A Cost-Efficient Metadata Scheme for High-Performance Deduplication Systems

Yuxuan Mo, Yu Hua*, Pengfei Li, Qin Cao, Xue Liu†
Huazhong University of Science and Technology
{*yuxuanmo, csyhua, cspfli, qcao*}*@hust.edu.cn*
†*McGill University*
xueliu@cs.mcgill.ca

***Abstract*—Data deduplication has been widely used in backup systems to eliminate redundant data, which speeds up the backup process and reduces the storage overhead. Deduplication packs multiple chunks into a large, fixed-size container as a storage unit to maintain the locality and achieve efficient compression. We observe that the traditional containers have low filling ratios due to a large amount of metadata generated by small files. Unfilled containers require more space to store a backup, which decreases the storage efficiency and reduces restore performance. In order to address this problem, we propose a Metadata region Adaptive Container Structure, called MACS. MACS maintains a tag to record the length of metadata region in the container. The boundary between metadata region and data region is dynamically decided to ensure the maximum space efficiency of the containers. Moreover, we propose a container metadata length-based indexing and cache replacement strategy to allow MACS to be practical in data backup systems. We demonstrate the advantages of MACS with three real world backup datasets. MACS achieves over 95% average container filling ratio, which is significantly higher than existing designs. MACS further achieves better restore performance than the traditional container structure. When combined with existing rewriting method, MACS achieves an efficient trade-off between deduplication ratio and restore performance.**

***Keywords*-Data Deduplication; Storage Efficiency; Restore Performance; Container Structure**

I. Introduction

In recent years, the digital data rapidly increase with the growth of mobile Internet, intelligent devices, scientific computing and other applications. The total amount of data will reach 163 ZB by 2025 according to the white paper released by International Data Corporation(IDC) [1]. Data deduplication becomes a significant technique to efficiently reduce and store massive data, which has been widely used in current backup storage systems.

In chunk-based deduplication systems, the storage efficiency is improved by identifying and eliminating redundant data. The data is divided into small-sized chunks (e.g., 4-8KB), which are represented as *fingerprints* (e.g., typical 20-byte) that are calculated via a hash algorithm, e.g., MD5, SHA1, SHA256 [2] [3]. Instead of comparing the data byte-by-byte, deduplication systems identify the duplicate chunks via fingerprints, which speeds up the comparison process. Two chunks share the same fingerprint are identified as duplicate chunks, and only one copy is stored on disks for storage efficiency.

To preserve the spatial locality of the backup data streams and improve the efficiency of read and write, multiple chunks are packed and stored into a large fixed-size (e.g., 4MB) *container* [4]. The container is the basic unit to read and write data in deduplication systems, which consists of a fixed length data region and a fixed length metadata region. However, we observe that existing schemes contain many unfilled containers, i.e., these containers contain a large amount of available space. The main reason is that the backup systems contain a tremendous amount of small files, while the container consists of metadata and data regions with fixed sizes. As a result, the metadata region is filled when the data region is partially empty. This lead to poor space utilization and backup system performance degradation.

Moreover, deduplication process influences the restore performance, due to only storing unique chunks but the subsequent duplicate chunks are not stored. The chunks of a backup are likely to be dispersed and stored in different containers. We have to load numerous different containers to restore the backup. With the growth of unfilled containers, the overhead to restore a backup increases.

To efficiently address above problems, we present a Metadata region Adaptive Container Structure, called MACS. It dynamically determines the boundary between the metadata region and data region according to chunk characteristics. The proportion of metadata region in MACS is not fixed and containers are flushed to disks with high filling ratio. Thus, MACS maximizes the utilization of storage space.

In the traditional container structure, the metadata are easily obtained according to the starting position of the container, since the metadata region has a fixed length. However, the metadata in MACS can't be directly obtained via the traditional approach due to the dynamic-size metadata regions. MACS adds a tag to indicate the length of metadata region in a container. Based on the observation that the adjacent backup versions are most similar, we just cache the length tags of last backup version. Moreover, we propose a metadata length-based cache replacement strategy for efficient caching.

* Yu Hua is the corresponding author.

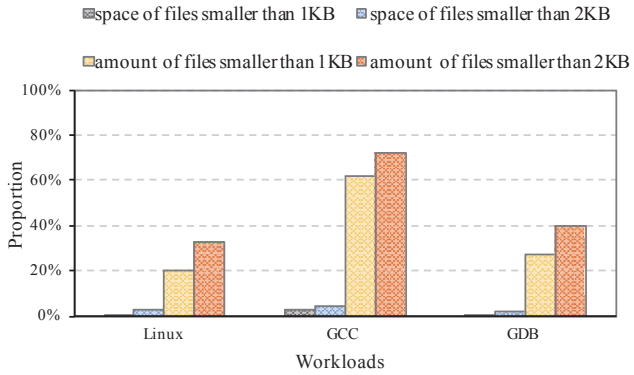


Figure 1. Space and quantity proportion of small files in the workloads.

