CLMS: Configurable and Lightweight Metadata Service for Parallel File Systems on NVMe SSDs

Qiong Li[1,2], Shuaizhe Lv[1], Xuchao Xie[1](✉), and Zhenlong Song[1]

[1] College of Computer, National University of Defense Technology, Changsha, China
xiexuchao@nudt.edu.cn
[2] Defense Innovation Institute, Academy of Military Sciences, Beijing, China

Abstract. With the tendency of running large-scale data-intensive applications on High-Performance Computing (HPC) systems, the I/O workloads of HPC storage systems are becoming more complex, such as the increasing metadata-intensive I/O operations in Exascale computing and High-Performance Data Analytics (HPDA). To meet the increasing performance requirements of the metadata service in HPC parallel file systems, this paper proposes a Configurable and Lightweight Metadata Service (CLMS) design for the parallel file systems on NVMe SSDs. CLMS introduces a configurable metadata distribution policy that simultaneously enables the directory-based and hash-based metadata distribution strategies and can be activated according to the application I/O access pattern, thus improving the processing efficiency of metadata accesses from different kinds of data-intensive applications. CLMS further reduces the memory copy and serialization processing overhead in the I/O path through the full-user metadata service design. We implemented the CLMS prototype and evaluated it under the MDTest benchmarks. Our experimental results demonstrate that CLMS can significantly improve the performance of metadata services. Besides, CMLS achieves a linear growth trend as the number of metadata servers increases for the unique-directory file distribution pattern.

Keywords: High-Performance Computing · Parallel File System · CLMS · Metadata Service · Storage System

1 Introduction

Integrating the new emerging NVMe SSDs and Non-Volatile Memory (NVM) into the storage hierarchy of High-Performance Computing (HPC) systems is attractive, especially as the growing tendency of running large-scale data-intensive applications on HPC systems [2]. Nowadays, almost all the top supercomputers in the world employ a large number of NVMe SSDs to build Burst Buffer which provides applications with a high-performance I/O acceleration layer [8,20]. Typically, compared with the HDD-based parallel file system (PFS),

C. Li et al. (Eds.): APPT 2023, LNCS 14103, pp. 101–112, 2024.
https://doi.org/10.1007/978-981-99-7872-4_6

the NVMe SSD-based BB can improve HPC I/O performance by one to two orders of magnitude. In terms of Burst Buffer implementations, NVMe SSDs can be placed on a dedicated I/O Node (ION) between compute nodes and PFS as shared Burst Buffer, or simply placed on compute node as node-local Burst Buffer [17,19].

With the price of NVMe SSDs continuously decreasing, HPC systems are trying to construct PFS using only NVMe SSDs to achieve high I/O performance with a fixed Total Cost of Ownership (TCO) for storage systems. The legacy PFS implementations for HPC storage systems are specially designed to provide high bandwidth for parallel file accesses [4,14,15]. For metadata-intensive workloads with a large number of data synchronization, non-continuous random access, and small I/O requests, PFS potentially incurs high I/O latency and low throughput for its inefficiency of metadata performance [5,9,11,18]. Thus, how to design the specific PFS for NVMe SSDs and break through the metadata performance bottleneck of existing PFS is critical for HPC storage systems.

Data-intensive applications will inevitably generate hot data directories, therefore a large number of file and sub-directory create, delete, modify, and find operations will be frequently triggered in the hot directories. Once the metadata of the hot data is located in a single or small number of metadata servers (MDS), the load unbalance issue of MDS will result in poor metadata scalability, and the overall PFS performance is severely limited [10]. For example, BeeGFS adopts a static directory partition method for load balancing, metadata accesses to a large directory will be redirected to a single MDS. BeeGFS cannot efficiently handle the metadata operations to hot directories. GekkoFS can appropriately solve such hot directory problems through a hash-based metadata distribution strategy. However, the hash-based metadata distribution will lead to a poor locality of metadata accesses, which generates a large number of inter-server I/O communication overhead in PFS.

