A Low-Latency Metadata Service for Geo-Distributed File Systems

Chuangwei Lin[1], Bowen Liu[1], Wei Zhou[1], Yueyue Xu[1], Xuyun Zhang[2](✉), and Wanchun Dou[1](✉)

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
{lcw,liubw,zw,xuyuey}@smail.nju.edu.cn, douwc@nju.edu.cn

[2] Department of Computing, Macquarie University, Sydney, Australia
xuyun.zhang@mq.edu.au

Abstract. Geo-distributed file systems have been widely used by web services. An increasing number of time-critical web applications have been deployed on the cloud across geographical regions. In this circumstance, intolerant service latency will occur when user accesses remote servers. In view of this challenge, We design a metadata service which aims at reducing the service invocation latency. This low-latency metadata service is named LoLaMS. Taking advantage of the latency-aware dynamic subtree partition and migration, LoLaMS is capable to handle more metadata service invocations in the nearby metadata server. On account of that, the expected latency could be satisfied by LoLaMS. We implemented the LoLaMS and deployed it in a real-word cloud environment across different regions. The experimental results show that LoLaMS reduces the network latency effectively while ensuring high metadata consistency.

Keywords: Distributed file system · Low latency · Metadata service · Wide area storage

1 Introduction

With the development of mobile edge computing, industrial Internet, Internet of vehicles and other technologies, there are more and more applications deployed and used across geographical regions. For users, timely data loading can improve their experience. For some applications, such as accessing high-precision maps in navigation systems, resource loading in VR/AR applications, data access latency is the key to user experience. These application scenarios are latency-sensitive. Users and servers are distributed in different regions far away. However, most previous designs for cloud storage services cannot provide users with low enough access latency.

Distributed file system, as a kind of distributed storage service with strong universality, has been widely used in cloud storage. More than 50% of the operations on a file system are metadata operations [12]. Therefore, low latency metadata access is an important part of a low latency distributed file system.

In most modern distributed file systems, a dedicated metadata service is designed to manage metadata. There are two metadata management mechanisms in general: a centralized mechanism and a decentralized mechanism. [25] The centralized metadata management, including PVFS [13] GFS [6] and HDFS [14] use a single metadata server (MDS) to manage metadata of all files and directories. The decentralized metadata management, including CephFS [17] and MRFS [23] use a group of MDS to manager metadata. In order to manage metadata in geo-distributed storage system, Granary [22] and IPFS [2] use Distributed Hash Table (DHT) to locate metadata.

However, most of the distributed file systems such as GFS [6], HDFS [14], Lustre [3], CephFS [17], IPFS [2] are not optimized for latency caused by geographic distance [15]. In order to reduce the latency of metadata access operation in geo-distributed file systems, this paper proposes a low-latency metadata service for geo-distributed file systems—LoLaMS.

LoLaMS is a distributed metadata service that provides strong consistency (sequential consistency). For a general-purpose file system, strong consistency is more appropriate because it can adopt most kinds of applications. This makes it easier for existing applications to use file systems based on LoLaMS without additional work on consistency issues for developers.

LoLaMS is a wide area networked metadata service which runs in data centers distributed in different geographical regions. In order to reduce access latency, LoLaMS takes advantage of the locality of data access by users. The data access behavior of many applications is characterized by geographic locality. In another word, users in a same area prefer to access similar data. For example, navigation system of vehicle only queries high-precision map of its nearby area rather than other remote areas. In file system, data is organized in a directory tree structure. Relevant data is often placed in the same or adjacent directory. LoLaMS dynamically divides the file system directory tree into subtrees based on the analysis of user's operation behavior. Each subtree is managed by a metadata server. The subtree is placed in a specific position so that as many future incoming access requests to the subtree as possible with less access latency.

We developed a prototype of LoLaMS and deployed it across 8 regions of the world. We tested the performance of LoLaMS and compared it with HDFS by using a real dataset. The results of the experiment show that in LoLaMS, the latency of 78% write operates is less than 50 ms, which is 3.36× better than HDFS. The latency of 65.6% read operates is less than 50 ms, which is 2.66× better than HDFS.

The main contribution of this paper is threefold.

- Developed a prototype framework of geo-distributed file system. This framework enables metadata to be distributed across different MDSs and ensures strong consistency of operation. This framework enables metadata migration to proceed correctly.
- Designed a migration method used in LoLaMS. This method records users' operation behavior, counts the latency between users and each node. When the latency of the user operation exceeds the pre-set threshold, the method

calculates the subtree to migrate out and the target MDS to migrate to, then migrates the metadata to the target MDS in a reliable manner.
- Developed a prototype of LoLaMS and deployed the prototype of LoLaMS across 8 regions of the world. Tested the performance of LoLaMS and compared it with HDFS by using a real dataset.

