An optimized storage method for small files in Ceph system

Chao Li
Department of Cloud Platform Research and Development
Inspur Cloud Information Technology Co.
Jinan, China

TengFei Wang
Department of Cloud Platform Research and Development
Inspur Cloud Information Technology Co.
Jinan, China

Lei Cao
Department of Cloud Platform Research and Development
Inspur Cloud Information Technology Co.
Jinan, China
* Corresponding author: cao.leilc@inspur.com

TaoTao Xie
Department of Cloud Platform Research and Development
Inspur Cloud Information Technology Co.
Jinan, China

2023 4th International Conference on Information Science, Parallel and Distributed Systems (ISPDS) | 979-8-3503-3718-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISPDS58840.2023.10235504

***Abstract*—In the scene of massive small files, there are a lot of data holes and redundant metadata in the distributed storage system based on Ceph, resulting in a waste of storage space. And the network consumption, PG(placement group) mutex and IO order preservation in the IO path will affect the IO performance. In order to solve the above problems, an online small file merging storage method is proposed. This method uses the relevance judgment module based on Crush (controlled replication under scalable hashing) to group the small files, and merges the small file data in the request context. When merging, the CV(condition variable) pool method is used to synchronize the threads in the same merging group. After merging, small files are stored sequentially and share the same metadata. In addition, this method optimizes the deletion of merged small files, and only records GC(garbage collection) logs when deleting, and garbage data is cleaned asynchronously by GC threads. The experimental results show that this method can improve the writing performance of small files by 62%, reduce data holes by 80% and reduce metadata by 50%.**

Keywords-distributed storage systems; Ceph; online small file merging method; CRUSH-based relevance judgment; CV pool; garbage data cleanup

I. INTRODUCTION

With the rapid development of Internet and storage hardware technology, a large amount of data is stored on various storage platforms [1]. Jim Gray pointed out that the growth of data conforms to the exponential model: the amount of data after 18 months is always twice the total amount of original data [2]. IDC predicts that the total data storage in the world will increase from 33 ZB in 2019 to 175ZB in 2025, with a compound annual growth rate of 61%. In order to store these data safely and efficiently, distributed storage systems such as HDFS[3], Ceph[4] and GFS[5] are designed. Ceph is widely used in cloud storage because of its high reliability and unlimited horizontal expansion. CRUSH[6], as the core algorithm of Ceph, supports dynamic mapping of data locations and avoids single-node failures.

In recent years, with the development of data-intensive business, such small files as user business data and experimental data are increasing day by day. The research of Environmental Molecular Sciences Laboratory [7] has also reached a similar conclusion. In a storage system with 12 million files, 94% of the files are no more than 64MB and 58% are no more than 64KB. In terms of data access, requests for small files also account for 90% of all data requests [8].

In the storage scenario of massive small files, the traditional Ceph distributed storage architecture has problems. First of all, loose data will produce data holes and high metadata redundancy, resulting in a waste of storage space. Secondly, the underlying storage of Ceph is usually based on mechanical disks, which have poor reading and writing performance for random small IO. In order to solve the LOSF(lots of small files) problem [9], there are two main methods: (1) Redesign the architecture of the distributed storage system [10], but the method of reconstructing the architecture is complex and difficult to apply to production, which easily leads to other performance problems. (2) Merging small files or adding a caching mechanism [11], but the existing merging strategy does not consider the relevance between small files. Moreover, many existing small file merging schemes read small files and merge them into large files asynchronously, and then write them back to the storage system. This solution has the problem of reading and writing amplification, which will impact the user's business, and frequent reading and writing will reduce the life of the disk. Many of the above two mainstream solutions lack the optimization of IO path for small file storage, which leads to the ineffective improvement of IO performance.