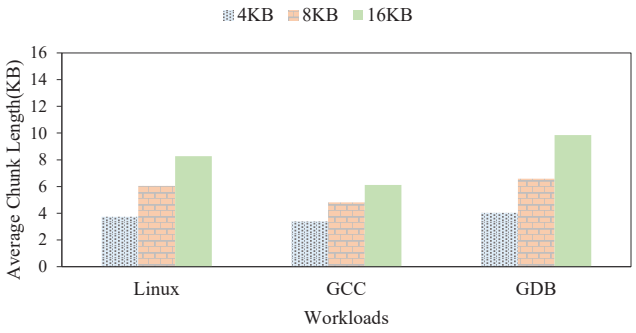


Figure 2. Actual average chunk length under different expected chunk lengths.

In this paper, we have the following contributions.

- We explore the workload characteristics in backup systems and propose a size-aware container structure to store the various-size chunks, which provides two types of containers to fully use the container space.
- To maintain the locality of the data streams and ensure the high space utilization, we optimize the proposed structure with adaptive metadata regions or MACS, which improves the storage efficiency.
- We propose a container metadata length-based indexing and metadata length-based cache replacement strategy to ensure the applicability of MACS in the data backup system.
- We evaluate our proposed scheme using three real-world datasets. The experimental results show that MACS achieves over 95% average container filling ratio and has the best restore performance compared to the commonly used design and SACS. When combined with existing rewriting method, MACS achieves an efficient trade-off between the deduplication ratio and restore performance.

II. Background and Motivation

A. Small Chunks in Deduplication

The deduplication systems typically divide the data stream into 4-8KB chunks. However, small files generate smaller chunks (e.g., 1KB), and the small files widely exist in real-world applications such as social networks, ecommerce sites, and digital libraries. Existing research shows that small files in a typical deduplication system occupies less than 20 % of the storage space, but more than 80 % of the total number of files [5].

To explore the impacts of small files on the chunk sizes, we use three datasets to evaluate the basic deduplication process on real workloads, including Linux, GCC, and GDB. The details of these datasets are described in Section IV. Figure 1 shows the occupied space ratio and quantity ratio of the small files in different workloads. We observe that although the proportion of the storage space occupied by the small files is low, the number of the small files is very large. In order to explore the effects of the small files, we use asymmetric extremum(AE) algorithm [6] to chunk the above three datasets. AE is a chunking scheme, which employs an asymmetric sliding window to identify extrema as cut points. Figure 2 shows the actual average chunk lengths when the chunking algorithm predefines different chunk sizes. We observe that all actual average chunk sizes are smaller than expected chunk sizes under different workloads. Due to the tremendous amount of small files in the workloads, there is a large proportion of small chunks in the data backup system, which result in the average chunk length to be lower than the expected chunk length. In addition, when the expected chunk length increases, more small files generate small chunks. Therefore, the gap between the actual average chunk length and the expected chunk length increases as the expected chunk size increases.

B. Low filling ratio of the container

The container is the basic unit to read and write in the deduplication systems, which groups multiple chunks together to preserve the locality. Specifically, a container consists of a data and metadata region. The data region stores the contents of chunks. The metadata region stores the description information of chunks, including the offset of the chunk and the size of the chunk, which are used to restore the original data. The container pre-allocates a fixed-size metadata and data regions with the assumption that all chunks have the predefined chunk size to simplify the process of container allocation and re-allocation. In theory, the utilization of storage space is maximized when the data region and the metadata region are filled at the same time. However, in practice, the two regions fail to be filled simultaneously. As shown in II-A, there are a large proportion of small chunks in the system, while the boundary between the metadata region and data region is set according to the expected chunk size in advance. As a result, the container is considered to be full when the metadata region is full, even if the data region remains a lot of available space.

To explore the efficiency of container space utilization,

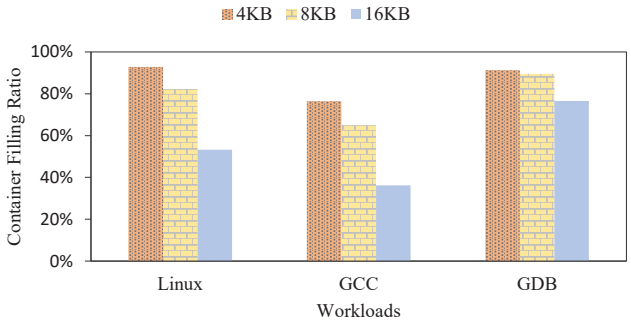


Figure 3. Average container filling ratio at different expected chunk lengths.