The remainder of this paper is organized as follows. Section 2 introduces the framework of a geo-distributed file system. The details of the proposed low-latency metadata service are introduced in Sect. 3. Section 4 evaluates the proposed metadata service experimentally. The related works are summarized in Sect. 5. Finally, the paper is concluded in Sect. 6.

2 The Geo-Distributed File System Framework

LoLaMS runs as a part of a geo-distributed file system. Servers of the system include 3 roles: *Manager*, *MDS* and *Datanode*. *MDS* and *Manager* provide metadata services to *Client*. Figure 1 depicts the framework of the geo-distributed file system.

Manager. A cluster of nodes (typically 3 nodes) run *Manager* program and ZooKeeper [7]. *Manager* is used to monitor the health of individual MDSs and deal with MDS failures. At a time, only one Manager node is active, and the remaining Manager nodes are standby nodes. When an active Manager node fails, ZooKeeper re-selects a standby node as the Active node. The active Manager node monitors the performance of individual MDSs. Manager senses and handles the failure when the MDS fails or goes offline. ZooKeeper ensures data consistency between manager nodes.

Metadata Server. MDSs are responsible for managing the metadata of the file system. MDS nodes are located in the data center or on the edge server. A client can access any metadata server to start using the LoLaMS. In LoLaMS, there are two types of metadata. The first type of metadata is *INode*, which describes the

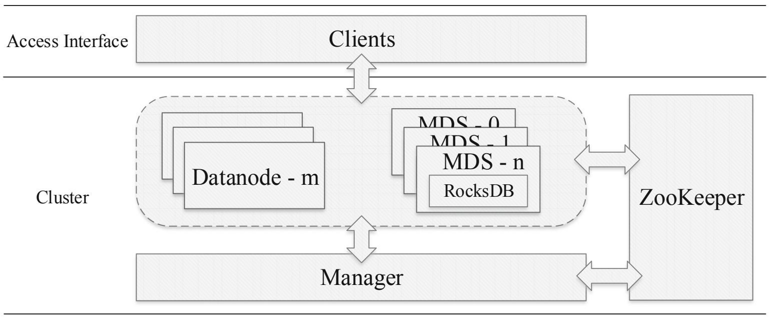

Fig. 1. The geo-distributed file system framework

directory structure and file information. Each MDS only maintains and persists the inodes within the subtree managed by the MDS. LoLaMS uses RocksDB [5], an embeddable persistent key-value store for fast storage, to persist metadata on the disk. When creating or modifying a file, the client needs to write the inode on the MDS that manages the metadata of the file. When writing an inode, RocksDB first appends the operation to the log file on the disk, and then updates the inode data in the memory. The second type of metadata is *AuthTreeNode*, which describes how the file system directory tree is partitioned and each subtree managed by which MDS. For a path, by querying the *AuthTreeNode*, can know which MDS manages the directory or file. The *AuthTreeNode* is updated only when a migration operation is performed. Replicas of the *AuthTreeNode* are available on all MDSs and managers. The replicas on the manager are the primary replica, and updates to the primary replica of the *AuthTreeNode* are broadcast to all replicas via ZooKeeper, keeping the *AuthTreeNode* data consistent on the system. Dynamic subtree partition [18] divides file system directory tree into multiple subtrees, each of them is managed by different MDS. In LoLaMS, MDSs log client operations and gather latency information between clients and MDSs. If latency higher than the threshold, MDS will use the method described in Sect. 3 to re-partition subtree and migrate metadata.

Datanode. Datanodes store file data and perform operations on file data.

Client. *Client* provides a command line interface which support common file system operations, such as *mkdir* and *ls*, which enables users to access metadata via file path such as "/aaa/bbb/ccc".

A client can connect to any known MDS to get a list of current online MDSs. The client measures the latency between itself and each MDS at regular intervals and reports latency information to all MDSs that have already connected to the client.

When a user operates on a path, the client first checks whether it has accessed the path before. If it has, the client directly accesses the MDS corresponding to this path before. If not, the client connects to the MDS nearest to the user and queries which MDS manages the directory or file that path represents.

3 A Low-Latency Metadata Service

The concept of latency in this article refers to the time elapsed from when the client sends a request to when the client receives a response. The premise of using the LoLaMS is that users' access behaviour to metadata follows the principle of locality, including temporal locality and spatial locality. LoLaMS is a distributed service that runs across MDSs. When the MDS finds that the user operation latency exceeds the threshold, LoLaMS selects a directory subtree managed by this MDS and migrates it to the appropriate MDS, so as to make the user operation latency below the threshold as much as possible, and the target MDS is not overloaded. The goal of LoLaMS is to minimize the number of operations whose latency exceeds the threshold.