In order to solve the above problems, this paper proposes an optimized storage method for small files in Ceph system, which directly merges small file data in IO context, thus avoiding the impact on users' business. The merged data sequentially requests the underlying storage, which optimizes the IO path and reduces the network consumption from the storage gateway to the underlying storage. When merging small files, the relevance between small files is considered, and a relevance judgment module based on CURSH algorithm is

designed to improve the IO performance of small file storage. Because the deletion of small files after merging will cause data holes, this method designs a GC(garbage collection) thread pool to remerge small files. Finally, a 7-node Ceph storage cluster and a 3-node test cluster are built to verify this method. The experimental results show that this method can obviously improve the IO performance of small files and reduce data holes and metadata redundancy.

II. ONLINE SMALL FILE MERGING STORAGE METHOD

The online small file merging process is shown in Fig. 1, which mainly consists of two parts: relevance judgment module and online merging module. Relevance judgment module groups small files based on CRUSH (controlled replication under scalable hashing) algorithm, and online merging module merges small file data through CV (condition variable) pool method, while maintaining the merging status map, and finally writes the merged files into the underlying storage in sequence.

A. Relevance judgment module

The relevance judgment module groups small files based on CRUSH algorithm. According to the principle of time locality, small files with the same target disk are screened out from the small files uploaded in the same time period, and they are further split according to the small file merging threshold rule to get a new merge group.

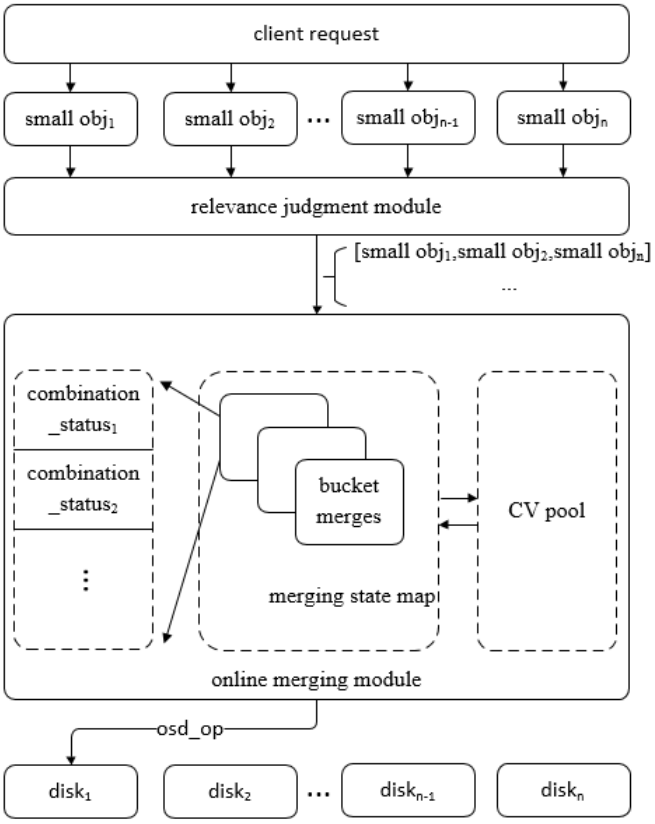

Figure 1. Online small file merging process

ALGORITHM I. CRUSH ALGORITHM

input: x
output: disk

PG id = Pool id + hash(x) % PG nums
for(iter: disks)
  draw = hash (PG id, disk id, replicate id)
  disk straw = draw * disk weight
disk = max(disk straws)

CRUSH algorithm is shown in algorithm 1, where x is the object name and disk is the target disk. Firstly, the PG(placement group) to which an object belongs is obtained, where PG represents a logical set of some objects, Pool represents a set of disks storing object data, and PG nums represents the count of PG contained in the Pool. Then all the disks are traversed, and the draw of the disk is obtained, and the draw is adjusted according to the disk weight, where replicate id uniquely identifies file's redundant copies, and straw represents the tendency of the object to write to the disk. The larger the value, the more the object tends to store data on the disk, and the hash function can get a pseudo-random fixed-length hexadecimal integer (pseudo-random: the same input gets the same output). Finally, select the disk with the longest straw as the target disk.

