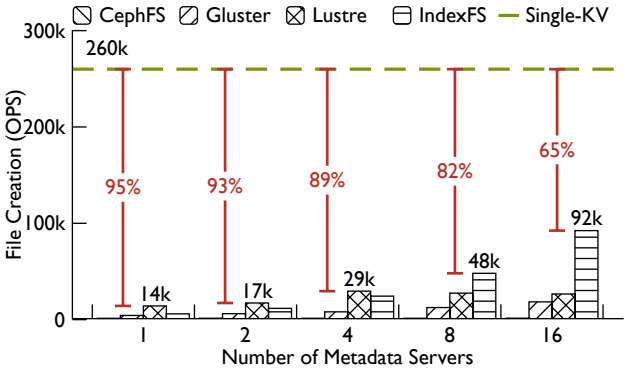

Figure 1: Performance Gap between File System Metadata (Lustre, CephFS and IndexFS) and KV Stores (Kyoto Cabinet (Tree DB)).

2.1 Performance Gap Between FS Metadata and KV Store

There are four major schemes for metadata management in distributed file system: single metadata server (single-MDS) scheme (e.g., HDFS), multi-metadata servers schemes (multi-MDS) with hash-based scheme(e.g., Gluster [13]), directory-based scheme (e.g., CephFS [46]), stripe-based scheme (e.g., Lustre DNE, Giga+ [36]). Comparing with directory-based scheme, hash-based and stripe-based schemes achieve better scalability but sacrifice the locality on single node. One reason is that the multi-MDS schemes issue multiple requests to the MDS even if these requests are located in the same server. As shown in figure 1 compared with the KV, the file system with single MDS on both one node (95% IOPS degradation) and mulitple nodes (65% IOPS degradation on 16 nodes).

From the figure 1 , we can also see that IndexFS, which stores metadata using LevelDB, achieves an IOPS that is only 1. 6% of LevelDB [16], when using one single server. To achieve the Kyoto Cabinet's performance on a single server, IndexFS needs to scale-out to 32 servers. Therefore, there is still a large headroom to exploit the performance benefits of key-value stores in the file system metadata service.

2.2 Problems with File System Directory Tree

We further study the file system directory tree structure to understand the underline reasons of the huge performance gap. We find that the cross-server operations caused by strong dependencies among DFS metadata dramatically worsen the metadata performance. We identify two major problems as discussed in the following.

2.2.1 Long Locating Latency. Distributed file systems spread metadata to multiple servers to increase the metadata processing capacity. A metadata operation needs to communicate with multiple metadata servers, and this may lead to high latency of metadata operations. Figure 2 shows an example of metadata operation in a distributed file system with distributed metadata service. In this example, `inodes` are distributed in server n1 to n4. If there is a request to access file 6, the file system client has to access *n*1 first to read dir 0, and then read dir 1, 5 and 6 sequentially. When these

servers are connected using ethernet with a 100μs latency, the file 6 access request consumes at least 400μs. This situation occurs in IndexFS, Giga+ and CephFS. To mitigate this problem, all the file systems cache `inodes` in the clients to reduce the overhead of locating a file. However, these file systems still suffer from the long locating latency issue when there are cache misses.

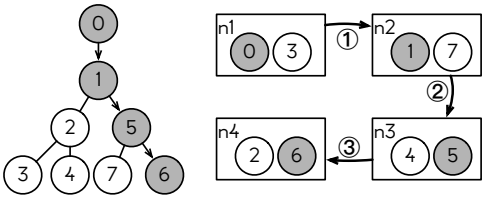

Figure 2: Locating a File or Directory: The left half shows an example of directory tree. In the example to locate a file/directory 6, it has to access four servers as shown in the right half. This dependency of directory tree metadata causes long locating latency.

Considering a file system with 1 millions IOPS (e.g., IndexFS achieves 0. 85 millions IOPS using 128 metadata servers) and 200-byte metadata, it only consumes 190MB bandwidth from clients to servers, which are far from saturating the bandwidth of mainstream Ethernet (e.g., 10Gbps) nowadays. In other words, the network bandwidth is not the bottleneck, while the network latency caused by the data access pattern needs to be carefully designed. For key-value stores that are recently introduced for metadata storage of file systems, the latency of a local `get` operation is 4μs. In Ethernet with TCP/IP, the latency with an RTT (Round-Trip Time) is 100μs, which is 25 times of the key-value `get` latency. More complex network links lead to higher latency.

In conclusion, the cross-server metadata operations that need to follow the directory tree dependencies makes the logical metadata organization inefficient.

2.2.2 High Overhead in KV (De)Serialization. The size of a key-value record has strong impacts on key-value store's performance. We measure both LevelDB and Kyoto Cabinet, and the two KV stores show poorer performance with large value sizes. To understand this result, we also find that the serialization and de-serialization of writing and reading data values consume more time when the sizes of values are increased, as reported in [14]. Existing key-value store based distributed file systems, e.g., IndexFS, store the file metadata of one file all in a single value. The whole value in the key-value store has to be reset when only a part of the value is modified, and this leads to unnecessary (de)serialization overhead and thereby poorer performance.

To mitigate this problem, IndexFS builds a cache to reduce performance loss during (de)serialization. However, there are two cache layers in those systems. One layer reduces the large value overhead in file system itself. Another layer is the in-memory LSM-Tree (Log-Structured Merge Tree) cache to reduce the small write overhead in LevelDB. An extra cache layer increases the copy time in memory and causes complex consistency issues. Our takeaway is that file system metadata needs optimizations in the organization to fully leverage the performance of KV stores.

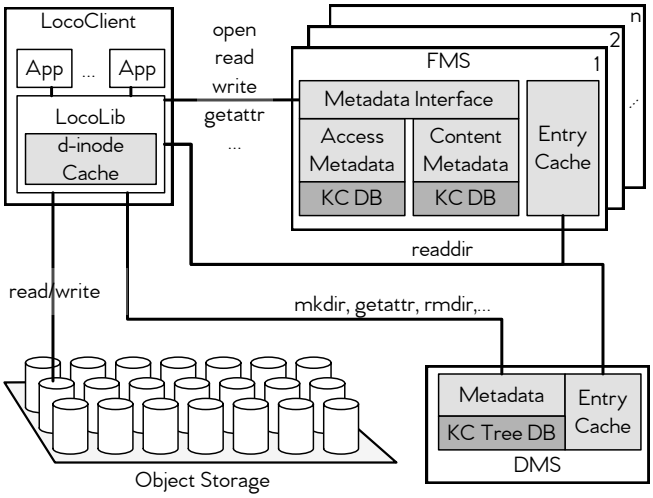

Figure 3: LocoFS's Architecture

In this paper, our goal is to reduce the dependencies. We propose a loosely-coupled metadata service, LocoFS, to provide a high-performance metadata service for distributed systems.

3 DESIGN AND IMPLEMENTATION

LocoFS is designed to scale its metadata performance to better leverage the performance benefits of key-value (KV) stores. To achieve this goal, LocoFS exploits three novel approaches to weaken its metadata dependencies.

- *Loosely-Coupled Architecture* that separates the directory and file metadata services to reduce the traversal path and latency time.
- *Flattened Directory Tree* that removes the directory inode (`d-inode`) dependencies and organizes metadata objects in a flat space to reduce the consistency between different nodes and improve throughput.
- *Decoupled File Metadata* that stores different types of file metadata differently to improve the KV access performance.

This section also discusses optimizations of the rename operation in LocoFS.