To formulate the goal of LoLaMS, we define

$$p_{latency} = \frac{\text{number of operations with latency} < T_t}{\text{number of operations}}$$

T_t denotes latency threshold. $p_{latency}$ denotes the proportion of operations whose latency is less than the latency threshold among all operations.

The goal of LoLaMS can be expressed as

$$Maximize(p_{latency})$$

The optimal method to achieve this goal is migrating metadata of $inode_x$ to MDS which is closest to client before the client access $inode_x$. However, it is impossible to accurately predict every operation in a real file system.

We have the experience from the storage system that if a user has accessed a path in the recent past, it is more likely that the user will access the path or its adjacent paths in the recent future. If a user has accessed a path frequently in the recent past, there is a high probability that the user will continue to visit the path frequently in the recent future.

LoLaMS takes advantage of the locality of user access to the file system. When timeout access to $inode_i$ occurs, LoLaMS analyzes the operation behavior of users who visit $inode_i$ in a period of time to determine the migration of $inode_i$ and its subdirectories.

Figure 2 shows the flow chart of subtree partition and migration method in LoLaMS which is separated into 3 steps:

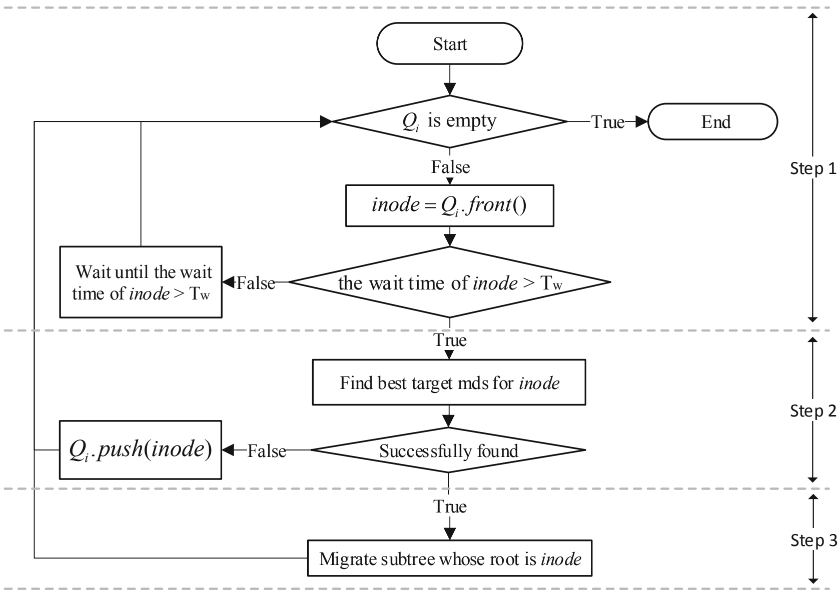
Start
Q_i is empty
True
End
False
$inode = Q_i.front()$
the wait time of *inode* > Tw
False
Wait until the wait time of *inode* > Tw
True
Find best target mds for *inode*
Successfully found
False
$Q_i.push(inode)$
True
Migrate subtree whose root is *inode*
Step 1
Step 2
Step 3

Fig. 2. The flow chart of subtree partition and migration method in LoLaMS.

Step 1. Identify Inodes that Need to Be Optimized: If an operation latency exceeds T_t, put the inodes that have been accessed in this operation into a queue. Inodes in the queue are taken out for optimization in enqueue order.

Step 2. Find Best Target MDS for Inode: Find the best MDS as a migration target for the taken out INode.

Step 3. Migrate Subtree to Target MDS: Partition and migrate a subtree from current directory trees managed by mds_i whose root is the INode which has been found best target MDS.

Identify Inodes that Need to Be Optimized. mds_i logs the operation of $inode_x$ from $client_c$ and instantly check L_c^i. L_c^i denotes network latency between $client_c$ to mds_i. If $L_c^i > T_t$, push pair $\langle inode_x, time_{now} \rangle$ into Q_i. Q_i denotes the queue of inodes that are waiting for optimization in mds_i. $time_{now}$ denotes current time. INodes in Q_i are not optimized immediately when they are pushed into Q_i. A single timeout log may be accidental. To make better optimization, INodes in Q_i have to wait for enough time to gather more logs. For each $inode_x$ in Q_i which has been waiting for enough time (denoted as T_w), try to find the best target MDS for $inode_x$.