The small file merging threshold rule is designed as (1):

$$\begin{cases} combination\_size \le max\_combination\_size \\ combination\_num \le max\_combination\_num \\ time\_now - combination\_start \le \\ \qquad max\_combination\_duration \end{cases} \tag{1}$$

combination_size is the data size of the object after merging small files, combination_num is the count of merging small files, time_now is the upload time corresponding to the small file, combination_start is the start time of small file merging task, max_combination_size is the threshold of object data size after merging small files, max_combination_num is the threshold of merging small files, max_combination_duration is the duration threshold for merging task.

The rule of small file merging threshold shows that: in the same merge group, the size of small file merge data does not exceed the preset threshold, and the count of small files merged does not exceed the preset threshold, and the merging duration of small files does not exceed the preset threshold.

To sum up, after small files are grouped according to relevance, the target disks of small files in the same merging group are the same, and the merge threshold rules of small files are met.

B. Merging Status Map

After the small files are grouped, the merging status map records the status of each group's merging tasks, as shown in Fig. 2.

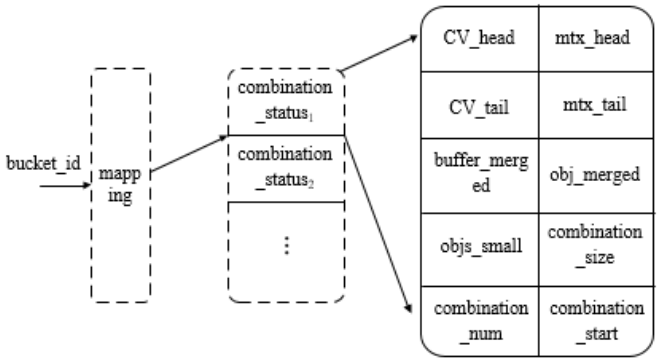

Figure 2. Merging status map

The key of the merging status map is bucket id, and the value is the list of combination_status which is the merging task status. The small file with the earliest upload time in the merge group is defined as merge header file, and the corresponding upload thread is merge header thread. Other small files in the merge group are merge tail files, and the corresponding upload thread is merge tail thread. In the merging task status, CV_head is the conditional variable of the merge head thread, and mtx_head is the mutex of the merge head thread, which is added to synchronize the merge head thread. CV_tail is the conditional variable of the merge tail thread, and mtx_tail is the mutex of the merge tail thread, which is added to synchronize the merge tail thread. obj_merged is the object name of the merged file, assigned as the object name of the merge header file, buffer_merged is the object data of the merged file, and objs_small is the list of all small file object names in the merge group.

The process of merging small files is as follows: first, the merge head thread updates the merge status map and obtains CV_head, locks mtx_head, sets the waiting timeout, and waits for CV_head. Then, all merge tail threads update the merge status map, lock mtx_tail and get CV_tail to wait, notify CV_head after the above operations are completed, and the merge head thread packages buffer_merged as osd_op (disk operation) and sends it to the target disk. Finally, the response of the merge head thread is returned to the client, and CV_tail is notified at the same time. After receiving the notification, the merge tail thread returns the response to the client.

C. CV Pool

In the online small file merging process, the synchronization of merge threads is completed by two conditional variables and two mutexes. Frequent allocation and release of variables in the global memory area will affect the IO performance of small file storage, and the control accuracy of release time is extremely high. If the release time is too early, the threads will be hung, and if it is too late, it will lead to segment errors.

Therefore, this paper proposes the CV pool method, which consists of a CV group list and a read-write lock. The CV group is defined as combination_sync, which consists of CV_head, CV_tail, mtx_head and mtx_tail. The CV pool locks the write lock before inserting and popping up mutexes and conditional variables, and locks the read lock before reading and querying mutexes and conditional variables. Write lock and read lock are not allowed when the write lock is in the locked status; Only read locks are allowed when the read lock is in the locked status.

When a merge thread obtains the condition variable and the mutex lock, it pops up a combination_sync from the CV pool. If the pop-up operation fails, combination_sync is allocated in the memory and used to synchronize the merge threads. After the merge, the combination_sync is pushed into the CV pool. At the end of the program, the conditional variables and mutexes in the CV pool are released.