3.1 Loosely-Coupled Architecture

Figure 3 shows the architecture of LocoFS file system. LocoFS consists of four parts: *LocoClient*, *DMS (Directory Metadata Server)*, *FMS (File Metadata Server)*, and *Object Store*. LocoFS shares similar designs of Ceph to organize file data into objects using *Object Store*, while takes different approaches in the metadata service. For the metadata service, LocoFS manages directory and file metadata separately using a single *DMS* server and multiple *FMS* servers:

Single Directory Metadata Server. In LocoFS, the DMS server stores the directory metadata. Currently, we use only one single DMS in our design. The benefits of using one single DMS server rely on two facts. One reason is that a single server can support accesses to a large number of directories (e.g., around 10^8 directories for a DMS with 32GB memory) with the flattened directory tree design, which will be discussed in Section 3.2. The other reason is that one

single DMS can simplify the ACL (Access Control List) checking. Since file or directory accesses need to check the ACL capacity of its ancestors, the checking can be performed on a single server with one single network request from a client, rather than multiple network requests for distributed directory metadata service.

The DMS server organizes the directory metadata into key-value pairs. It uses the full path name as the key, and stores the metadata in the value. LocoFS integrates an existing key-value store implementation, Kyoto Cabinet, as the key-value store for the metadata.

Multiple File Metadata Servers. LocoFS uses multiple File Metadata Servers (FMS) to store file metadata. File metadata are distributed to different FMS servers using consistent hash in which the (`directory_uuid + file_name`) combination is used as the key. The `directory_uuid` is a universally unique identifier of the file's parent directory. In an FMS server, it also organizes file metadata as key-value pairs and stores them to the Kyoto Cabinet key-value store. The key used in the key-value store is the same key in the consistent hash.

A file system operation starts from the LocoClient. LocoClient provides either a FUSE interface or a LocoLib interface for applications. Through FUSE interface provides POSIX interface transparently like local file systems, FUSE's performance overhead is not negligible in a high-performance distributed file system [45]. Therefore, LocoLib is the default interface, though it requires application recompilation using new file system interface. From the LocoClient, LocoFS sends directory operations (e.g., `mkdir`, `rmdir` and `opendir`) to the DMS server, and sends file operations (e.g., `open`, `close`) to the FMS servers. After getting the file metadata from the FMS servers, LocoClient directly sends data operations (e.g., `read`, `write`) to the Object Store.

3.2 Flattened Directory Tree

Key-value stores better exploit the performance of storage devices than file systems do. This is because data are independently stored and accessed in key-value stores. Although some file systems gain such benefits by storing metadata in a key-value style, the dependencies between directory tree metadata still prevent file systems from effectively exploiting the key-value advantages. To tackle the issue, LocoFS proposes strategies to weaken the dependencies between directory tree metadata and organize them in a flat space. We call it *Flattened Directory Tree*.

3.2.1 Backward Directory Entry Organization. Figure 4 illustrates the flattened directory tree structure. This structure removes the `dirent-inode` links, which are used to locate the inodes of its files or subdirectories in a directory. Each directory keeps the directory entries (`dirents`), including the fields of the name and inode number of files or subdirectories in current directory, as directory data blocks. LocoFS does not keep directory entries in the directory data blocks, but instead reorganizes the `dirent` structure in a *backward* way. In Figure 4, the top half is a traditional directory tree structure. The directory metadata consists of two parts: the `d-inode` and the `dirent`. The file metadata only has one part, i.e., the file inode (`f-inode`). Directory and file metadata are connected using `dirent-inode` links, which are stored in directory entries in the directory data blocks. Both sides of a link are required to be updated atomically. This dependency incurs high metadata access

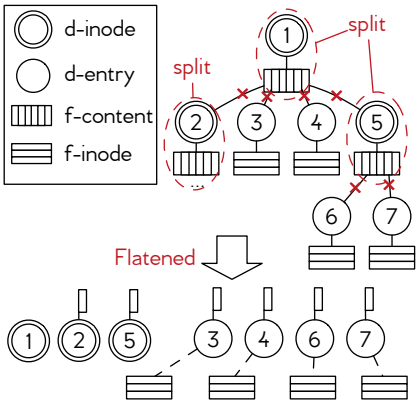

Figure 4: Flattened Directory Tree

overhead, especially when metadata objects are distributed to multiple servers. In the flattened directory tree, the `dirent-inode` is broken up, similar to the inverted index in ReconFS [28]. The directory entry (`dirent`) structure is reorganized. Rather than stored with the directory metadata, the `dirents` are respectively stored with the inodes of files or subdirectories. Specifically, each subdirectory or file in a directory stores its `dirent` with its `inode`. As shown in the bottom half of Figure 4, the main body of a directory or file metadata object is its `inode`. Each `inode` keeps the corresponding `dirent` alongside. As a consequence, these metadata objects are independently stored in a flat space.

In the flattened directory tree, directory or file metadata objects are then mapped to different metadata servers. All the directory inode (`d-inode`) metadata objects are stored in the DMS. Since these `d-inodes` are independent objects without tree links, they are more friendly to be stored in a key-value way. Each access to a directory inode can be performed by a key-value get operation by hashing its full path name. File inode(`f-inode`) objects are distributed to different file metadata servers using consistent hash. These `f-inode` objects are independent to not only other `f-inode` objects but also `d-inode` objects. Thus, they can be efficiently mapped to multiple servers, without incurring high-overhead inter-server requests.

In addition to the `inode` objects, the `dirent` objects are stored differently in LocoFS than in existing file systems. In a directory, all `dirents` of subdirectories are mapped to the directory metadata server, and the `dirent` of a file is mapped to the file metadata server that the file locates. In each metadata server, the `dirent` is also stored in a key-value store by using the hash value of `directory_uuid` as the key. In the directory metadata server, all the subdirectories in a directory have their `dirents` concatenated as one value, which is indexed by the `directory_uuid` key. Similarly, all the files that are mapped to the same file metadata server have their `dirents` concatenated and indexed. The directory entries that used to be data content of a directory are divided and co-located with its subdirectories or files. This flattened directory tree design weakens the dependencies of directory tree metadata objects, and thus improves the performance of the distributed metadata service.

3.2.2 Client Directory Metadata Cache. In LocoFS, each file operation (e.g., `create, remove and open`) has to check its parent directory from DMS. Though LocoFS can get 120K IOPS in one

node for `create` and `mkdir` without client cache, the single DMS design might limit the scalability.

To reduce the access latency and improve scalability, LocoFS uses client cache to keep directory metadata in clients. When a client creates a file in a directory, LocoFS caches the directory's `inode` from the DMS to client. When the client creates a file in the same directory, accesses to its directory metadata can be met locally. For the path traversal operations, LocoFS caches all the directory `inodes` along the directory path. This also offloads the DMS's traffic. The client directory metadata cache favors low latency and good scalability of LocoFS, even with a single DMS. Since it is common that HPC applications (e.g., earth simulation, weather forecast) nowadays store files in a specific set of directories, the directory metadata client cache are effective in most of these applications that have good directory locality.

The client directory metadata cache in LocoFS only caches directory inode (`d-inode`) metadata. Different from existing client cache designs, the LocoFS client cache does not cache file inode (`f-inode`) or directory entry (`dirent`) metadata. Similar to existing designs like NFS v4, LocoFS uses the *leases* mechanism for client cache, which grants a period (30s by default in LocoFS) to cached metadata for the valid status. LocoFS allocates 256 bytes for a `d-inode`. A client server accesses a limit number of directories. Since only `d-inodes` are cached in the client directory metadata cache, it consumes limited memory of a client server.