Find Best Target MDS for Inode. Find the best MDS as a migration target for selected INode. If failed, use breadth-first search to find the best target MDS for child nodes of the current selected INode.

Algorithm 1 describes the process to find the best target MDS for $inode$. Here, mds_c denotes current mds which running this algorithm. $ep_{inode_t}(mds_i)$ means if $inode_t$ is in mds_i, the estimate proportion of operations access $inode_t$ whose latency $< T_t$. $p_{latency}^{expect}$ denotes a threshold $p_{latency}$ expected to be achieved.

We use $ep_{inode_x}(mds_y)$ to measure whether mds_y can be a migration target of $inode_x$. It means estimate $p_{latency}$ of $inode_x$ if $inode_x \in Set_{mds_y}$. Set_{mds_y} means the set of INodes managed by mds_y. $ep_{inode_x}(mds_y)$ reflects if $inode_x$ is managed by mds_y, when clients repeat operations during $[time_{now} - T_{tr}, time_{now})$, the proportion of operations whose latency is less than T_t. Here, T_{tr} means the trace-back time of operation logs. In a file system, users usually access files from top to bottom according to the directory hierarchy. For inodes in a subtree that use $inode_x$ as root, clients normally need to access $inode_x$ before access other inodes. Therefore, $ep_{inode_x}(mds_y)$ is approximate to estimate $p_{latency}$ of the subtree. If we find a suitable MDS as migrate target for $inode_x$, which improved $p_{latency}$ of $inode_x$, then the estimate $p_{latency}$ of the subtree is equivalently improved. If a suitable MDS cannot be found for $inode_x$ as the migration target, a breadth-first search will be used to find a suitable migration target for other nodes in the subtree. If found a suitable migration target for $inode_z$, the $inode_z$ will be the root node to divide a new subtree. This subtree will be migrated to the target MDS.

Algorithm 1. The process to find the best target MDS

```
Require: $inode$: The INode need to optimize; MDS_list: The list of all mds;
Ensure: state: a boolean value represents whether found target mds or not; $inode_t$:the
    root of subtree to be partitioned; $mds_t$:the migrate target mds;
 1: q.push($inode$) // q denotes the queue used for breadth-first search.
 2: while q NOT empty do
 3:     $inode_t$ = q.pop();
 4:     if $time_{now} - time_{lo}^{inode_i}$ then
 5:         l = list of clients accessed $inode_t$ during $[time_{now} - T_{tr}, time_{now})$
 6:         $count_{client_x}$ = the access time from $client_x$ during $[time_{now} - T_{tr}, time_{now})$
 7:         $time_{lo}^{inode_t} = time_{now}$
 8:         if l is empty then continue;
 9:         for $mds_i$ in MDS_list do
10:             in_threshold_req = 0;
11:             for $client_j \in l$ do
12:                 if $L_j^i < T_t$ then in_threshold_req+=$count_{client_j}$
13:             $ep_{inode_t}(mds_i) = in\_threshold\_req / \sum_{client_t \in l} count_{client_t}$
14:         $mds_t$ = the mds which has the largest $ep_{inode_t}$
15:         if $ep_{inode_t}(mds_t) > p_{latency}^{expect}$ then return (True, $inode_t$, $mds_t$)
16:         else q.push(childnodes of $inode_t$)
17: return False
```

Migrate Subtree to Target MDS. Partition a subtree from current directory trees managed by mds_i whose root is the INode which has been found best target MDS. Then migrate this subtree to target MDS.

The migration process is depicted in Fig. 3. If an MDS doesn't receive broadcast information due to failure, it will get the latest "UpdateAuthTree" message through zookeeper after it returns to normal. The MDS will fetch all missing updation transactions of *AuthTree* from *Manager*, then redo these transactions. This ensures the consistency of *AuthTree* in the system. Thus, at any given moment, an INode will only be managed by one MDS.

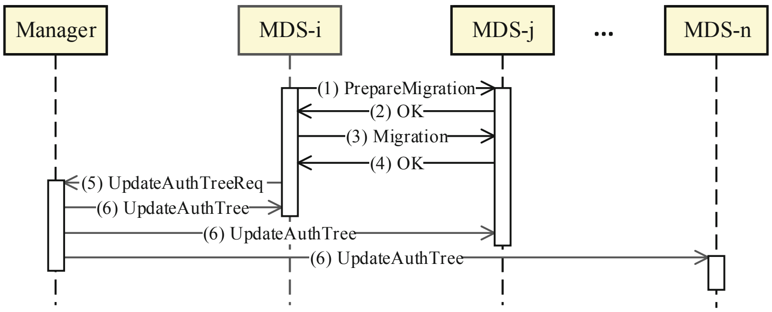

Fig. 3. Sequence diagram of migration.