III. GARBAGE DATA CLEANUP

When deleting small files after merge, the data on the disk is not deleted in the request context, but only the GC log is recorded, and the GC thread completes the cleaning of the above garbage data in a preset time period. The cleaning process is shown in Fig. 3.

A. GC Log

The GC log records the offset, length and flag of garbage data, and its index is bucket_id:obj_merged:instance_id:req_id. Where instance_id is the instance id of the storage gateway, and req_id is the upload request id of the merge header thread, this will ensure that the index is unique in the storage cluster. The range of flag is shown in (2):

$$ENMU\{\ GC\_initial = 0,\ GC\_processing,\ GC\_failed,\ GC\_complete\} \tag{2}$$

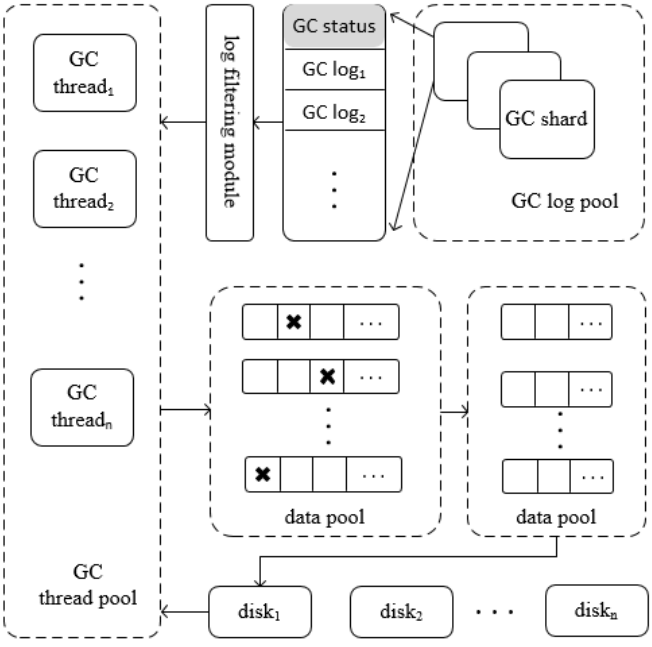

Figure 3. Garbage data cleanup

Where GC_initial is in initialization status, GC_processing is in cleaning status, GC_failed is cleaning failure status, and GC_complete is cleaning success status.

The GC log is recorded on the GC object. When the storage gateway is initialized, the GC object is divided into 64 shards. When recording GC logs, get the corresponding shard number after hash of bucket_id, so as to ensure that the GC logs of the same bucket are recorded on the same GC object shard.

B. Garbage Data Cleanup

The GC thread pool allocates GC threads in the preset GC working period, the process of GC thread is as follows:

a) randomly selects a GC object shard to read GC status, which records the start time of the last GC thread cleaning task, and the last GC log index processed, which is defined as marker.

b) If the last cleaning task occurred 24 hours ago, list all GC logs after the marker on the GC object shard, and set the GC flag of the above GC logs as GC_initial, and the cleaning task only processes GC logs in GC_initial status. If it is within 24 hours, skip the above process.

c) With marker as the index, read the GC log and check the GC flag. If the status is GC_processing, skip this cleaning and randomly select another GC object shard to perform the cleaning process again.

d) Read the next GC log of marker, parse the log index to get bucket_id and obj_merged and form a prefix, match all GC logs with GC_initial status according to this prefix, read the merged object data from the disk, delete the garbage data recorded in the above GC logs, write the remerged object back to the disk, and delete the above GC logs. If the above operation fails, update the flag of GC logs to GC_failed, and wait for the next garbage cleaning task to retry these GC logs.

IV. Experimental Results

Build a distributed storage cluster based on Ceph, which version is 16.2.11. Use cosbench to conduct stress test, and make statistics on IOPS performance, data hole rate and metadata amount of online small file merging method.