Existing PFS is usually constructed in a superimposed style that metadata and data in PFS are stored on the local filesystem of MDS and OSS, such as EXT4, XFS, etc. However, due to the small size of PFS metadata, storing each PFS metadata as a single file in the local filesystem is inefficient [3,6,12]. This is because the getattr, setattr, and other system calls in the I/O path will incur significant user and kernel mode switching overhead. In addition, the inode and directory blocks of the local filesystem are not optimized for parallel I/O [7,16]. Each block can only be accessed by one process at a time, and it will bring a large number of serial processing operations when there are multiple processes to access a single PFS directory, resulting in serious locking competitive overhead.

In this paper, we propose CLMS, a configurable and lightweight metadata service for the parallel file systems on NVMe SSDs. CLMS introduces a configurable metadata distribution policy that simultaneously enables the directory-based and hash-based metadata distribution strategies and can be activated according to the application I/O access pattern, thus improving the processing efficiency of metadata accesses from different kinds of data-intensive applications. CLMS further reduces the memory copy and serialization processing overhead in the I/O path through the userspace metadata service design. We implemented the CLMS prototype and evaluated it under the MDTest benchmarks. Our experimental results demonstrate that CLMS can significantly improve the performance of metadata services. Besides, CMLS achieves a linear growth trend as the number of metadata servers increases for the unique-directory file distribution pattern.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 provides an overview of CLMS architecture. Section 4 and Sect. 5 describe the detailed configurable metadata distribution and lightweight metadata service designs of CLMS. We evaluate CLMS in Sect. 6 and conclude the paper in Sect. 7.

2 Related Work

HPC storage systems usually equip a limited number of metadata servers for PFS. As the th increasing metadata-intensive I/O operations in Exascale computing and High-Performance Data Analytics (HPDA), the metadata performance has been one of the PFS performance bottleneck. Chen et al. [3] proposed to redirect the metadata operations to high-performance NVMe SSDs, thereby achieving higher metadata performance. Wang et al. [19] build a distributed key-value store based on the local burst buffer for metadata to alleviate the performance bottleneck of the key-value store under burst and concurrent I/O workloads. Ren et al. [13] propose IndexFS that each server is responsible for managing part of the PFS metadata. IndexFS dynamically divides the metadata according to the directory set and package file metadata and small files into SSTable based on the LSM Tree, thus the random metadata can be written in sequential and the performance of metadata can be significantly improved.

However, managing metadata in IndexFS needs to deploy a dedicated metadata management server and may cause resource waste for a large number of MDS. Zheng et al. [21] propose DeltaFS to provide a private local metadata server for each application process, and use the data path of the underlying PFS to realize the metadata management. The application obtains the metadata snapshot required through a public registration service before runtime. All metadata operations are completed on local private servers, thereby avoiding unnecessary metadata synchronization overhead. Once the application finishes its execution, DeltaFS will push the output result as a new snapshot to the global registration service. However, DeltaFS is not suitable for applications that need frequent inter-process interaction during the running process.

3 System Overview

In the scenario of data-intensive computing, PFS has to handle a large number of metadata-intensive I/O workloads. Metadata is usually divided into dentry and inode. Specifically, dentry of a file or directory is used to record the file and sub-directory ID, stat data, parent directory ID, strip distribution information, etc. inode of a directory is used to record the directory ID, the number of file and sub-directory of this directory, stat data, parent directory ID, strip distribution information, etc. The efficiency of metadata service depends mainly on two factors. One is whether the metadata distribution has good locality and parallelism, the other is the access efficiency of a single metadata service. When providing parallel file system services with different I/O workloads, PFS is difficult to satisfy all application I/O modes through a single metadata distribution strategy. To effectively improve the PFS metadata performance under different types of data-intensive applications, and fully utilize the performance advantage of NVMe SSDs, this paper proposes CLMS that enables a configurable metadata distribution method for PFS and matches metadata distribution strategies according to the application I/O characteristics. CLMS also designs metadata services based on NVMe SSD features and the key-value store to achieve high-performance metadata handling in MDS.