If mds_i intent to migrate a subtree to mds_j, the migration process is consists of 6 phases.

1. mds_i send "PrepareMigration" message to mds_j to ask if mds_j has enough storage space.
2. mds_j send "OK" message to mds_i to indicate that mds_j is ready to receive the migration data. If mds_i does not receive any message from mds_j, mds_i will find another migration target MDS.
3. mds_i send the metadata of the inodes in the subtree to mds_j through "Migration" message.
4. After mds_j receives the "Migration" message, it persists the data and then replies "OK" message to mds_i.
5. mds_i send "UpdateAuthTreeReq" to *Manager* to commit the migration. In *Manager*, The change of *AuthTree* is entered into the database as a transaction.
6. The *Manager* send message "UpdateAuthTree" to broadcast the change of *AuthTree*.

4 Experiment Evaluation

4.1 Experiment Settings

Experiment Environment. Table 1 shows the experiment environment. We evaluate LoLaMS using Aliyun's elastic cloud service. We deploy 9 VMs in 8 different regions. These regions are: cn-beijing (China), cn-shenzhen (China), ap-northeast (Japan), ap-southeast (Australia), us-west (US), ap-south (India), eu-west (London), ap-southeast (Malaysia). Each VM has 2 virtual CPUs, 8 GiB memory and 50 GiB storage. For LoLaMS, a VM in cn-beijing region is used to deploy the Manager, 8 MDSs are deployed in 8 VMs each in a different region. For HDFS, a VM in cn-beijing region is used to deploy the Namenode, 8 Datanodes are deployed in 8 VMs each in a different region.

Table 1. Experiment environment.

Item	Description
Number of servers	9
Region of servers	2 in cn-beijing (China), 1 in cn-shenzhen (China), 1 in ap-northeast (Japan), 1 in ap-southeast (Australia), 1 in us-west (US), 1 in ap-south (India), 1 in eu-west (London), 1 in ap-southeast (Malaysia)
CPU	2 vCPU, 3.5GHz
Memory	8 GiB
Stroage	50 GiB
Operation System	Ubuntu 18.04 $\times$ 64
Network bandwidth	100 Mbps

Experiment Data. In the experiment, we use an online shopping platform access log dataset [24] to test the read-write latency and throughput of LoLaMS and HDFS. We take the first 100,000 access records from the dataset for the evaluation. Each user in the dataset is mapped to a specific region. In the 8 regions, we replay the access records of the users mapped to this region in order at the same beginning time.

4.2 Experiment Comparison Analysis

Latency Evaluation. We measured the latency between these VMs. The round-trip time (RTT) between these regions within the range of 35 and 420 ms. The geographical distance between data centers will inevitably cause latency. In addition, there are other factors that can influence latency, such as network conditions. The average RTT between ap-south-1 and cn-shenzhen is 410 ms while the distance between them is 4280 km. The average RTT between ap-south-1 and us-west-1 is 249 ms while the distance between them is 13545 km. The geographical distance between ap-south-1 and us-west-1 is 3× that between ap-south-1 and cn-shenzhen, but the RTT between ap-south-1 and us-west-1 is 0.6× that ap-south-1 and cn-shenzhen. Therefore, it is not credible to estimate the latency between two nodes according to the geographical location, which can only be used as the lower bound. There are other factors that can affect network latency, such as the number of hops in routing.

Typically, end-user devices (smartphones, tablets, laptops), as well as IoT devices, can reach an edge server with a rather low latency less than 10 ms [4]. The latency between users and clients in the same region usually no more than 50 ms. In the experiment, we set T_t in LoLaMS to 50 ms.

Table 2 shows the latency performance of LoLaMS and HDFS. The average write latency of LoLaMS is 38.14 ms, which is 16.75% of HDFS. The average read latency of LoLaMS is 35.98% of HDFS. In LoLaMS, the latency of 78% write operates is less than 50 ms, which is 3.36× better than HDFS. The latency of 65.6% read operates is less than 50 ms, which is 2.66× better than HDFS.

Table 2. Operation latency between LoLaMS and HDFS.