The Ceph cluster in the experimental environment includes 4 storage nodes and 3 management nodes, as shown in Tab. 1 and Tab. 2:

TABLE I. STORAGE NODE

parameter	Value
System disk	HDD
System	Ubuntu 18.04.6 LTS
Kernel	5.4.61-050461-generic
CPU	Intel (r) Xeon (r) Silver4214 CPU @ 2.20 GHz
Number of CPUs	48
CPU frequency	2700MHz
Memory	128GB
Disk	sata * 10+nvme * 1
Network card model	I350Gigabit * 4+82599ES10-Gigabit * 4
Network card bond	802.3ad

TABLE II. MANAGEMENT NODE

parameter	Value
System disk	HDD
System	Ubuntu 18.04.6 LTS
Kernel	5.4.61-050461-generic
CPU	Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
Number of CPUs	32
CPU frequency	1982MHz
Memory	64GB
Network card model	I350 Gigabit*4 + X710 for 10GbE*4
Network card bond	802.3ad

There are 10 OSD(object storage daemon) in each storage node, each with a capacity of 3.7TB, and 10 partitions with a capacity of 40GB are isolated from the NVME disk as DB(database) and WAL(write-ahead log) partitions of the osds. Each management node deploys a MON(monitor) to monitor the status of Ceph cluster; Randomly select a management node to deploy a storage gateway, and configure the chunk size to 4 MB. Storage_cluster and storage_public networks are networks with two 10-Gigabit ports as bond.

Build a cosbench cluster on three test nodes to simulate the client. The information of the test nodes is same as that in Tab. 2. Three cosbench worker are deployed in the test node, build the network with two 10-Gigabit ports as bond, and request the storage gateway through the bond network.

A. IOPS Performance

With 4KB as the file size, cosbench makes 1500 concurrent requests to request the storage gateway, and obtained the IOPS (number of IOs that can be processed per second) performance. The result is shown in Fig. 4.