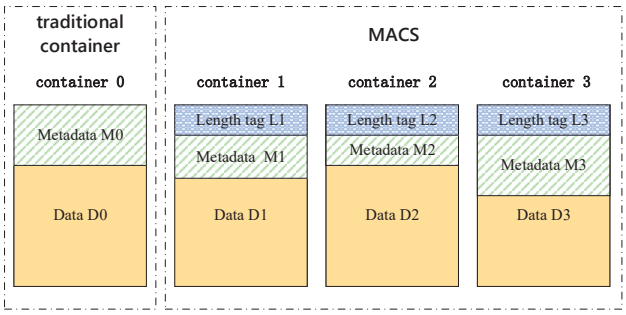


Figure 4. Traditional container structure and three examples of Metadata Region Adaptive Container Structure. i.e., *L1 = L2 = L3*.

we track the average filling ratio of the containers on disk after performing deduplication process. The average filling ratio is defined as the total amount of data written to disk divided by the amount of actually allocated physical space, which reflects the efficiency of physical space. In this evaluation, we use the same configurations for the containers via commonly used schemes and the evaluation results are shown in Figure 3. Each metadata entry occupies 28 bytes, including a 20-byte fingerprint, a 4-byte container offset and a 4-byte chunk length. The size of the metadata region is set to be equal to or slightly larger than the value of the expected metadata. The setting method and reason will be elaborated in III-A. Experimental results show that a large number of small files cause the metadata region to be filled before the data region, resulting in a waste of container space. More containers are required when containers have low filling ratio, which reduces the storage space efficiency. In addition, a low container filling ratio also decreases the restore performance of the system, since more containers are required to load to restore the original data.

III. Design and Implementation

In this section, we first propose a size-aware container structure to alleviate the metadata problem generated by small files. Then we propose metadata region adaptive container structure to further increase the container filling ratio. Moreover, we introduce the indexing process and cache replacement strategy when taking the MACS.

A. Size-aware Container Structure

Based on the observations in II, the number of chunks generated by small files is large, but the total space is small. In addition, compared with large files, small files are very unlikely to be changed. Therefore, in order to solve the problem of a large amount of container space being wasted in existing backup systems, we first propose a benchmark solution called Size-Aware Container Structure (SACS). Since the fixed-size metadata will cause the container to sacrifice storage space in its data region or metadata region, SACS adopts two kinds of containers with different metadata region sizes. SACS sets a chunk length threshold to divide chunks into two types, namely small chunks and large chunks. A chunk whose length is smaller than the threshold is called a small chunk, otherwise it is called a large chunk. At the same time, SACS maintains a dual container mechanism, namely special chunk containers and normal chunk containers. Small chunks are stored in special chunk containers, and large chunks are stored in normal chunk containers. The normal chunk container uses the same metadata region and data region length as the traditional container. The length of the metadata region in the special chunk container is larger than that in the normal chunk container.

For a special container, the size of metadata region is set to be equal to or slightly larger than that in the normal containers, as shown in Equation 1. S_{meta} indicates the size of metadata region, L_{entry} represents the length of metadata item, N_{chunk} represents the number of chunks in the container, and ϵ represents an integer equal to or larger than 0.

$$S_{meta} = L_{entry} * N_{chunk} + \epsilon \quad (1)$$

SACS uses the hash table to store the metadata in the container. When the metadata region becomes full while the data region remains much available space, SACS increases the size of metadata region according to Equation 2, where S_{meta} indicates the size of metadata region, L_{entry} represents the length of metadata item, and 2^k represents the capacity of the hash table.

$$S_{meta} = L_{entry} * 2^k \quad (2)$$

For example, when the small chunk size is 1KB and the container size is 4MB, a container will hold about 4096 chunks. In this case, the metadata region requires at least 112KB according to Equation 3, as the metadata item length of each chunk is 28 bytes.

$$S_{meta} = 28B * 2^{12} = 112KB \quad (3)$$

B. Metadata Region Adaptive Container Structure

Directly using SACS in existing deduplication systems decreases the restore performance because the chunk locality is destroyed. The locality in the deduplication systems refers

to the existence of similar or identical files between backup versions, e.g., The chunks divided by these files have a very high probability be re-accessed in the same order [4], [7]. However, SACS separately stores small chunks and large chunks, which destroys the locality between the small chunks and its adjacent large chunks. Moreover, SACS still has some containers with low filling ratios, since SACS still uses a fixed-length metadata region. The predefined threshold significantly affects the container utilization and fails to be set according to the workload characteristics.

To further improve the space utilization, we propose a metadata region adaptive container structure, called MACS, as shown in Figure 4. MACS uses dynamic boundary division to ensure that the metadata region and the data region are filled at the same time. MACS adds a 32-byte tag to record the length of the metadata region in the container. Compared with the storage space of the entire container, the proportion of the space occupied by the length tag is very small. By dynamically adjusting the sizes of the metadata and data regions, MACS improves the space utilization of the containers, as shown in Section IV. The space efficiency improvement is much greater than the space consumption of the length tag so the cost of length tag is worthwhile.