Metadata service	Average latency (ms)	Median latency (ms)	$p_{latency}$
LoLaMS (write)	38.14	5	0.780
HDFS (write)	227.69	153	0.179
LoLaMS (read)	61.17	9	0.656
HDFS (read)	170	148	0.179

Figure 4 shows the cumulative distribution function of operation latency in LoLaMS and HDFS. Figure 5 depicts the proportion of operation latency in different ranges. In LoLaMS, 67.2% of write operations are completed within 10 ms, and 54.3% of reading operations are completed within 10 ms. Compared

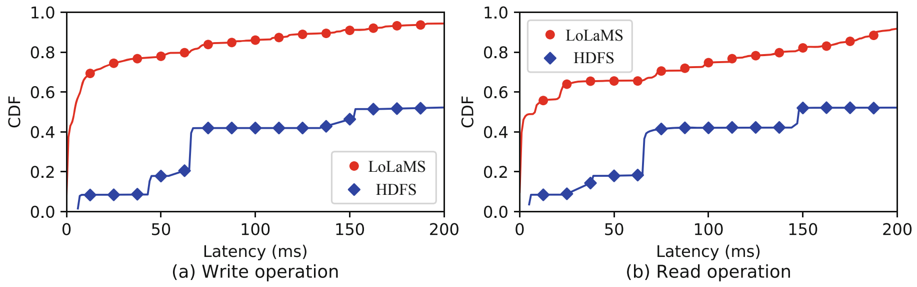

Fig. 4. Cumulative distribution function (CDF) of operation latency.

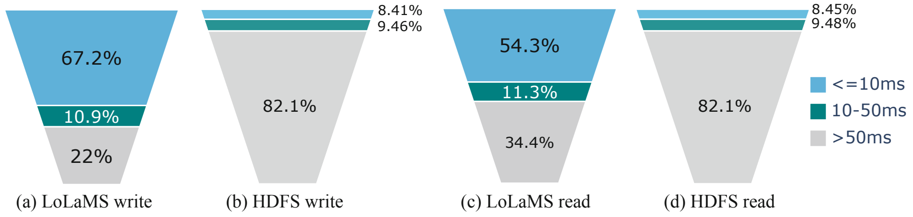

Fig. 5. The proportion of latency in different ranges.

with HDFS, LoLaMS can reduce the latency significantly. Most operations can be completed in a short time, which can make the application run faster and improve user experience.

For metadata write operations from clients, both LoLaMS MDS and HDFS Namenode first append the operation to the log file on the disk. Then they update the data in the memory. Finally, they return a confirmation to the client. The latency of an operation comes from two parts: the processing time on the server and the transmission time on the network. In LoLaMS, since a large part of client operations can be completed in nearby MDS, the transmission time is significantly reduced. This makes LoLaMS have a lower latency than HDFS.

In the experiment, the average read operation latency of LoLaMS is higher than the average write operation latency. This is because the experiment replayed the dataset twice. The first replay executed the records as write operations and construct the directory structure. The second replay executed the records as read operations. Replaying the dataset for the second time breaks the temporal locality of the access record. Thus, fewer operations were executed on the nearby MDS during the second replay.

Throughput Evaluation. Previous experiments show that HDFS can reach 126100 ops/s when it performs read operations on Namenode [14]. However, in a geo-distributed environment, the bottleneck of system throughput is no longer the single machine performance, but the network performance. Figure 6 and Fig. 7 show the operation per second of write and read operation in LoLaMS and HDFS. LoLaMS reduces the latency significantly, which also improves the

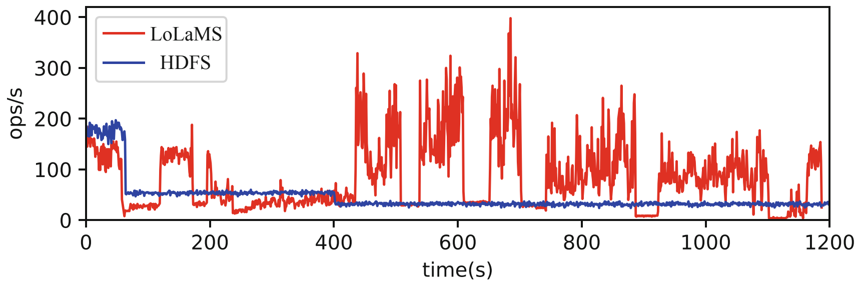

Fig. 6. Operation per second (OPS) of write operation.