Table 1: Metadata Access in Different File Operations

		Dir	File		Dirent
Key		full path	uuid+filename		uuid
			Access	**Content**	
Value		ctime mode uid gid uuid	ctime mode uid gid	mtime atime size bsize suuid sid	entry
Operations	mkdir	●			●
	rmdir	●			●
	readdir	●			●
	getattr	●	●	●	
	remove		●	●	●
	chmod	●	●		
	chown	●	●		
	create		●		●
	open		●	❍	
	read			●	
	write			●	
	truncate			●	

● stands for field updating in an operation.
❍ stands for optional field updating in an operation (different file system have different implementations).

3.3 Decoupled File Metadata

While the flattened directory tree is friendly to key-value stores for weakening the metadata dependencies in the tree structure, LocoFS further decouples the file metadata to make it more friendly to key-value accesses. There are two major overheads that prevent file metadata from efficiently accessing key-value stores:

Large-Value Access Overhead. In key-value stores, we observe that performance drops drastically when the value size is large, which is along with the observation in WiscKey [25]. A file metadata object consumes hundreds of bytes, which is a relatively large size for key-value stores. Usually, an update to the file metadata only involves a few fields which only consumes several bytes [28], while this update requires the whole value to be read and updated. Therefore, storing file metadata in large values incurs unnecessary overhead in key-value stores.

(De)Serialization Overhead. When reading or writing file metadata, the values have to be deserialized and serialized between the memory and disk. The serialization and deserialization can also hurt key-value access efficiency even with protobuf [10] which in addition introduces extra metadata to manage the cached values.

To reduce the above-mentioned overheads, we design *Decoupled File Metadata* for LocoFS. It reduces the value size of file metadata using both *fine-grained file metadata* and *indexing metadata removal* techniques. It also removes the serialization and de-serialization by using fixed-length fields.

3.3.1 Fine-grained File Metadata. We design the fine-grained file metadata scheme that relies on the fact of the small-size value can be accessed more efficiently in the key-value stores. Specifically, LocoFS splits the file metadata into different parts. Each part is stored as one value, and different file operations access different parts of the file inode metadata. LocoFS splits file metadata into two parts: *the access metadata part* and *the content metadata part*. The access metadata part contains the fields `atime`, `mode`, `uid` and `gid`. These fields are used to describe the files access right. The content metadata part contains the fields `mtime`, `atime`, `block size` and `file size`. These fields are descriptions of the file content update.

Table 1 lists all operations in LocoFS and their accessed parts of the file metadata. We can see that most operations (e.g., `create, chmod, read, write and truncate`) access only one part, except for few operations like `getattr, remove, rename`. For instance, the `chmod` operation only reads the `mode`, `uid` and `gid` fields to check the access control list (ACL), and then updates the `mode` and `ctime` fields, accessing only the access part; The `write` operation only updates the `size` and `mtime` fields, both of which fall in the content part. By managing these fields in a manner of fine-grained, the value sizes in the FMS could be effectively reduced, which further improves the key-value access performance.

3.3.2 Indexing Metadata Removal. To further reduce the file metadata size, LocoFS indexes data blocks using `uuid + blk_num`, and thus removing the data indexing metadata. The `uuid` is a universally unique identifier for a file. It is composed of `sid` and `fid`, which respectively represents server ID and file ID in the server it first locates. A file can be identified in the distributed file system based on `uuid`. In addition, the `uuid + blk_num` can be used to identify a data block. When accessing a data block, LocoFS calculates the `blk_num` by dividing the file offset using block size, and locates the data block using both the `uuid` and `blk_num`. By indexing data blocks this way, the indexing metadata is removed from file metadata. This reduces the value size of file metadata, and further improves key-value access performance.

3.3.3 (De)Serialization Removal. LocoFS removes the serialization and deserialization steps to further improve key-value access performance. File metadata of a file contains multiple fields, though they are stored in a single value. File systems always access different fields. Therefore, these fields are deserialized from the value strings to the memory hierarchy (i.e., the `struct` in the program). Updates are performed on different fields in the memory structure. When they are required to be written to secondary storage, they are serialized to a value string. One reason of (de)serialization is that there are varied-length fields, which requires parsing using (de)serialization. Since LocoFS removes the varied-length part, i.e., the indexing metadata, the left parts of the file metadata contain only fixed-length fields. With all fields fixed-length, LocoFS can directly locate a field through a simple calculation. As such, LocoFS directly accesses fields in the value string without serialization or deserialization. With the removal of this layer, key-value access performance is improved for file metadata accesses.

3.4 Rename Discussion

LocoFS distributes metadata objects based on hash algorithms both within or across metadata servers. The hash-based metadata distribution effectively exploits the key-value access performance. However, the `rename` operation raises performance concern to such hash-based design. In this part, we first statistically analyze the percentage of rename operations in real applications, and then propose two optimization techniques to further mitigate this overhead.

3.4.1 Low Percentage of Rename Operations in Real Applications. To estimate the `rename` cost, we first analyze the percentage of rename operations out of the total file system operations. In file systems, there are two types of `rename` operations: file rename (`f-rename`) and directory rename (`d-rename`). We analyze the file system trace [50] on Sunway TaihuLight Supercomputer [1], and find there is no `d-rename` and `f-rename` operations. Our observation is also consistent with previous research from GPFS file system of Barcelona Supercomputing Center [23], where the `d-rename` consumes only 10^{-7} percentage of the total operations. Even a file system has extremely low percentage of rename operations, LocoFS provides two optimizations to accelerate rename performance as follows.

3.4.2 Relocation Reduction using UUID. The first optimization is to use UUID to reduce the relocations caused by rename operations. Different from traditional hash-based distribution, LocoFS assigns each directory or file with a UUID. LocoFS uses UUID as part of indices to index the subdirectories, files or data blocks. Since the UUID does not change the name, the successors that are indexed using UUID mostly do not need to be relocated.

Rename operations contains both the `f-rename` and the `d-rename`. For `f-rename`, only the file metadata object needs to be relocated, due to its changed index `directory_uuid + file_name`. Because all data blocks are indexed using `uuid + blk_num`, in which the file's uuid does not change, the data blocks do not need to be relocated. For `d-rename`, the directory itself and its succeed directories need to have their metadata objects relocated. The succeed files do not need to be relocated because they are indexed using `directory_uuid + file_name`. In conclusion, only succeed directories have to relocate their metadata objects. Since they are stored in a single DMS server, all these operations do not involve cross-server operations, thus limiting the performance degradation.

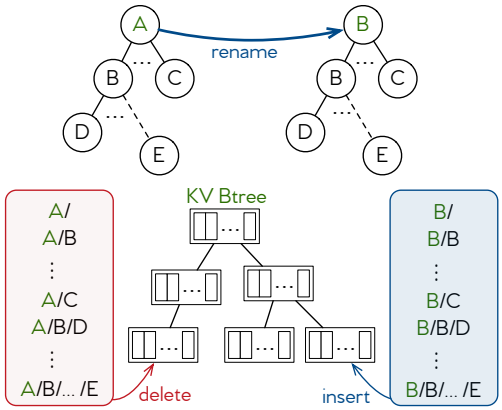

Figure 5: Rename Acceleration using B+ Tree

3.4.3 Rename Acceleration using B+ Tree. The second optimization is to leverage B+ tree structure to accelerate the relocations of sub-directories in a directory rename operation, which is the only major cost as discussed above. LocoFS leverages the B+ tree structure in the Kyoto Cabinet key-value store to manage the directory metadata in the DMS server. In the B+ tree structure, the keys are organized in alphabetical order. In the directory metadata store, the full path name is used as the key. As a consequence, all sub-directories of a directory are placed together in the B+ tree, as shown in Figure 5. With this property, these sub-directories can be moved to a new place following the new name of the directory with low cost.