Two areas are defined in the container to store the data region and the metadata region respectively. When a chunk is identified as a unique chunk or a fragmented chunk, it will be written to the active container. MACS checks whether the newly added chunk incurs the overflow of the container according to Equation 4.

$$L_{data} + L_{chunk} + (N_{chunk} + 1) * L_{entry} > S_c - L_{flag} \quad (4)$$

N_{chunk} represents the number of chunks cached in the active container, L_{data} is the total length of these chunks, L_{chunk} represents the length of the chunk to be added, L_{entry} is the space occupied by the metadata entries, S_c indicates the size of the container, and L_{flag} is the space occupied by the length tag. MACS puts the new chunk into a new active container if the added chunk overflows the current container.

C. Metadata Length Based Indexing

When querying the fingerprint of a chunk, the system prefetches the metadata region of the found container into memory to leverage the locality. In subsequent queries, other fingerprints in this container hit the cache, thereby reducing the number of disk accesses. However, MACS can't directly read the containers via the traditional indexing scheme, since the sizes of the metadata regions are unfixed. One solution is to read the length tag before reading the metadata region. The weakness of this method is that it needs two times of disk I/O to read the metadata region of a container, resulting in high time overhead and performance degradation. Another solution is to read the entire container into memory using the container ID, which just need one disk I/O. However it wastes the limited memory space and bandwidth of I/O, as not all data of a data region are needed.

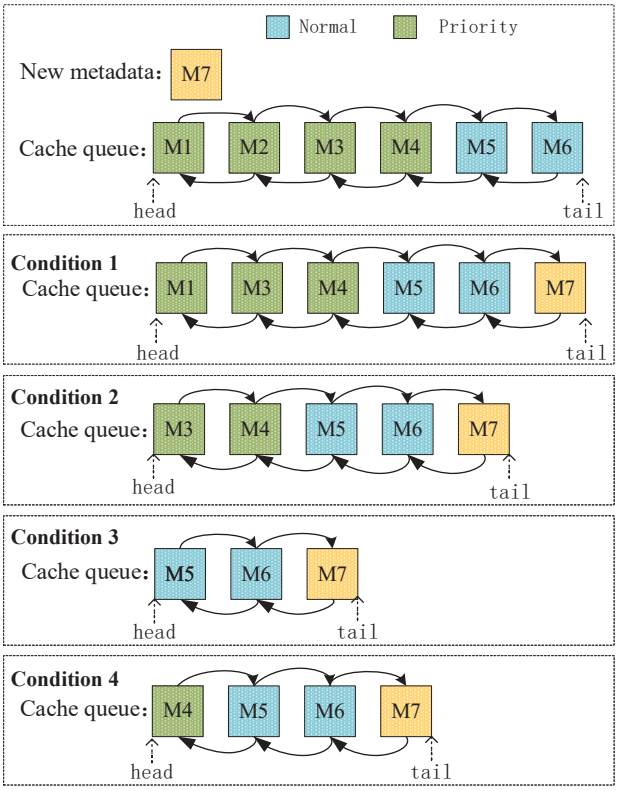
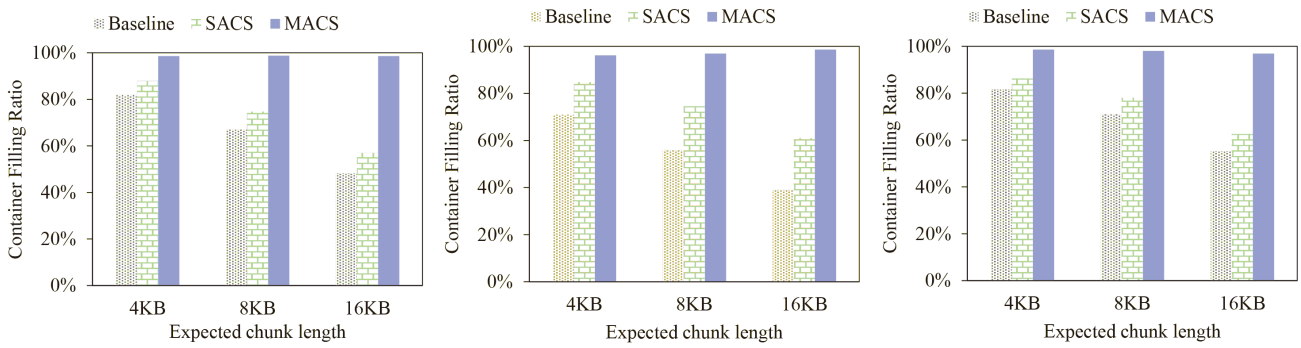


Figure 5. Examples of four conditions with metadata length-based cache replacement strategy. *Condition 1-3* correspond to the three listed priority replacement conditions, and *Condition 4* is the default.