There are two kinds of hot data in PFS, i.e., the hot directory where most data operations are concentrated in a single directory, and the hot files that are intensively accessed but distributed in multiple directories. Once the metadata of the hot directory or files are distributed in a small amount of MDS, the overall PFS performance will be limited by the metadata handling efficiency of each MDS. As a result, PFS cannot provide desired high-performance and scalable parallel file access services for large-scale HPC systems. Different from existing PFS, CLMS enables both directory-based and hash-based metadata distribution to simultaneously provide access locality and parallelism for the metadata of hot data.

Traditional PFS directly stores its metadata and data on the local file system on MDS and OSS, which potentially brings serious performance problems to PFS. This is because these file systems strictly follow the POSIX semantics and frequently perform system calls such as file open, close, etc., which will cause frequent user and kernel mode switching and eventually lead to low local file system performance. Besides, the legacy I/O software stack introduces significant performance overhead in PFS as each PFS metadata and data go through local file systems, I/O scheduling layers, general block layers, and block device drivers in MDS and OSS, resulting in additional system management overhead and performance reduction. CLMS designs and implements a high-performance local storage engine for metadata services based on NVMe SSD features and Key-Value store, and relaxes POSIX semantics to achieve fast persistent storage of directory and file metadata, thus reducing the software performance overhead introduced by the local file system.

4 Configurable Metadata Distribution

4.1 Directory-Based Distribution

In the directory-based metadata distribution strategy, each directory in PFS will be attached to a metadata service that processes its metadata operations, as shown in Fig. 1, the metadata of directory A is processed by MDS0, then the metadata of all files in directory A will be stored on MDS0. When creating a

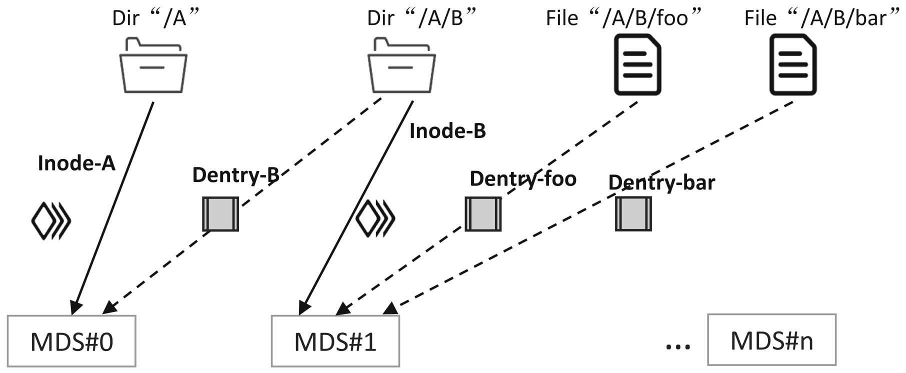

Fig. 1. Directory-based Metadata Distribution

subdirectory B in directory A, CLMS needs to randomly pick a new MDS node from the available metadata service nodes to store the inode of directory B, but the dentry information of directory B is still stored on MDS0. It should be noted that in the directory-based metadata distribution strategy, the metadata operations of the root directory are always handled by MDS0, which contains the information of all subdirectories under the root directory, including the link to the MDS where the subdirectory is located, so that there is a defined fixed entry point in the CLMS directory tree, and PFS clients can find the metadata service node responsible for a specific directory by traversing the directory tree. In the directory-based metadata distribution strategy, all files in a single directory are distributed in the same MDS node, and when the application intensively processes a large number of files in a single directory, PFS performance will be limited by the efficiency of the metadata service of a single MDS node.