4 EVALUATION

In this section, we compare LocoFS with three widely used distributed file systems - Lustre, CephFS and Gluster. We first evaluate the metadata performance (Section 4.2), including the latency, scalability, performance gap to raw key-value stores, and effects respectively from *flattened directory tree* and *decoupled file metadata* techniques. We then evaluate the full system performance by performing `read` and `write` operations (Section 4.3). In addition, we evaluate the `d-rename` overhead (Section 4.4.2) in LocoFS and its sensitivity to the directory depth (Section 4.4.1).

4.1 Experimental Setup

4.1.1 Cluster Hardware Configuration. We perform the evaluations on two clusters, the Dell PowerEdge cluster and the SuperMicro cluster, as shown in Table 2. The Dell PowerEdge cluster consists of 16 nodes. Each node is equipped with 16GB DDR2 memory and 8 CPU cores. The SuperMicro cluster consists of 6 nodes. Each node is equipped with 384GB DDR4 memory and 24 CPU cores. Servers in both of the two clusters are connected with 1Gbps Ethernet. We use the wimpy Dell PowerEdge cluster for the metadata service deployment, and the beefy SuperMicro cluster for clients, to generate

Table 2: The Experimental Environment

Cluster Name	Metadata Cluster	Client Cluster
Server Name	DELL PowerEdge	SuperMicro
#Machines	16	6
OS	CentOS 7	CentOS 7
CPU	AMD Opteron 8 core 2.5GHz	Intel Xeon 24 core 2.5GHz
Memory	DDR2 16G	DDR4 384G
Storage	SAS 128G	SATA 1TB
Network	1GE	1GE
Evaluated FS	CephFS 0.94, Gluster 3.43 Lustre 2.9, LocoFS	

enough client requests to stress the metadata service. For the SuperMicro cluster, each core runs two clients using hyper-threading, with a total of 288 clients in the whole cluster.

4.1.2 Software Configuration. We use Lustre at version 2.9, Ceph at version 0.94 and Gluster at version 3.7.8. The Lustre binary is obtained from the *Lustre-release* [6] source, and the CephFS and Gluster binaries are obtained from the *Epel-Release* [2] source. We deploy these file systems on CentOS 7. All the servers use btrfs as local file systems except that Lustre runs on a loop device. Lustre is configured with DNE1, which divides the MDS manually, and DNE2, which stripes the directories automatically.[1]

We use *mdtest* [7] benchmark to evaluate the metadata performance of above-mentioned file systems. The clients have OpenMPI configured at version 1.10. In the evaluations, we abandon the FUSE interface to avoid the performance bottleneck in the clients. Instead, we modify the `mdtest` benchmark which uses the client libraries to directly communicate with metadata servers, e.g., `libgfapi` [5] in Gluster, `libcephfs` [4] in CephFS, and `locolib` in LocoFS.

4.2 Metadata Performance

4.2.1 Latency. We first evaluate the latency benefits obtained through our design of flattened-directory tree. In this evaluation, we use a single client to perform `touch`, `mkdir`, `rmdir`, `stat` and `rm` operations. The *mdtest* benchmark is used for those workloads. It generates 1 million files and directories in every test. Since all these operations are synchronous operations, we collect the latency for each operation in the client. In addition, we also use a single client to perform `readdir` operations. This workload is performed by reading a directory with 10k files and sub-directories.

Figure 6 shows the normalized latencies of `touch` and `mkdir` in each file system. The normalized latency is the metadata operation latency that is normalized to the latency of one round trip. For brevity, we show the normalized latencies of other operations only with 16 metadata servers in Figure 7. From the two figures, we make three observations.

(1) LocoFS achieves the lowest latencies for `touch` and `mkdir`. Compared with LocoFS-C, the average latency of Lustre D1, Lustre D2, CephFS, Gluster respectively are 4×, 6×, 8× and 8× of LocoFS for the `touch` operation, and 4×, 4×, 6× and 26× of LocoFS for the `mkdir` operation. In particular, LocoFS-C and LocoFS-NC achieves an average latency of 1.1× RTT time for creating a directory using the `mkdir` operations. This latency is very close to the round-trip latency of Ethernet. This demonstrates that the single Directory Metadata Server (DMS) design can provide low-latency directory access operations. LocoFS-C and LocoFS-NC also achieve much lower latencies than Lustre, CephFS and Gluster for creating files using the `touch` operations. This is because that flattened directory tree provides direct file locating service, which enables low-latency file locating. Gluster gets the highest latency in `mkdir` due to its directory synchronization operation in every node.

(2) Compared with Gluster, LocoFS shows relatively stable latency with the increase of metadata servers. Latencies in Gluster is dramatically increased when the number of metadata servers increased. In contrast, latencies in LocoFS show the much lower increase, which indicates better stable performance. This performance stability in LocoFS is contributed mostly by the flattened directory tree structure, which reduces the cross-server operations in the metadata server cluster. An interesting observation occurs to `touch` operations in LocoFS. The average latency is increased from 1.3× to 3.2× of the RTT time in LocoFS-C and from 3.1× to

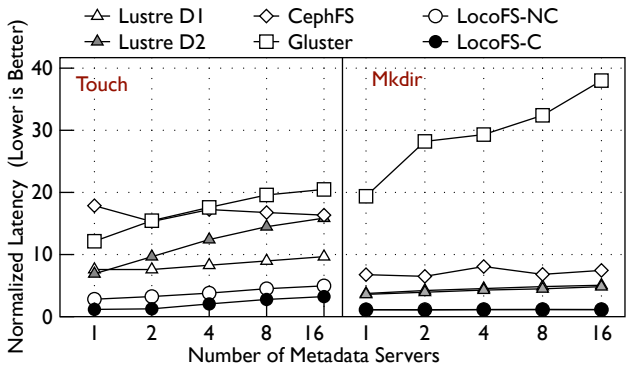

Figure 6: Latency Comparison for `touch` and `mkdir` operations. Y-axis is the latency normalized to the single RTT (Round-Trip Time, 0.174ms).

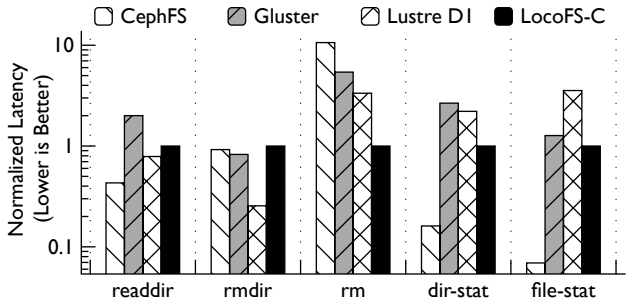

Figure 7: Latency Comparison for `readdir`, `rmdir`, `rm`, `dir-stat` and `file-stat` operations with 16 Metadata Servers. Y-axis is the latency normalized to the LocoFS-C.

[1] In our graph legend, *LocoFS-C* stands for LocoFS with cache in client, *LocoFS-NC* stands for LocoFS without cache in client, *Lustre D1* stands for DNE1, and *Lustre D2* stands for DNE2 [19].

6.2× of the RTT time in LocoFS-NC. However, this is not a design flaw of LocoFS. It is because the client has to set up more network connections to different metadata servers. More network connections slow down the client performance. CephFS and Lustre also show the similar pattern with LocoFS for the `touch` operations.