Cache is commonly used hardware to reduce slowly disk I/Os. Caching all the lengths of all metadata regions in memory seems a feasible method to reduce disk I/O overhead. However this method can cause cache overflow as the amount of containers is very huge and grows with the increase of backup versions in backup systems. We observe that there is a high degree of similarity between adjacent backup versions. Instead, MACS only caches the length tags of the containers in last back version, which is affordable. Therefore, caching the container information of the last backup version reduces most of the disk I/Os. If the container ID of the duplicate chunk fails to hit the memory, MACS will read the entire container into memory and obtain the content of the metadata region according to the length tag. After the backup stream performs deduplication, the metadata region length index of the current version is retained in memory, and the information of the last backup version is cleared.

D. Cache Replacement Strategy

During the process of fingerprint query, the whole metadata region is prefetched into cache if the fingerprint hits



(a) Linux (b) GCC (c) GDB

Figure 6. Comparison results of average filling ratio of three container structures under different expected lengths.

the index table on the disk. An old container is removed from cache when the fingerprint cache is full. The least recently used(LRU) algorithm is a commonly used cache replacement algorithm, which removes the cache line that has not been accessed for the longest time. Directly applying the traditional LRU algorithm to MACS will not work as the chunks have various lengths.

The main idea of the cache replacement strategy based on metadata region length is to remove as few existing container metadata as possible and make full use of memory space when prefetching metadata region of a container to ensure that it can be put into cache. MACS divides the queue of the cache into a priority replacement part and a regular replacement part. The regular replacement part will only be used if all the conditions for priority replacement are not met. Suppose that the length of the metadata region to be added is S, the priority replacement part is traversed from the front to the end following three replacement conditions.

- Determine whether there is a metadata region with a length larger than S. If so, select the metadata with the closest length to S for replacement.
- Determine whether there are 2 metadata regions with a length larger than $S/2$ at the same time. If so, select the metadata with the closest length to $S/2$ for replacement.
- Determine whether there are 4 metadata regions larger than $S/4$ at the same time. If so, select the metadata with the closest length to $S/4$ for replacement.

If not meeting above replacement principles of minimum container metadata, MACS removes the metadata from the head of the queue until the remaining space in the cache is large enough to hold the new one. If all the container metadata of the priority replacement part is removed and the free part is still smaller than the newly added metadata, the metadata of the regular replacement part is sequentially removed from the back until the new metadata can be completely placed in the cache.

Figure 5 shows the examples of this cache replacement strategy. *M7* is the new metadata region to be added and its size it is S . The original replacement part of cache queue has four metadata regions and the regular replacement part has two. In Condition 1, the size of *M2* is larger than S and the closest length to S, meeting the first replacement principle; In Condition 2, *M1* and *M2* have the closest lengths to $S/2$, meeting the second replacement principle; In Condition 3, all metadata regions in priority part are larger than $S/4$ and smaller than $S/2$ at the same time, meeting the third replacement principle; In Condition 4, none of these replacement principles are met. It removes *M1*, *M2* and *M3* to make sufficient space for *M7* from the head of the queue. The metadata region length-based cache replacement strategy ensures the adaptability of MACS in the data backup system and makes better use of the limited cache space.

IV. Performance Evaluation

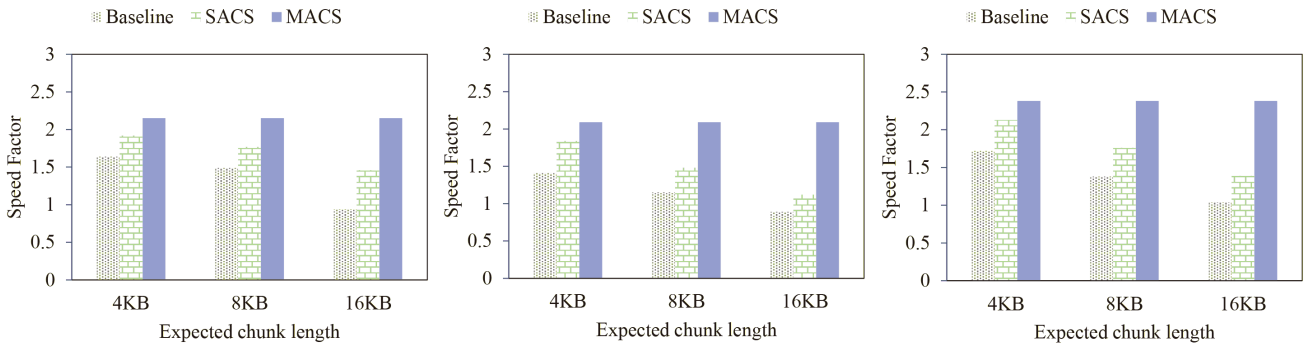
A. Experimental Setup

The experiments are deployed on a 4-core Intel E5620 2.4GHz system with 24GB memory and 1TB hard disk. We implement our prototype based on Destor [5], which is a widely used open-source deduplication system [8], [9]. For the configurations, we leverage AE algorithm [6] for chunking and SHA-1 hash algorithm [2] for hashing. Since fingerprint indexing performance is not the focus of this study, all the experiments put all fingerprint indexes in memory.