4.2 Hash-Based Distribution

For applications where hot data is centralized in a few directories, CLMS provides a metadata distribution strategy based on the hash of the file path. In this hash-based metadata distribution design, the metadata of directories and files is hashed under the VFS schema and distributed to multiple MDS nodes, and the files are no longer distributed on the same MDS nodes following the parent directory. Ash shown in Fig. 2, CLMS executes the hash function on ¡parent_directory_uuid, file_name¿ combination, selects the MDS node responsible for processing file metadata requests, and forwards the subsequent operations of the file to the daemon process of the MDS node for execution, to achieve the uniform distribution of file metadata among all MDS nodes. In this way, the metadata of files and directories is distributed pseudo-randomly among MDS nodes. In the hash-based distribution policy, the number of MDTs can be configured for directory hash distribution. CLMS will select the MDT that meets the satisfaction according to this information, the MDT with the lowest utilization will be selected by default and prioritized in the MDT list. Once the

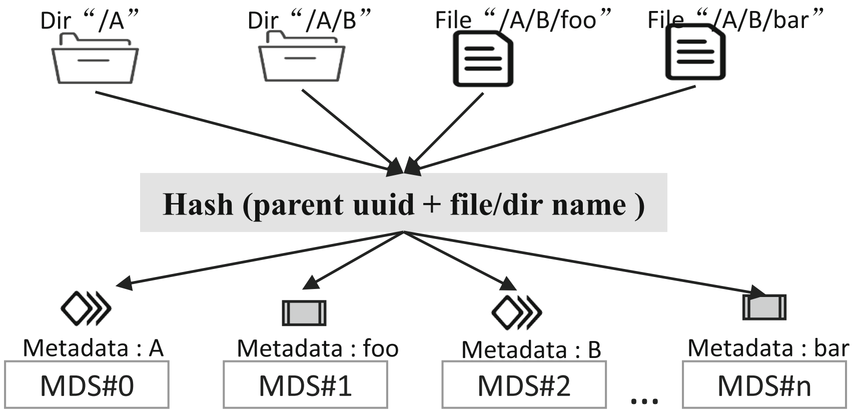

Fig. 2. Hash-based Metadata Distribution

selection is completed, the directory metadata will record the user-configured MDT list, and the subdirectory and file metadata will be distributed according to the MDT list.

5 Lightweight Metadata Service

5.1 Efficient Key-Value Store

CLMS designs and implements a high-performance metadata storage engine based on NVMe SSD features and user-mode key-value store for metadata services. This MDS-specific storage engine can achieve fast storage performance for metadata and reduce the software performance overhead introduced by traditional local file systems. CLMS implements user-mode key-value store instead of the local file system as a metadata storage engine, as shown in Fig. 3, the metadata storage engine divides the NVMe SSD address space into fixed-size slots, each slot is used to store directory or file-related metadata, and all slots are organized by a simple and efficient hash table data structure, where key is the ID of the directory or file, and value is the address pointer of the metadata. To accelerate the insertion, deletion, and search of metadata in the hash table, CLMS further introduces region locks and locates the region locks to be used according to the Key. This region lock design is used to improve the parallel processing capabilities of the metadata storage engine and abandon the inefficient mutex lock to further reduce the lock overhead in the metadata storage path.

CLMS assigns a UUID (Universally Unique Identifier) to each directory or file and uses the UUID as part of the directory and file metadata index. To save memory, only the key fields of inodeID, parentID, name, and entryType of file and directory metadata are stored and organized in memory, and the remaining fields are stored in NVMe SSDs and read when needed. There are three types of hash tables, one is the UID Hash Table (UHT) with the object