(3) LocoFS achieves comparable latencies with the other evaluated file systems for other operations, as shown in figure 7. LocoFS provides similar `readdir` and `rmdir` performance to Lustre and Gluster. This is because LocoFS cannot identify whether the directory has files in FASs (File Metadata Servers). Thus, LocoFS needs to check every node to confirm that files have already been deleted for the `rmdir` operation, and pull the `d-entry` metadata to the client for the `readdir` operation. The overhead of `readdir` also occurs in other distributed file systems that distribute files of one directory to different metadata servers, such as in Gluster and Lustre. LocoFS gets the lower latencies in `rm`, `dir-stat` and `file-stat` operations compared with Lustre and Gluster. It is because LocoFS avoids the path traverse operation among different metadata servers. CephFS gets the lowest latency for `dir-stat` and `file-stat` due to the metadata cache in the client.

4.2.2 Throughput. Since the reduced dependencies in file system metadata of LocoFS contribute to scalability, we measure the throughputs of LocoFS, Lustre, CephFS and Gluster for comparison. In this evaluation, we deploy OpenMPI to run the *mdtest* benchmark. We scale the metadata servers from 1 to 16 to evaluate the throughput of metadata service.

To saturate the metadata service capacity, we try to generate enough pressure from the clients. In the experiments, we find that the performance of metadata service drops when the number of clients exceeds a certain value, due to client contentions. Therefore, we use the optimal number of clients that achieve the best metadata performance for each experiment. To identify the optimal numbers, we start from 10 clients while adding 10 clients every round until the performance reaches the highest point. Each client creates 0.1 million directories and files. Table 3 shows the optimal number of clients for different settings.

Table 3: The Number of Clients in Each Test

	1	2	4	8	16
LocoFS with no cache	30	50	70	120	144
LocoFS with cache	30	50	70	130	144
CephFS	20	30	50	70	110
Gluster	20	30	50	70	110
Lustre with DNE 1	40	60	90	120	192
Lustre with DNE 2	40	60	90	120	192

Figure 8 shows the throughput (IOPS) of evaluated file systems by varying the number of metadata servers from 1 to 16. It gives the results of `touch`, `mkdir`, `stat`, `rmdir` and `rm` operations. From the figure, we make following observations:

(1) The IOPS of LocoFS for file or directory create (i.e., `mkdir`) operations with one metadata server is 100K, which is 67×2012*leveldb* higher than CephFS, 23× than Gluster and 8× than Lustre DNE1 and Lustre DNE2. The IOPS of LocoFS gets close to the random

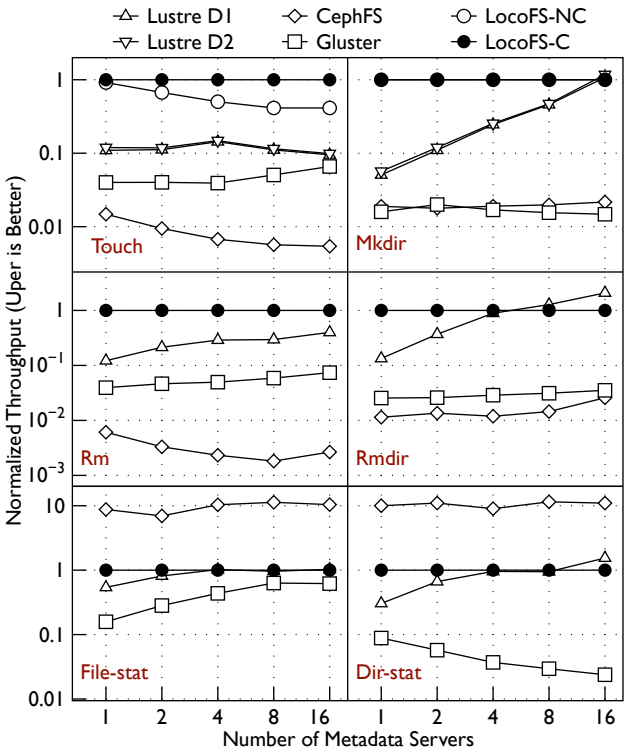

Figure 8: Throughput Comparison of touch, mkdir, rm, rmdir, file-stat and dir-stat. Y-axis is the throughput normalized to that of LocoFS-C. Results of Lustre D2 and LocoFS-NC are omitted in rm, rmdir, file-stat and dir-stat evaluations, because Lustre D2 and LocoFS-NC respectively share similar performance with Lustre-D1 and LocoFS-C.

write performance of key-value (KV) stores, e.g., LevelDB at 164K IOPS [16]. It demonstrates that the loosely-coupled structure is suitable for the key-value stores and is effective in bridging the gap between key-value stores and file system metadata performance.

(2) LocoFS-C shows file create (i.e., `touch`) performance that is much higher than Lustre, CephFS and Gluster. LocoFS-C also shows good scalability when the number of metadata servers is increased from 1 to 16. While flattened directory tree structure enables good scalability of file accesses, the directory cache in the client further removes the directory metadata access bottlenecks. The `touch` figure shows that LocoFS-C achieves 2.8× compared with LocoFS-NC with sixteen servers. It means that with the client cache of directory metadata, LocoFS achieves good scalability.

(3) LocoFS outperforms the other file systems for file create and remove (i.e., `touch` and `rm`) operations, and outperforms CephFS and GlusterFS for directory create and remove (i.e., `mkdir` and `rmdir`) operations in all evaluated cases. One exception is that LocoFS has poorer scalability than Lustre for `mkdir` and `rmdir` operations. The scalability of `mkdir` in Lustre comes from the parallelism of operations to the increased number of metadata servers, while LocoFS has only one DMS (Directory Metadata Server). The

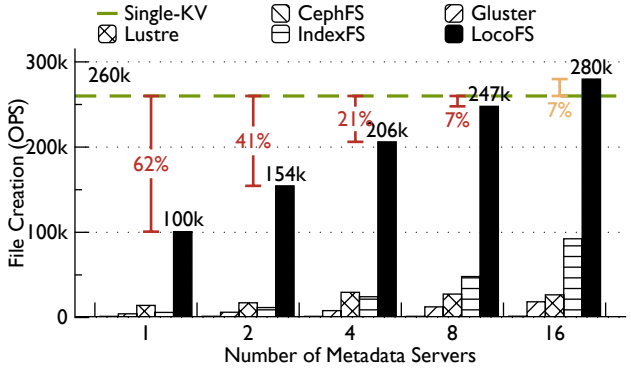

Figure 9: Bridging the Performance Gap Between File System Metadata and Raw Key-value Store.

poorer scalability of `rmdir` in LocoFS is because LocoFS have to get the directory entries from all metadata servers.

(4) LocoFS outperforms Lustre and Gluster, but has poorer performance than Ceph for `dir-stat` and `file-stat` operations. This is because LocoFS only caches d-inode, but Ceph caches both d-inode and f-inode. Also, LocoFS uses a strict lease mechanism (as in Section 3.2.2) that leads to high miss ratios in d-inode cache.

4.2.3 The Bridging Performance Gap. Figure 9 compares file systems' metadata performance with raw key-value store's performance. From the figure, we can see that LocoFS can achieve 38% the performance of Kyoto Cabinet (Tree DB) using one metadata server. It has significantly higher performance than the other file systems.

In addition, LocoFS using 16 metadata servers can achieve the single-node Kyoto Cabinet performance. LocoFS has a peak performance of 280k IOPS when using 16 metadata servers, which is 2.5× higher than IndexFS with the same number of metadata servers which is reported in [38]. This also tells that LocoFS is more effective in exploiting the benefits of key-value stores than existing distributed file systems. While data sources are the same with those in Figure 1, we conclude the loosely-coupled metadata service in LocoFS is effectively bridging the gap between file system metadata service and key-value stores.