We use three widely used real workloads for performance evaluation, including Linux, GCC, and GDB [10], [11]. Linux dataset contains 50 consecutive versions of the Linux Kernel (linux-3.10.1 to linux-3.10.50), with a total size of 23.2GB. GCC dataset contains 42 consecutive versions of the GNU compiler (gcc-3.3 to gcc-4.4.7), with a total size of 11.5GB. GDB dataset contains 48 consecutive versions of the GNU debugger (gdb-5.2.1 to gdb-9.1), with a total size of 7.4GB. Since the tar format will affect the identification of duplicate data, this experiment uses fully decompressed workloads.

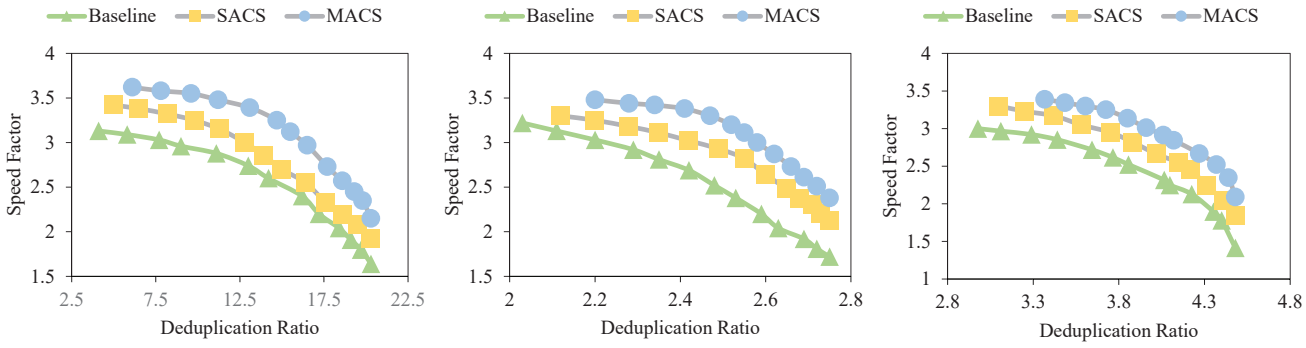
B. Space Efficiency

We use the average container filling ratio to evaluate the space efficiency of different schemes, which is defined as the



(a) Linux (b) GCC (c) GDB

Figure 7. Comparison results of restore performance of three container structures with different expected chunk lengths.



(a) Linux (b) GCC (c) GDB

Figure 8. Comparison results of deduplication ratio and restore performance of three container structures after integrated rewrite method.

actual stored data divided by the total size of the container. Figure 6 shows the evaluation results of MACS, which is compared with the baseline and SACS. The baseline refers to the container structure with a fixed-length data region and metadata region. SACS is a size-aware container structure as elaborated in Section III-A.

We set all containers to 4MB and configure different chunking sizes (i.e., the expected chunk lengths) in the AE algorithm. In SACS, this experiment defines small chunks as chunks smaller than 1KB. In this experiment, 4KB, 8KB, and 16KB are used as the expected chunk lengths.

From the experimental results, we observe that the average container filling ratio of SACS at different expected lengths is higher than that of the traditional container structure, and MACS achieves the highest container filling ratio among the three container structures. The reason is that SACS uses two sizes of container structures, and could reduce the number of container overflows caused by the full metadata region. The MACS dynamically determines the boundary between the data and metadata regions to ensure that both regions are filled at the same time. Therefore, even if MACS adds a length tag occupying 32 bytes of space in each container to record the length of the metadata region in the current container, its space efficiency is still highest one. MACS achieves over 95% average container filling ratio and gets the best performance.

C. Restore Performance

The restore process needs to read the data from different containers to assemble the original data. We use the widely used metric *speed factor* to measure the restore performance, which is defined as the mean data size read by one container [12]. The higher speed factor indicates the better restore performance.

The experiment uses 4KB, 8KB and 16KB as the expected chunk length to perform the basic deduplication process, and then recover the original data stream according to the recipe. The memory contains a container cache, set to 30 container sizes. The replacement strategy when the container cache overflows is the LRU algorithm. The container size is set to 4MB.