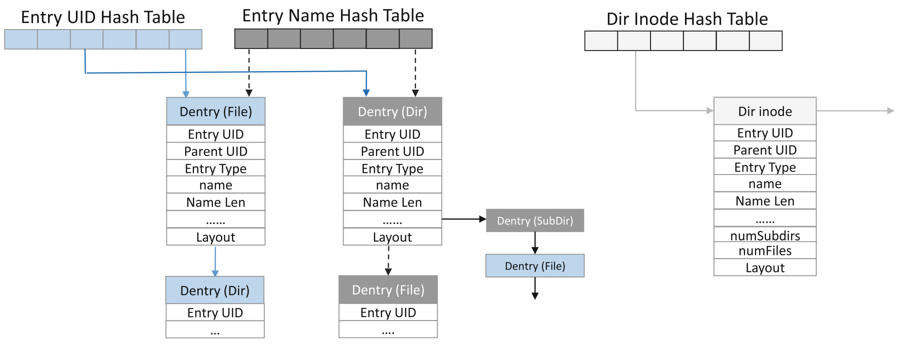

Fig. 3. MDS-Specific Key-Value Store in CLMS

entry UID as the key, which is used to quickly find the metadata of the object according to the entry UID. The second is the Name Hash Table (NHT) with the object entry name as the key, which is used to quickly find the metadata of the object according to the entry name. The third is the Inode Hash Table (IHT) built specifically for directory inodes, which is used to retrieve directory inode information. For the same catalog item, join UHT with object UID as key and NHT with object name as the key. Based on this design, CLMS metadata service provides two types of API interfaces, one is to find the corresponding directory or file metadata according to the parent directory UUID (parent_uuid) and name. The second is to directly manipulate the corresponding metadata according to the UUID of the directory or file. Therefore, whether it is based on directory-based distribution metadata or hash-based distribution metadata, the local KV storage engine can effectively support it.

5.2 POSIX Semantic Relaxation

The inode and directory blocks of traditional file systems are not designed for parallel access, as only one block can be accessed by one process at a time. When a large number of files are created from multiple processes in a single directory, this introduces significant serial processing and performance penalty. As shown in Fig. 4, the same is true in distributed file systems, for example, for file and directory creation and deletion operations, the parent directory must be locked first, and the parent directory is updated after the file and directory operation is completed. To relax POSIX semantics in CLMS, instead of using directory blocks that are difficult to use in a distributed environment, each directory entry is stored in the KV store of the daemon process, and the current state of the directory is not guaranteed to be returned in these indirect file system operations when the directory contents are requested. For example, the readdir() operation follows the eventual consistency model, that is, operations such as creating and deleting files in the same directory are no longer processed serially with parent directory updates, and the serial lock of the parent directory is removed. When

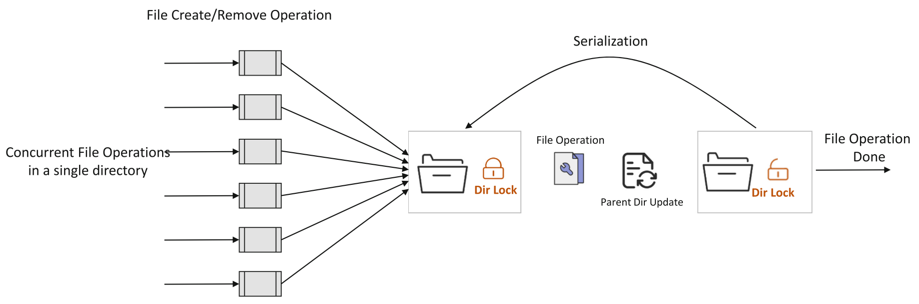

Fig. 4. Serialized File Operations with Legacy Filesystems

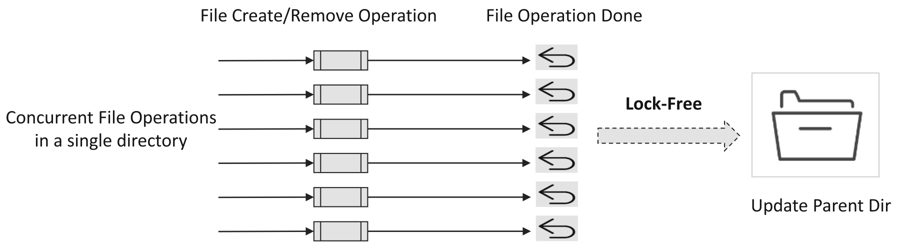
File Create/Remove Operation
File Operation Done
Concurrent File Operations in a single directory
Lock-Free
Update Parent Dir

Fig. 5. Parallel File Operations with POSIX Semantic Relaxation

deleting a file, the data corresponding to the file also needs to be deleted, which requires communication with the data storage server and is a time-consuming process. In CLMS, we can also configure this strategy, implement asynchronous processing, and delete file data in the background.