4.2.4 Effects of Flattened Directory Tree. We also measure the throughputs of LocoFS, as well as Lustre, CephFS, Gluster and IndexFS, by co-locating the client with its metadata server, which does not involve network latencies. By mitigating the network impact, we focus on the effects of the flattened directory tree (that has been described in section 3.2) on the metadata performance. IndexFS code is fetched from the Github[2]. Evaluation methods are as same as configured in section 4.2.1. In this evaluation, we use one client to run the *mdtest* benchmark.

Figure 10 shows the latency results of evaluated file systems by co-locating clients with its metadata servers. From Figure 10, we can see that LocoFS achieves the lowest latency among the evaluated distributed file systems for `mkdir`, `rmdir`, `touch` and `rm` operations. Compared to CephFS and Gluster, IndexFS achieves lower latency.

[2]https://github.com/zhengqmark/indexfs_old

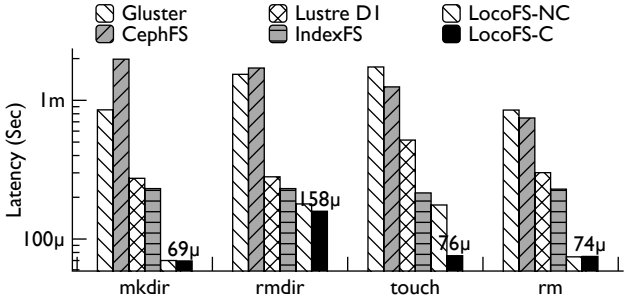

Figure 10: Effects of Flattened Directory Tree. All the tests are running on a single server.

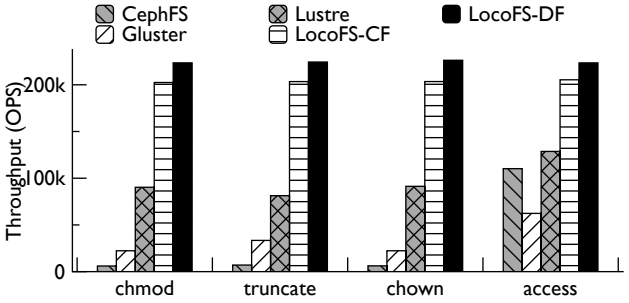

Figure 11: Effects of Decoupled File Metadata. LocoFS-DF stands for LocoFS with decoupled file metadata. LocoFS-CF stands for LocoFS with coupled file metadata. Those experiments are executed with 16 metadata servers.

Storing file system metadata using key-value stores contributes to the performance benefits. LocoFS has even lower latency than IndexFS, because LocoFS further optimizes its logical organization in a key-value friendly way.

By comparing Figure 6 and Figure 10, we observe that LocoFS can achieve better performance with faster network. In Figure 6, which shows the latency with network round-trips, LocoFS achieves latency that is 1/6 of that in CephFS or Gluster. But in Figure 10, LocoFS's latency is only 1/27 of CephFS and 1/25 of Gluster. The latency attributes to two factors: network latency and software overhead. From the two figures, we also find that faster network devices do little help in improving the metadata service of CephFS and Gluster, whose bottleneck still lies in the metadata processing and storage parts (i.e., the software part). In contrast, since LocoFS has lower software overhead, faster network devices can further improve its performance. It means that loosely-coupled metadata design also reduces the software latency, which may also contribute to better performance when using a high-speed network.

4.2.5 Effects of Decoupled File Metadata. In this section, we evaluate the performance improvement led by the decoupled file metadata technique that has been described in section 3.3. We perform the experiments with 16 metadata servers using a modified *mdtest* benchmark, which adds `chmod`, `truncate`, `chown` and `access` operations. Those operations change either the *access region* or *content region* that are defined in Table 1.

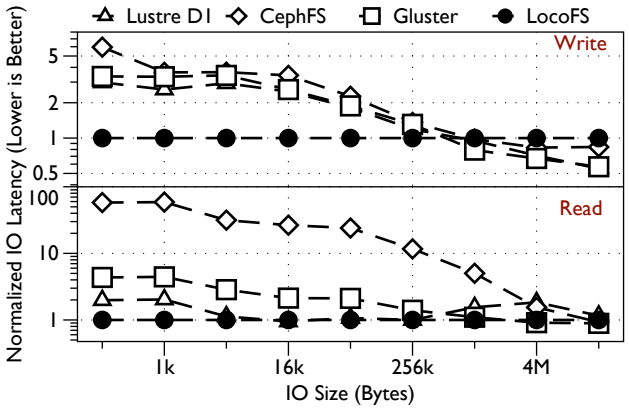

Figure 12: The Write and Read Performance. Y-axis is the latency normalized to LocoFS.

Figure 11 shows the IOPS of the above-mentioned metadata operations for evaluated file systems. LocoFS is configured with the decoupled file metadata technique disabled and enabled, which are respectively annotated with LocoFS-CF and LocoFS-DF. While LocoFS-CF has already improved these file metadata performance, LocoFS-DF further improves performance over LocoFS-CF. With the key-value friendly organization, LocoFS achieves relatively good performance even without the file metadata cache layer. This improvement is also contributed by the reduced serialization and de-serialization of key-value stores when performing write and read operations.

4.3 Full System Performance

In this section, we measure the full system performance to evaluate the performance impact from the metadata service in LocoFS. In this experiment, we use a combination of create, read/write and close operations as the workload. Lustre, CephFS, Gluster and LocoFS are configured with 16 metadata servers. All these file systems do not use data replicas in the experiments. In each file system, 1000 files in one directory are created and read/written using fixed-size I/Os.

Figure 12 shows both write and read latencies for the three evaluated file systems using different I/O sizes. For the 512B I/O evaluation, LocoFS's write latency is only 1/2 of Lustre, 1/4 of Gluster and 1/5 of CephFS. LocoFS's read latency is 1/3 of Gluster and Lustre and 1/50 of CephFS. This is because, with small I/O sizes, the metadata performance is important to the whole system performance. This benefit in LocoFS lasts before the write size exceeds 1MB or the read size exceeds 256KB. With large size I/Os, data performance is more dominated by the data I/O performance than the metadata performance. As such, LocoFS is more effective in improve I/O performance with small access sizes, due to its significant improvement in the metadata performance.

4.4 Overhead Evaluation

4.4.1 Impact from Directory Depth. In this section, we evaluate the impact on performance from the directory depth. We vary the depth of directories from 1 to 32 in the *mdtest* benchmark. In the evaluation, we configure LocoFS with both cache enabled and disable, and respectively evaluate them using two and four metadata servers, resulting four configurations in total.

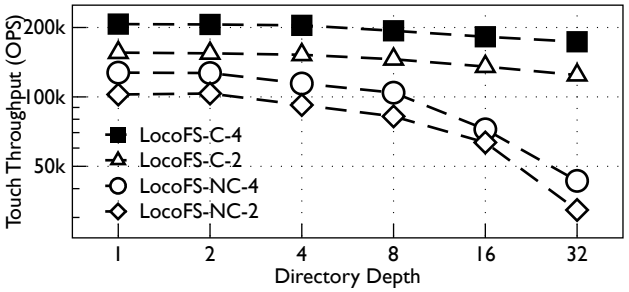

Figure 13: Sensitivity to the Directory Depth. LocoFS-C and LocoFS-NC respectively stand for LocoFS with cache enabled and disabled. 2 and 4 stand for the number of metadata servers.