Figure 7 shows the restore performance of the three container structures. From the evaluation results, we observe that MACS achieves the highest restore performance among the different schemes, since MACS gathers the chunks into fewer containers and improves the physical locality of the data. As a result, MACS incurs fewer disk I/Os to read the containers, hence achieving higher restore performance than other schemes.

Existing deduplication systems selectively rewrite some chunks to improve the restore performance. In order to further verify the impact of different container structures on the restore performance, we equip the backup systems

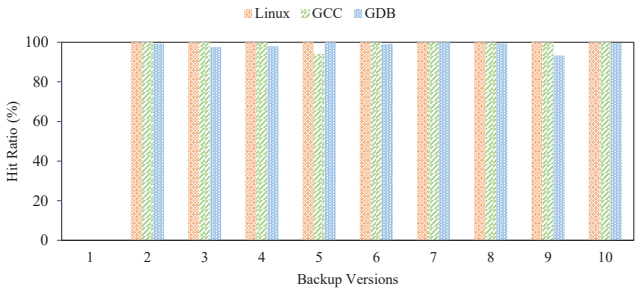


Figure 9. The metadata length-based index in MACS reduces the ratio of disk accesses.The hit ratio of backup version 1 is 0.

with rewriting algorithm LBW and observe the trend in the deduplication ratio and restore performance of the backup data stream. In this experiment, the sliding window size is 8 container lengths, the window moves forward by 1 container length each time, and the adjustment range of the rewriting threshold is 5 to 60.

In order to get the fair result, it is necessary to compare the speed factors of systems using these three container structures when the deduplication ratio is the same. Figure 8 shows the restore performance of different container structures when enabling the rewriting algorithm. From the evaluation results, we observe that MACS still achieves higher restore performance than other schemes, since MACS makes full use of the containers and improves the physical locality. Therefore, MACS can be combined with rewriting methods to improve the restore performance of the system.

D. The Caching Efficiency

In order to explore the efficiency of container metadata length-based index in reducing disk access, we evaluate the *hit ratio* to measure the caching efficiency. The hit ratio is defined as the number of the cached chunks divided by the total number of the duplicate chunks in the current backup version. The experiment selects the first 10 backup versions of three workloads and inputs them into the backup system with MACS. The expected chunk length is set to 4KB.

The result is shown in Figure 9. The hit ratio of backup version 1 is 0, since no data exist in the cache. For the other backup versions, the evaluation results show that the hit ratios are close to 100% due to the strong locality between adjacent backup versions. Hence, our caching strategy efficiently reduces the disk I/Os in backup systems.

V. Related Work

Improving the storage efficiency. As a small change in data stream can cause all subsequent chunks to be identified as unique, existing deduplication systems leverage the content-based chunking algorithms [13] to improve the deduplication ratio. It uses a sliding window to traverse the data stream. FastCDC [14] combines five key techniques, including simplifying and enhancing the Gear hash judgment, gear based fast rolling hash, skipping sub-minimum chunk cut-points, normalizing the chunk-size distribution, and rolling two bytes each time to further speed up CDC, which incredibly accelerates the chunking process.

The traditional compression algorithms [15] are used to further reduce the redundancy. Kruus and Romanski et al. improve the deduplication ratio by further chunking the unique chunks and re-executing the deduplication process [7], [16]. Moreover, incremental compression after deduplication eliminates more redundancy between unique but similar chunks without performing further deduplication [17]. Combining the compression and deduplication, BCD [18] effectively utilizes the partial matches among cache lines and achieves better compression ratio than the best art-of-state work.

Restore optimization. Designing efficient caching strategies and selectively rewriting duplicate chunks become two main solutions to improve the restore performance. Fu et al. [8] propose an optimal restore cache algorithm, called OPT, which caches containers during the restore process to improve the restore performance. Lillibridge et al. [12] propose a forward assembly scheme (FAA). Cao et al. [19] combine chunk-based caching and FAA to further improve the restore performance. The performance of cache schemes is relevant to block size, CDAC [20] leverages the content usage patterns to address this problem. CDAC achieves high read hit ratio, when the block size ranges from 4KB to 32 KB.

On the other hand, we can use rewrite technique based on different standards to improve the physical locality [8], [12], [21]–[24]. Cao et al. [23] use a rewriting method based on the look-back window, called LBW, which is proposed to identify the fragment using the sliding window. Lu et al. [25] propose an read-leveling data distribution scheme to improve read performance and reduce contention, which scatters the highly-duplicated data into different parallel units. Besides, Zou et al. [26] find that most duplicate chunks in a backup are directly from its previous backup and use a data classification approach to generate an optimal data layout.