As shown in Fig. 5, CLMS relaxes POSIX directory semantics. The creation of directories is similar to files, and directories are configured with hash-based distribution, and the contents of directories can only be collected from distributed inode entries, that is, the contents of directories must be traversed through the hash distribution nodes associated with the directory. Although there are multiple node communications, the problem can be mitigated by reading as many directory entries as possible into the buffer at once. In the rename operation, metadata may need to be re-migrated to the new MDS based on hash values due to the directory or file name changes, but due to the unchanged UUID of the directory or file and the path traversal lookup mechanism, unlike GekkoFS and LocoFS that use the path of a file system object as an index within a flat namespace, when a directory is moved to a different file system path, the path of all its contents must also be modified recursively.

6 Performance Evaluation

6.1 Experimental Setup

We built a 10-node HPC system to evaluate the performance of CLMS, each compute node is equipped with 2 Intel Xeon Gold 6134 CPUs, 512 GB of node memory, 2 NVMe SSDs to build RAID for storing metadata, and 8 NVMe SSDs for object storage. In the experiment, the cluster interconnects all compute nodes using a 100 Gbps RDMA network. We evaluate CLMS performance using the MDTest [1] benchmarks in both Unique Dir and Single Dir modes, where Unique Dir means that the files and file operations of each process are in separate directories, and Single Dir means that the files and file operations of all processes are in the same directory. MDTest simulates common metadata-intensive HPC workloads to evaluate the metadata performance of CLMS and BeeGFS.

To comprehensively evaluate the behavior of CLMS, we introduce BeeGFS, CLMS-Dir, and CLMS-Hash as comparison candidates. Both CLMS-Dir and CLMS-Hash implementations follow the basic principles of CLMS designs but with different metadata distribution schemes, i.e., CLMS-Dir uses directory-based distribution and CLMS-Hash uses hash-based distribution. In HPC systems, the concurrent metadata operations in a single directory are an important workload in many applications, we tested the performance of file Create, Stat, and Remove operations using 100,000 zero-byte files per process and 16 processes per node. The x-axis of the evaluation results shown in Sect. 6.2 represents the number of nodes, up to 10 nodes, and the y-axis represents OPS for each kind of I/O workload.

6.2 Evaluation Results

In the first set of tests, we use Single Dir workload, i.e., all processes of MDTest run in a single directory, to evaluate the performance of the metadata distribution strategy and the dedicated storage engine design in CLMS. Figure 6 shows the OPS of file Create, Stat, and Remove operations. CLMS-Dir uses the same directory-based metadata distribution strategy as BeeGFS, and CLMS-Dir introduces the local metadata storage engine with full userspace key-value store rather than the legacy EXT4 filesystem configured with BeeGFS, thus the performance of metadata service for Create, Stat, and Remove are improved by 14.86×, 3.61×, and 7.49× on average. However, because all metadata operations target a single metadata server under Single Dir workload, the metadata performance of the file system is limited by the performance of a single MDS and does not improve as the system scales. In contrast, CLMS-Hash can distribute metadata operations to all MDSs in the file system through hash functions, so parallel metadata services can be implemented and metadata performance can be greatly improved. Specifically, CLMS-Hash can further improve the OPS of file Create, Stat, and Remove operations by 1.6× when the number of MDS increases from 1 to 2, and by 10.48 when the number of MDS increases from 1 to 10. The metadata service scalability of CLMS-Hash outperforms BeeGFS and CLMS-Dir under the Single Dir workloads.