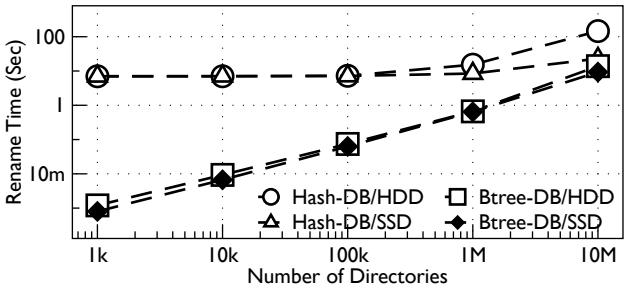

Figure 14: Rename Overhead.

Figure 13 shows the IOPS of file `create` operations of four different configurations with varied depths of directories. From the figure, we can see that the performance of LocoFS-NC (with cache disabled) drops dramatically, e.g., from 120K to 50K when using four metadata servers. This is because directory tree has to check the access control list in each level. Deeper directories lead to higher latency in path traversal. In contrast, LocoFS-C (with cache enabled) has less performance loss with increased depth (e.g., from 220K to 125K) when using four metadata servers. The client cache mitigates this problem.

4.4.2 Directory Rename Overhead. In this section, we evaluate the impact on directory rename (`d-rename`) operations. Since (`d-rename`) is a major concern in hash-based metadata designs, we measure the rename overhead by varying the number of renamed directories from 1000 to 10 millions in LocoFS. To simulate the real environment, in this experiments, we first create 10 millions directories in DMS. We also compare the hash DB and Btree DB modes in Tokyo Cabinet, in which we leverage the tree DB mode for rename optimization (as in Section 3.4.3). The experiments are also performed respectively in SSDs and HDDs.

Figure 14 shows time that is used in the `d-rename` operations for different number of renamed directories with four modes. From the figure, we can see that the Btree mode, which is used with rename optimizations, can significantly reduce the rename time, compared to the hash mode. Compared with the hash-based key-value store that needs to traverse all the records, Btree-based ones can perform

fast rename operations, and is suitable for directory management (i.e., the directory metadata server) in LocoFS. In the B-tree mode, it can perform rename operations within a few seconds for renaming 1 million directories. There is no big difference between HDDs and SSDs for the rename operations. Even with the hash mode and extremely large number of rename operations (e.g., 10 million), the rename operation can also be completed in around 100 seconds, which is acceptable. In conclusion, even though directory rename operations are common in some cases, LocoFS can efficiently limit its overhead.

5 RELATED WORK

Efficiency of the file system software has recently been a hot topic, especially when used on the high-performance flash SSDs [21, 22, 28, 29, 55] or persistent memory [12, 15, 26, 35, 49, 54]. Different from the above-mentioned works that aim to reduce software overhead leveraging emerging storage hardware features, LocoFS improves software efficiency by decoupling the metadata dependencies to exploit the key-value store potentials. In this section, we focus on the metadata organization in both local and distributed file systems.

Namespace Structure in Local File Systems: The directory tree data structure has been the dominate namespace structure in file systems for decades. It is argued that the directory tree could be a limitation in file systems, due to either the increasing data volume [37, 41] or the deployment of fast non-volatile memory [28].

With the increased data volume, a file system needs to manage trillions of files, in which finding a specific file is much difficult. For this reason, the hFAD (hierarchical File Systems Are Dead) project argues that semantic file systems, which accelerate file searching by extending the attributes, could be the future file systems [41].

A less aggressive approach to improve the namespace performance is to introduce the table-based organization to the directory tree. In the early years, Inversion File System [33] has chosen to organize file system metadata using Postgre database. Recently, researchers propose to organize metadata using key-value (KV) stores, which provides high performance due to simple organization. For instance, XtreemFS [8] uses BabuDB [44]; TableFS [37] and its distributed version IndexFS [38] use LevelDB [3]. Researchers even propose to built the whole file system on KV stores. HopsFS [32] uses NewSQL to store the metadata of HDFS and remove the metadata bottleneck by distribute the in-memory metadata of HDFS. KVFS [43] is built on the VT-Tree, a variance of LSM-Tree (Log-Structured Merge Tree) [34]. BetrFS [20, 53] is built on fractal tree, which is also an optimization version of LSM-Tree. LocoFS takes this approach by using the Kyoto Cabinet KV store [17].

Recent flash file systems have proposed to indexing data objects in a backward way to reduce write amplification and improve flash endurance [28, 29]. OFSS [29] proposes the backpointer-assisted lazy indexing to keeping the index pointers along with data objects, which enables lazy updating to the normal indexing. ReconFS [28] redesigns the namespace management of a local file system, and makes it reconstructable. Leveraging the unbalanced read and write performance of flash memory, it does not strictly maintain the directory tree connections in the persistent storage. Instead, it stores the inverted indices with each file or directory to reduce the write overhead while enables fast recovery leveraging the comparatively fast read performance of flash memory. The backward indexing designs reduces the metadata writes to flash memory and thereby extends flash lifetime. Inspired by the backward indexing designs in OFSS and ReconFS, LocoFS introduces the flatten directory tree design to distributed file systems to weaken the metadata dependencies.

Metadata Distribution in Distributed File Systems: The scalability of metadata service is the key design point in the scalability of distributed file systems. Most existing distributed file systems either distributed metadata to data servers [13, 26, 30, 47] or use metadata server cluster [11, 52]. Metadata distribution of these distributed file systems falls into two major categories: *hash-based* and *directory-based*.

The directory-based metadata distribution favors locality more than load balance. CephFS [47] distributes metadata in the units of sub-directories and balances the access dynamically. All files and sub-directories in one directory have their metadata located in the same server, which maintains data locality and reduces cross-server operations. Its successor, Mantle [42], further proposes a programmable interface for flexible metadata patterns.

The hash-based method favors load balance while sacrificing locality. Giga+ [36], as well as IndexFS [38] which is the successor of Giga+, uses hash and dynamic directory namespace to distribute metadata in different servers. Gluster [13] has no metadata server but layout the metadata in each data server, and uses extended attribute in local file system and DHT based on directory to distribute the metadata. LocoFS also takes the full-path hashing in metadata distribution, but reduces the dependency of metadata to further match the performance of key-value stores.

6 CONCLUSION

Key-value stores provide an efficient way for file system metadata storage. However, there is still a huge performance gap between file system metadata and key-values stores. It is the strong dependencies between metadata accesses that prevent file system metadata service from fully exploiting the potentials of key-value stores. In this paper, we propose a **loosely-coupled** metadata service, LocoFS, for distributed file systems. LocoFS decouples the `dirent-inode` dependencies to form flattened directory tree structure, and further decouples the file metadata into `access` and `content` regions, so as to organize file system metadata in a key-value friendly way. Evaluations show that LocoFS achieves 1/4 latency (when using one metadata server) and 8.5× IOPS (when using four metadata server) of IndexFS, a recent file system with metadata service optimizations using key-value stores, while providing scalable performance.

ACKNOWLEDGMENTS

We thank Wei Xue and Yibo Yang for sharing the file system trace on Sunway TaihuLight. This work is supported by the National Natural Science Foundation of China (Grant No. 61502266, 61433008, 61232003), the Beijing Municipal Science and Technology Commission of China (Grant No. D151100000815003), and the China Postdoctoral Science Foundation (Grant No. 2016T90094, 2015M580098). Youyou Lu is also supported by the Young Elite Scientists Sponsorship Program of China Association for Science and Technology (CAST).

REFERENCES

[1] China Tops Supercomputer Rankings with New 93-Petaflop Machine. https://www.top500.org/news/china-tops-supercomputer-rankings-with-new-93-petaflop-machine/.

[2] Extra packages for enterprise linux (epel). https://fedoraproject.org/wiki/EPEL.

[3] LevelDB, a fast and lightweight key/value database library by Google. https://code.google.com/p/leveldb/.