Unlike the works we metioned above, our MACS improves the space efficiency by making full use of container space. Moreover, MACS can be coalesced with existing schemes to achieve higher storage efficiency.

VI. Conclusion

Data deduplication becomes an important technique to improve the storage efficiency in backup systems. However, our experimental results show that the containers are not fully loaded due to many small files. To address this problem, we propose a size-aware container structure with adaptive metadata regions, called MACS. Moreover, we optimize MACS with a container metadata length-based index scheme and a cache replacement strategy. Our evaluation results demonstrate that MACS significantly improves the storage

efficiency over traditional container structure and achieves high restore performance.

ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61772212.

REFERENCES

[1] D. Reinsel, J. Gantz, and J. Rydning, "Idc white paper: Data age 2025: The evolution of data to life-critical (idc white paper)," 2017.

[2] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage." in *USENIX Conference on File and Storage Technologies (FAST)*, vol. 2, 2002, pp. 89–101.

[3] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001, pp. 174–187.

[4] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *USENIX Conference on File and Storage Technologies (FAST 8)*, vol. 8, 2008, pp. 1–14.

[5] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, Y. Zhang, and Y. Tan, "Design tradeoffs for data deduplication performance in backup workloads," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 331–344.

[6] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, and Y. Zhou, "Ae: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1337–1345.

[7] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams." in *USENIX Conference on File and Storage Technologies (FAST 10)*, 2010, pp. 239–252.

[8] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, and Q. Liu, "Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information," in *USENIX Annual Technical Conference (ATC)*, 2014, pp. 181–192.

[9] J. Liu, Y. Chai, X. Qin, and Y. Xiao, "Plc-cache: Endurable ssd cache for deduplication-based primary storage," in *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2014, pp. 1–12.

[10] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput." in *USENIX Annual Technical Conference (ATC)*, 2011, pp. 26–30.

[11] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Generating realistic datasets for deduplication analysis," in *USENIX Annual Technical Conference (ATC)*, 2012, pp. 261–272.

[12] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 183–197.

[13] M. O. Rabin, "Fingerprinting by random polynomials," *Technical report*, 1981.

[14] W. Xia, X. Zou, H. Jiang, Y. Zhou, C. Liu, D. Feng, Y. Hua, Y. Hu, and Y. Zhang, "The design of fast content-defined chunking for data deduplication based storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, pp. 2017–2031, 2020.

[15] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[16] B. Zhou and J. Wen, "Hysteresis re-chunking based metadata harnessing deduplication of disk images," in *2013 42nd International Conference on Parallel Processing*. IEEE, 2013, pp. 389–398.

[17] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Y. Zhou, "Ddelta: A deduplication-inspired fast delta compression approach," *Performance Evaluation*, vol. 79, pp. 258–272, 2014.

[18] S. Park, I. Kang, Y. Moon, J. H. Ahn, and G. E. Suh, "Bcd deduplication: Effective memory compression using partial cache-line deduplication," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, p. 52–64.

[19] Z. Cao, H. Wen, F. Wu, and D. H. Du, "ALACC: Accelerating restore performance of data deduplication systems using adaptive look-ahead window assisted chunk caching," in *16th USENIX Conference on File and Storage Technologies (FAST 18)*, 2018, pp. 309–324.

[20] Y. Tan, C. Xu, J. Xie, Z. Yan, H. Jiang, W. Srisa-an, X. Chen, and D. Liu, "Improving the performance of deduplication-based storage cache via content-driven cache management methods," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 214–228, 2021.

[21] M. Kaczmarczyk, M. Barczynski, W. Kilian, and C. Dubnicki, "Reducing impact of data fragmentation caused by in-line deduplication," in *Proceedings of the 5th Annual International Systems and Storage Conference*, 2012.

[22] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "iDedup: latency-aware, inline data deduplication for primary storage." in *10th USENIX Conference on File and Storage Technologies (FAST 12)*, vol. 12, 2012, pp. 1–14.

[23] Z. Cao, S. Liu, F. Wu, G. Wang, B. Li, and D. H. Du, "Sliding look-back window assisted data chunk rewriting for improving deduplication restore performance," in *17th USENIX Conference on File and Storage Technologies (FAST 19)*, 2019, pp. 129–142.

[24] Y. Zhang, Y. Yuan, D. Feng, C. Wang, X. Wu, L. Yan, D. Pan, and S. Wang, "Improving restore performance for in-line backup system combining deduplication and delta compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, pp. 2302–2314, 2020.

[25] M. Lu, F. Wang, D. Feng, and Y. Hu, "A read-leveling data distribution scheme for promoting read performance in ssds with deduplication," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 22:1–22:10.

[26] X. Zou, J. Yuan, P. Shilane, W. Xia, H. Zhang, and X. Wang, "The dilemma between deduplication and locality: Can both be achieved?" in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 171–185.