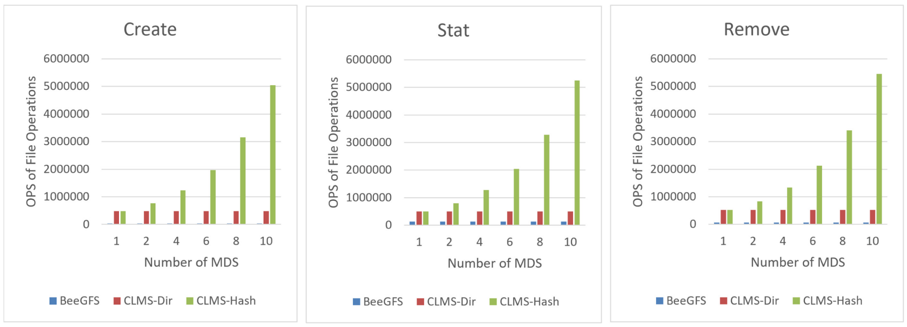

Fig. 6. OPS of File Operations under Single Dir Workloads

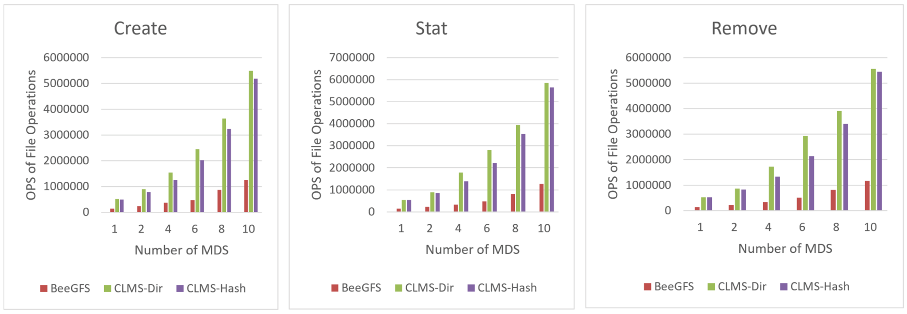

Fig. 7. OPS of File Operations under Unique Dir Workloads

We further compare BeeGFS, CLMS-Dir, and CLMS-Hash under the Unique Dir workloads that each process only accesses its directory. As shown in Fig. 7, different from the Single Dir workload, BeeGFS shows obvious metadata scalability as the number of MDS increases. Specifically, as the number of MDS increases from 1 to 10, the OPS of file Create, Stat, and Remove operations in BeeGFS get 8.82× improvements on average. This is mainly because the inherited directory-based metadata distribution strategy in BeeGFS achieves good metadata scalability. Compared with BeeGFS, CLMS-Dir achieves 4.21×, 4.75×, and 4.69× for Create, Stat, and Remove while CLMS-Hash achieves 3.72×, 4.22×, and 4.09× for Create, Stat, and Remove respectively. The metadata performance improvements are mainly from the efficient key-value store and POSIX semantic relaxation implementations in CLMS. Furthermore, as CLMS-Hash always distributes file and directory metadata across all the MDS, it inevitably incurs more inter-server networking communication overhead on the metadata access path. As a result, the metadata performance of CLMS-Hash is lower than that of CLMS-Dir under the Unique Dir workload. For file Create, Stat, and Remove operations, the OPS of CLMS-Dir is 11.59%, 9.94%, and 11.41% higher than that of CLMS-Hash.

7 Conclusion

This paper proposes CLMS, a configurable and lightweight metadata service for the parallel file systems on NVMe SSDs to meet the increasing performance requirements of the PFS metadata service. CLMS introduces a configurable metadata distribution policy that simultaneously enables the directory-based and hash-based metadata distribution strategies and can be activated according to the application I/O access pattern, thus improving the processing efficiency of metadata accesses from different kinds of data-intensive applications. CLMS further reduces the memory copy and serialization processing overhead in the I/O path through the full-user metadata service design. CLMS is comprehensively evaluated under the MDTest benchmarks. The experimental results demonstrate that CLMS can significantly improve the performance of metadata services and achieve a linear growth trend as the number of metadata servers increases.