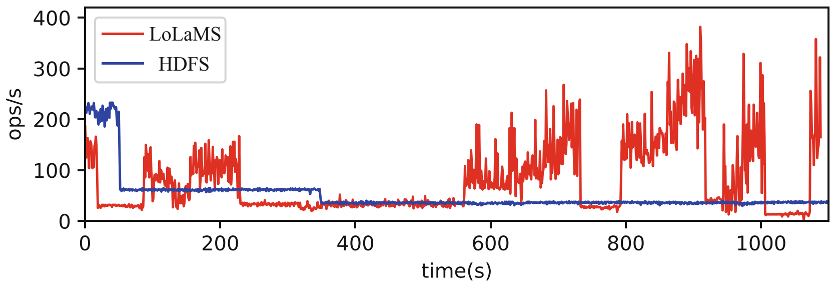

Fig. 7. Operation per second (OPS) of read operation.

throughput of the system. The maximum throughput of HDFS is 197 ops/s for write operations and 233 ops/s for read operations. The maximum throughput of LoLaMS is 389 ops/s for write operations and 382 ops/s for read operations. In terms of maximum read throughput, LoLaMS is 63.9% higher than HDFS. For maximum read throughput, LoLaMS is 97.4% higher than HDFS. This is due to LoLaMS's full use of multiple metadata servers and reasonable division of subtrees. The higher throughput also enables LoLaMS to complete the same number of read and write requests in a shorter time than HDFS.

4.3 Evaluation Result

From the experimental results of latency evaluation, LoLaMS performs significantly better than HDFS for the indicator $p_{latency}$. This shows that in LoLaMS, more operations can be completed with lower latency. According to the Throughput evaluation results, the Throughput of LoLaMS is higher than HDFS. This shows that LoLaMS can handle more operations in a shorter time. For file systems in geo-distributed scenarios, LoLaMS has advantages in terms of latency and throughput.

5 Related Works

Web services have made extensive use of distributed file systems, which can share storage for many users in the same namespace [8]. With the rise of cloud computing and online services, more and more applications with high timeliness are being deployed across borders. The latency induced by geographical distance, on the other hand, is difficult to overcome. In recent years, some researches have been done to address the latency associated with geo-graphical distribution.

Yu et al. [22] develop a distributed storage system called Granary that offers cyber users with a dependable data storage and sharing service. Granary stores file meta-data in a Distributed Hash Table layer and scatters big files using a raw data storage technique. Benet [2] design a peer-to-peer distributed file system InterPlanetary File System (IPFS) that aims to connect all computing devices to a single file system. IPFS is a content-addressed block storage architecture with content-addressed hyper connections that enables high throughput.

The above two studies used the method of storing metadata and file data nearby to reduce latency. In addition, it can also reduce system latency by caching metadata on the server side or on the client side. Yu et al. [23] proposed the Metadata Replication File System(MRFS) which split metadata into non-overlapping portions and kept on MDS, where the creation action is raised, whereas namespace and directory information is preserved in Namespace Servers in this system. Because it serves the majority of requests in local MDS, such a hierarchical architecture not only achieves great scalability but also delivers low latency. Oh et al. [9] present a geo-distributed cloud storage system called Wiera. Wiera provides first-class dynamic support owing to network, workload, and access pattern changes, and can actively handle dynamism at run-time. By externalizing the policy definition, Wiera allows unmodified programs to benefit from flexible data/storage rules. Ren et al. [11] develop SLOG that avoids the tradeoff for workloads which contain physical region locality in data access. In the access mode, each data particle is assigned a master region based on locality. Reading and writing to nearby data can be done quickly without cross-regional communication. Muhammed et al. [16] proposed an approach use erasure coding to significantly reduce costs while successfully mitigating the associated overheads in wide-area latency incurred for preserving consistency.

Exsiting work mainly focus on storing file data on the nearby server or caching metadata. While our research focus on reducing latency in metadata service without sacrificing consistency. In cloud computing and edge computing, workload migration is often used to improve the quality of experience (QoE) and the quality of service (QoS) [10,19,20]. While our idea is to improve QoS by migrating storage workload. In this paper, we propose a geo-distributed metadata service, LoLaMS, which uses the method of partition the directory tree dynamically and migrating subtree to reduce latency while ensuring the consistency of metadata. Virtualization technology has been widely used in cloud computing and edge computing [1,21]. By combining virtualization technology, distributed file systems based on LoLaMS is expected to provide low-latency storage services for cloud computing and edge computing.

6 Conclusion

This paper introduces LoLaMS, a low-latency metadata service for geo-distributed file system. LoLaMS can provide metadata service with lower latency, without sacrificing strong consistency. The design of LoLaMS makes full use of the temporal locality and spatial locality of the user's access to the file system. LoLaMS senses the user's network latency and dynamically divides the file system directory tree into several subtrees according to the user's access behavior. In the running process of LoLaMS, the division of subtree is optimized to make as many user requests as possible be completed on the nearest MDS. The experimental results show that compared with other file system metadata service, LoLaMS can significantly reduce the access latency and improve the throughput of the whole system.