[4] libcephfs. https://github. com/ceph/ceph/blob/master/src/include/cephfs /libcephfs. h.

[5] libgfapi. http://gluster-documentation-trial. readthedocs. org/en/latest/Features/libgfapi/.

[6] Lustre releases. https://wiki.hpdd.intel.com.

[7] mdtest. https://github.com/MDTEST-LANL/mdtest.

[8] XtreemFS. http://www.xtreemfs.org/.

[9] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. 2012.

[10] P. C. Authors. protobuf: bindings for Google's Protocol Buffers–Google Project Hosting.

[11] P. J. Braam et al. The lustre storage architecture. 2004.

[12] Y. Chen, Y. Lu, J. Ou, and J. Shu. HiNFS: A persistent memory file system with both buffering and direct-access. In *ACM Transaction on Storage (TOS)*. ACM, 2017.

[13] A. Davies and A. Orsaria. Scale out with glusterfs. *Linux Journal*, 2013(235):1, 2013.

[14] J. Dean and S. Ghemawat. Leveldb. *Retrieved*, 1:12, 2012.

[15] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys)*, pages 15:1–15:15, New York, NY, USA, 2014. ACM.

[16] Google. Leveldb benchmarks. https://leveldb.googlecode.com, 2011.

[17] M. Hirabayashi. Kyoto cabinet: a straightforward implementation of dbm.

[18] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario. The xtreemfs architectureâĂŤa case for object-based file systems in grids. *Concurrency and computation: Practice and experience*, 20(17):2049–2060, 2008.

[19] Intel. Analysis of dne phase i and ii in the latest lustre* releases. http://www.intel.com/content/www/us/en/lustre/dne-phase-i-ii-latest-lustre-releases.html, 2016.

[20] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, M. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter. BetrFS: A right-optimized write-optimized file system. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 301–315. USENIX Association, 2015.

[21] W. K. Josephson, L. A. Bongo, D. Flynn, and K. Li. DFS: A file system for virtualized flash storage. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA, 2010. USENIX.

[22] C. Lee, D. Sim, J. Hwang, and S. Cho. F2FS: A new file system for flash storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*, Santa Clara, CA, Feb. 2015. USENIX.

[23] P. H. Lensing, T. Cortes, J. Hughes, and A. Brinkmann. File system scalability with highly decentralized metadata on independent storage devices. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 366–375. IEEE, 2016.

[24] A. W. Leung, S. Pasupathy, G. R. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *USENIX Annual Technical Conference*, volume 1, pages 5–2, 2008.

[25] L. Lu, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Wisckey: separating keys from values in ssd-conscious storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 133–148, 2016.

[26] Y. Lu, J. Shu, Y. Chen, and T. Li. Octopus: an RDMA-enabled distributed persistent memory file system. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 773–785. USENIX Association, 2017.

[27] Y. Lu, J. Shu, S. Li, and L. Yi. Accelerating distributed updates with asynchronous ordered writes in a parallel file system. In *2012 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 302–310. IEEE, 2012.

[28] Y. Lu, J. Shu, and W. Wang. ReconFS: A reconstructable file system on flash storage. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*, pages 75–88, 2014.

[29] Y. Lu, J. Shu, and W. Zheng. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA, 2013. USENIX.

[30] M. Moore, D. Bonnie, B. Ligon, M. Marshall, W. Ligon, N. Mills, E. Quarles, S. Sampson, S. Yang, and B. Wilson. Orangefs: Advancing pvfs. *FAST poster session*, 2011.

[31] D. Nagle, D. Serenyi, and A. Matthews. The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 53. IEEE Computer Society, 2004.

[32] S. Niazi, M. Ismail, S. Grohsschmiedt, M. Ronström, S. Haridi, and J. Dowling. HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases. 2016.

[33] M. A. Olson. The design and implementation of the inversion file system. In *USENIX Winter*, 1993.

[34] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.

[35] J. Ou, J. Shu, and Y. Lu. A high performance file system for non-volatile main memory. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 12. ACM, 2016.

[36] S. Patil and G. A. Gibson. Scale and concurrency of GIGA+: File system directories with millions of files. In *FAST*, volume 11, pages 13–13, 2011.

[37] K. Ren and G. A. Gibson. TableFS: Enhancing metadata efficiency in the local file system. In *Proceedings of 2013 USENIX Annual Technical Conference (USENIX'13)*, pages 145–156, 2013.

[38] K. Ren, Q. Zheng, S. Patil, and G. Gibson. IndexFS: scaling file system metadata performance with stateless caching and bulk insertion. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*, pages 237–248. IEEE, 2014.

[39] D. S. Roselli, J. R. Lorch, T. E. Anderson, et al. A comparison of file system workloads. In *USENIX annual technical conference, general track*, pages 41–54, 2000.

[40] R. B. Ross, R. Thakur, et al. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th annual Linux showcase and conference*, pages 391–430, 2000.

[41] M. I. Seltzer and N. Murphy. Hierarchical file systems are dead. In *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS XII)*, 2009.

[42] M. A. Sevilla, N. Watkins, C. Maltzahn, I. Nassi, S. A. Brandt, S. A. Weil, G. Farnum, and S. Fineberg. Mantle: a programmable metadata load balancer for the ceph file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 21. ACM, 2015.

[43] P. J. Shetty, R. P. Spillane, R. R. Malpani, B. Andrews, J. Seyster, and E. Zadok. Building workload-independent storage with vt-trees. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 17–30. USENIX, 2013.

[44] J. Stender, B. Kolbeck, M. Ho gqvist, and F. Hupfeld. Babudb: Fast and efficient file system metadata storage. In *International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, pages 51–58. IEEE, 2010.

[45] B. Vangoor, V. Tarasov, and E. Zadok. To FUSE or Not to FUSE: Performance of User-Space File Systems. *FAST'17: 15th USENIX Conference* , 2017.

[46] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI)*, pages 307–320. USENIX Association, 2006.

[47] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic metadata management for petabyte-scale file systems. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 4. IEEE Computer Society, 2004.

[48] J. Xiong, Y. Hu, G. Li, R. Tang, and Z. Fan. Metadata distribution and consistency techniques for large-scale cluster file systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):803–816, 2011.

[49] J. Xu and S. Swanson. Nova: a log-structured file system for hybrid volatile/non-volatile main memories. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 323–338, 2016.

[50] Y. Yang, X. Wang, B. Yang, W. Liu, and W. Xue. I/O trace tool for HPC applications over Sunway TaihuLight Supercomputer. In *HPC China*, 2016.

[51] L. Yi, J. Shu, J. Ou, and Y. Zhao. Cx: concurrent execution for the cross-server operations in a distributed file system. In *2012 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 99–107. IEEE, 2012.

[52] L. Yi, J. Shu, Y. Zhao, Y. Qian, Y. Lu, and W. Zheng. Design and implementation of an asymmetric block-based parallel file system. *IEEE Transactions on Computers*, 63(7):1723–1735, 2014.

[53] J. Yuan, Y. Zhan, W. Jannen, P. Pandey, A. Akshintala, K. Chandnani, P. Deo, Z. Kasheff, L. Walsh, M. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter. Optimizing every operation in a write-optimized file system. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 1–14. USENIX Association, 2016.

[54] K. Zeng, Y. Lu, H. Wan, and J. Shu. Efficient storage management for aged file systems on persistent memory. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1769–1774. IEEE, 2017.

[55] J. Zhang, J. Shu, and Y. Lu. ParaFS: A log-structured file system to exploit the internal parallelism of flash devices. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 87–100, 2016.