Acknowledgements. This work was partially supported by the Foundation of National Key Research and Development Program of China under Grant 2021YFB 0300101, the Foundation of State Key Lab of High-Performance Computing under Grant 202101-09, and the Natural Science Foundation of NUDT under Grant ZK21-03.

References

1. IOR/mdtest (2020). https://github.com/hpc/ior
2. Amvrosiadis, G., Park, J.W., Ganger, G.R., Gibson, G.A., Baseman, E., DeBardeleben, N.: On the diversity of cluster workloads and its impact on research results. In: 2018 USENIX Annual Technical Conference (USENIX ATC 2018), pp. 533–546 (2018)
3. Chen, Y., Shu, J., Ou, J., Lu, Y.: HiNFS: a persistent memory file system with both buffering and direct-access. ACM Trans. Storage (ToS) **14**(1), 1–30 (2018)
4. Devarakonda, M.V., Mohindra, A., Simoneaux, J., Tetzlaff, W.H.: Evaluation of design alternatives for a cluster file system. In: USENIX, pp. 35–46 (1995)
5. Dorier, M., Antoniu, G., Ross, R., Kimpe, D., Ibrahim, S.: CALCioM: mitigating I/O interference in HPC systems through cross-application coordination. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp. 155–164. IEEE (2014)
6. Dulloor, S.R., et al.: System software for persistent memory. In: Proceedings of the Ninth European Conference on Computer Systems, pp. 1–15 (2014)
7. Hua, Y., Jiang, H., Zhu, Y., Feng, D., Tian, L.: SmartStore: a new metadata organization paradigm with semantic-awareness for next-generation file systems. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–12 (2009)
8. Kougkas, A., Devarajan, H., Sun, X.H.: Hermes: a heterogeneous-aware multi-tiered distributed I/O buffering system. In: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, pp. 219–230 (2018)
9. Lensing, P.H., Cortes, T., Hughes, J., Brinkmann, A.: 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), pp. 366–375. IEEE (2016)

10. Leung, A.W., Shao, M., Bisson, T., Pasupathy, S., Miller, E.L.: Spyglass: fast, scalable metadata search for large-scale storage systems. In: FAST, vol. 9, pp. 153–166 (2009)
11. Li, S., Lu, Y., Shu, J., Hu, Y., Li, T.: LocoFS: a loosely-coupled metadata service for distributed file systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2017)
12. Patil, S., Gibson, G.A.: Scale and concurrency of giga+: file system directories with millions of files. In: FAST, vol. 11, p. 13 (2011)
13. Ren, K., Zheng, Q., Patil, S., Gibson, G.: IndexFS: scaling file system metadata performance with stateless caching and bulk insertion. In: SC 20: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 237–248. IEEE (2014)
14. 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)
15. Schmuck, F.B., Haskin, R.L.: GPFS: a shared-disk file system for large computing clusters. In: FAST, vol. 2 (2002)
16. Sim, H., Kim, Y., Vazhkudai, S.S., Vallée, G.R., Lim, S.H., Butt, A.R.: TagIt: an integrated indexing and search service for file systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2017)
17. Thapaliya, S., Bangalore, P., Lofstead, J., Mohror, K., Moody, A.: Managing I/O interference in a shared burst buffer system. In: 2016 45th International Conference on Parallel Processing (ICPP), pp. 416–425. IEEE (2016)
18. Vef, M.A., et al.: GekkoFS-a temporary distributed file system for HPC applications. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 319–324. IEEE (2018)
19. Wang, T., Mohror, K., Moody, A., Sato, K., Yu, W.: An ephemeral burst-buffer file system for scientific applications. In: SC 2016: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 807–818. IEEE (2016)
20. Wang, T., Yu, W., Sato, K., Moody, A., Mohror, K.: BurstFS: a distributed burst buffer file system for scientific applications. Technical report, Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States) (2016)
21. Zheng, Q., et al.: DeltaFS: a scalable no-ground-truth filesystem for massively-parallel computing. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15 (2021)