Acknowledgements. This work was supported in part by the National Key Research and Development Program of China under Grant No. 2020YFB1707601, the Key Research and Development Program of Jiangsu Province under Grant No. BE2019104. Dr Xuyun Zhang is the recipient of an ARC DECRA (project No. DE210101458) funded by the Australian Government.

References

1. Alves, M.P., Delicato, F.C., Santos, I.L., Pires, P.F.: LW-CoEdge: a lightweight virtualization model and collaboration process for edge computing. World Wide Web **23**(2), 1127–1175 (2020)
2. Benet, J.: IPFS-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)
3. Braam, P.: The lustre storage architecture. CoRR abs/1903.01955 (2019). http://arxiv.org/abs/1903.01955
4. Confais, B., Lebre, A., Parrein, B.: Performance analysis of object store systems in a fog and edge computing infrastructure. In: Hameurlain, A., Küng, J., Wagner, R., Akbarinia, R., Pacitti, E. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXIII. LNCS, vol. 10430, pp. 40–79. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-55696-2_2
5. Facebook: Rocksdb. http://rocksdb.org/
6. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 29–43 (2003)
7. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: wait-free coordination for internet-scale systems. In: USENIX Annual Technical Conference, vol. 8 (2010)
8. Kubiatowicz, J., et al.: Oceanstore: an architecture for global-scale persistent storage. ACM SIGOPS Oper. Syst. Rev. **34**(5), 190–201 (2000)
9. Oh, K., Qin, N., Chandra, A., Weissman, J.: Wiera: policy-driven multi-tiered geo-distributed cloud storage system. IEEE Trans. Parallel Distrib. Syst. **31**(2), 294–305 (2019)
10. Qi, L., Chen, Y., Yuan, Y., Fu, S., Zhang, X., Xu, X.: A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems. World Wide Web **23**(2), 1275–1297 (2020)

11. Ren, K., Li, D., Abadi, D.J.: Slog: serializable, low-latency, geo-replicated transactions. Proc. VLDB Endowment **12**(11), 1747–1761 (2019)
12. Roselli, D.S., Lorch, J.R., Anderson, T.E., et al.: A comparison of file system workloads. In: USENIX Annual Technical Conference, General Track, pp. 41–54 (2000)
13. Ross, R.B., Thakur, R., et al.: PVFS: a parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, pp. 391–430 (2000)
14. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10. IEEE (2010)
15. Singh, H.J., Bawa, S.: Scalable metadata management techniques for ultra-large distributed storage systems-a systematic review. ACM Comput. Surv. (CSUR) **51**(4), 1–37 (2018)
16. Uluyol, M., Huang, A., Goel, A., Chowdhury, M., Madhyastha, H.V.: Near-optimal latency versus cost tradeoffs in geo-distributed storage. In: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20), pp. 157–180 (2020)
17. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D., Maltzahn, C.: Ceph: a scalable, high-performance distributed file system. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, pp. 307–320 (2006)
18. Weil, S.A., Pollack, K.T., Brandt, S.A., Miller, E.L.: Dynamic metadata management for petabyte-scale file systems. In: SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, p. 4. IEEE (2004)
19. Xu, X., et al.: Secure service offloading for internet of vehicles in SDN-enabled mobile edge computing. IEEE Trans. Intell. Transp. Syst. **22**(6), 3720–3729 (2020)
20. Xu, X., Zhang, X., Gao, H., Xue, Y., Qi, L., Dou, W.: Become: blockchain-enabled computation offloading for IoT in mobile edge computing. IEEE Trans. Ind. Inf. **16**(6), 4187–4195 (2019)
21. Yang, S., Wang, X., Wang, X., An, L., Zhang, G.: High-performance docker integration scheme based on openstack. World Wide Web **23**(4), 2593–2632 (2020)
22. Yu, H., Zhang, F., Wu, Y.: Granary: a sharing oriented distributed storage system. Future Gen. Comput. Syst. **38**, 47–60 (2014)
23. Yu, J., Wu, W., Yang, D., Huang, N., et al.: MRFS: a distributed files system with geo-replicated metadata. In: Sun, X. (ed.) ICA3PP 2014, Part II. LNCS, vol. 8631, pp. 273–285. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11194-0_21
24. Zaker, F.: Online Shopping Store - Web Server Logs (2019). https://doi.org/10.7910/DVN/3QBYB5
25. Zhou, J., Chen, Y., Wang, W., He, S., Meng, D.: A highly reliable metadata service for large-scale distributed file systems. IEEE Trans. Parallel Distrib. Syst. **31**(2), 374–392 (2019)