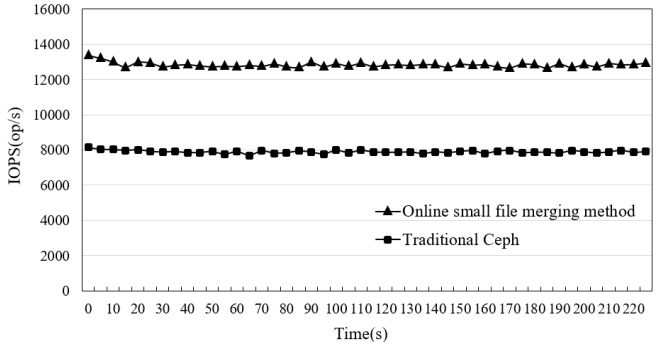

Figure 4. IOPS performance

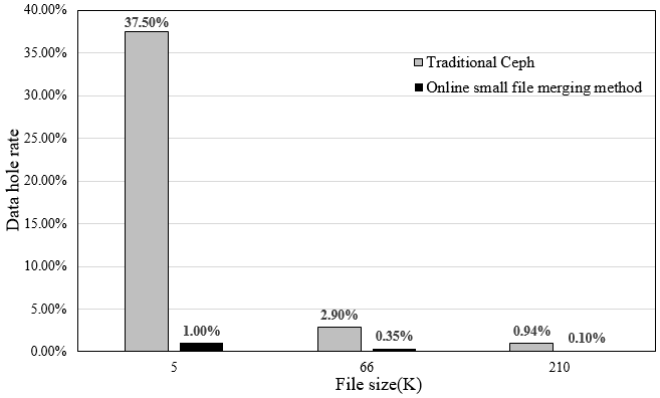

Figure 5. Data hole rate

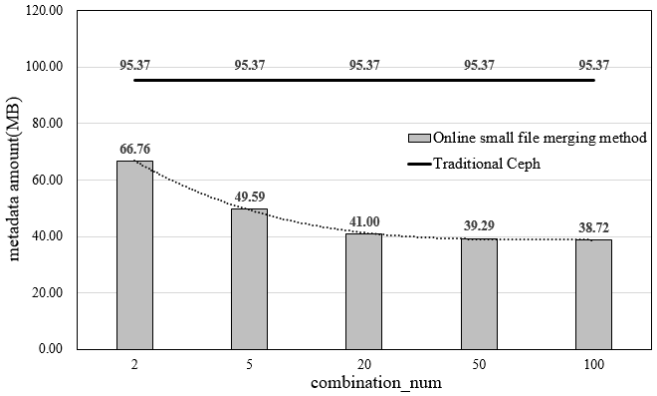

Figure 6. Metadata amount

B. Data Hole Rate

Because Ceph aligns the file data with a block size of 4KB, and the misaligned parts are filled with null values to the size of 4KB, which cause data holes in the disk, which leads to a waste of storage space. Request storage gateways with file sizes of 5KB, 66KB and 210KB, respectively. The data hole rate (the proportion of data holes in the used capacity) is shown in Fig. 5.

C. Metadata Amount

Upload 100,000 small files through the storage gateway. Count the metadata of small files when the combination_num is 2, 5, 20, 50 and 100 respectively, as shown in Fig. 6.

D. Analysis of the Results

Compared with the traditional Ceph scheme, the IOPS performance of online small file merging method is significantly improved, with an average increase of 62%. The data hole rate of online small file merging method decreased obviously, and it decreased by 97.3% when the file size was 5KB, and the effect weakened with the increase of file size. After merging small files, the amount of metadata is significantly reduced, and when the combination_num is 20, the effect tends to be stable, and the amount of metadata is reduced by 57% compared with the traditional Ceph scheme.

V. CONCLUSIONS

In order to solve the problems of poor IO performance and waste of data and metadata storage space in traditional Ceph scheme, an online small file merging method is proposed, which groups small files based on CURSH algorithm and merges them through CV pool in request context. Design GC thread to clean up garbage data asynchronously. From the experimental results, it can be seen that the above methods improve the IO performance of small files and the utilization of disk space. In the next step, based on the above methods, an optimization scheme is proposed for small file storage scenarios with loose time dimension.

REFERENCES

[1] V. Mayer-Schönberger, K. Cukier, “Big data: A revolution that will transform how we live, work, and think”, Houghton Mifflin Harcourt, 2013.

[2] J. Gray, “What next? A few remaining problems in Information Technology”, ACM Federated Research Computer Conference, 1999.

[3] P.R. Giri, G. Sharma, “Apache Hadoop Architecture, Applications, and Hadoop Distributed File System. Semiconductor Science and Information Devices”, vol. 4, pp. 14-20, 2022.

[4] S.A. Weil, S.A. Brandt, E.L. Miller, et al, “Ceph: A scalable, high-performance distributed file system”, Proceedings of the 7th symposium on Operating systems design and implementation, pp. 307-320, 2006.

[5] S. Ghemawat, H. Gobioff, S.T. Leung, “The Google file system”, Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 29-43, 2003.

[6] S.A. Weil, S.A. Brandt, E.L. Miller, et al, “CRUSH: Controlled, scalable, decentralized placement of replicated data” Proceedings of the 2006 ACM/IEEE conference on Supercomputing, pp. 122-, 2006.

[7] E. Felix, “Environmental molecular sciences laboratory: static survey of file system statistics”, http://www. pdsiscidac.org/fsstats/index.html, 2007.

[8] X. Li, B. Dong, L. Xiao, et al, “Adaptive tradeoff in metadata-based small file optimizations for a cluster file system”, International Journal of Numerical Analysis & Modeling, vol. 9, 2012.

[9] X. Li, B. Dong, L. Xiao, et al, “Small files problem in parallel file system”, 2011 International Conference on Network Computing and Information Security. IEEE, vol. 2, pp. 227-232, 2011.

[10] Q. Zhang, X. Pan, Y, Shen, et al, “A novel scalable architecture of cloud storage system for small files based on p2p”, 2012 IEEE International Conference on Cluster Computing Workshops. IEEE, pp. 41-47, 2012.

[11] B. Dong, J. Qiu, Q. Zheng, et al, “A novel approach to improving the efficiency of storing and accessing small files on hadoop: a case study by powerpoint files”, 2010 IEEE International Conference on Services Computing. IEEE, pp. 65-72, 2010.