Fast Variable-Grained Resemblance Data Deduplication For Cloud Storage

Xuming Ye [1, δ], Jia Tang [1, δ], Wenlong Tian [1, *], Ruixuan Li [2], Weijun Xiao [3], Yuqing Geng [1], and Zhiyong Xu [4, 5]

[1]School of Computer Science and Technology, University of South China, China
[2]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[3]Electrical and Computer Engineering, Virginia Commonwealth University, USA
[4]Math and Computer Science Department, Suffolk University, USA
[5]Shenzhen Institute of Advanced Technology, Chinese Academy of Science, China

***Abstract*—With the prevalence of cloud storage, data deduplication has been a widely used technology by removing cross users' duplicate data and saving network bandwidth. Nevertheless, traditional data deduplication hardly detects duplicate data among resemblance chunks. Currently, a resemblance data deduplication, called Finesse, has been proposed to detect and remove the duplicate data among similar chunks efficiently. However, we observe that the chunks following the similar chunk have a high chance of resembling data locality property, and vice versa. Processing these adjacent similar chunks in small average chunk size level increases the metadata, which deteriorates the deduplication system performance. Moreover, existing resemblance data deduplication schemes ignore the performance impact from metadata. Therefore, we propose a fast variable-grained resemblance data deduplication for cloud storage. It dynamically combines the adjacent resemblance chunks or unique chunks or breaks those chunks, located at the transition region between resemblance chunks and unique chunks. Finally, we implement a prototype and conduct a serial of experiments on real-world datasets. The results show that our method dramatically reduces the metadata size while achieving the high deduplication ratio.**

***Index Terms*—Resemblance Data Deduplication, Cloud Storage, Metadata Size, Variabe-Grained**

I. Introduction

With the development of the internet and electronic device, people prefer to outsource their data into cloud storage, such as Google Drive, Dropbox, and Baidu [1]. It achieves more flexibility and reliability for users' outsource data. Undoubtedly, there is much redundancy among the multi-user cloud storage scenario. These duplicate data significantly deteriorate the storage utilization and waste a part of network bandwidth. Thus, many researchers eliminate these redundant data during storage processing by introducing data deduplication technology.

Data deduplication detects duplicate data based on their hash values, such as Rabin Fingerprint values. The same data share one hash value. The deduplication system does not store duplicate data based on the duplicate data detection results. To explore more redundancy among these data, researchers break the large data into small chunks. For example, existing chunking algorithms evolve from fixed-size chunking into content-defined chunking algorithm such as BSW [2], TTTD [3], Elastic Chunking [4], and Fast CDC [5] . Thus, the smaller the chunk size it has, the more redundancy it achieves.

However, traditional data deduplication work only detects the completely duplicate data but fails to remove redundancy in resemblance data. Specifically, two chunks share a large part of the content with a few different bytes, leading to different hash values. Traditional data deduplication treats them as unique chunks because of different hash values. Obviously, the redundancy among similar chunks is hardly detected. Moreover, there are many redundancies in the multi-user storage scenario. Detecting and Removing these duplicate data in similar chunks can further save the network bandwidth and cloud storage space.

As the state-of-the-art work, N-transform [6] and Finesse [7] are proposed to detect the redundant data and use the delta compression to remove the redundancy part. Only the unique content is stored. In order to ensure a high similarity detection efficiency, N-transform generates the feature for each chunk. Furthermore, whether the chunk is a resemblance with others or not depends on the feature distance. The shorter the feature distance it has, the more similar the two chunks are. N-transform extracts all the Rabin fingerprints [8] of a chunk. All the Rabin fingerprint values are linearly transformed N times into N-dimensional hash sets. Then, the N maximal values, one from each of the N dimensions, are selected as features. Nevertheless, the N-transform suffers from the feature calculations caused by the linear transformation process.

To further speed up the resemblance detection, Finesse calculates the chunk features with a grouping strategy [7]. Specifically, it divides the chunk into sub-chunks and extracts their corresponding Rabin fingerprints into several contiguous sets of the same size. Then, these values construct the feature by grouping together based on each set's rank. The goal of Finesse is to achieve better performance than the N-transform method in resemblance detection. Although Finesse improves

δ These authors contributed to the work equally and should be regarded as co-first authors

$*$ Corresponding Author

978-1-7281-7744-1/21/$31.00 © 2021 IEEE

the performance by saving the transform process compared with N-transform, the existing resemblance detection scheme, include Finesse, ignores the metadata factor critical in deduplication performance.

Therefore, the metadata factor is critical in deduplication system performance based on previous research [9]. The smaller the average chunk size it is, the larger the metadata it generates. Although the small average chunk size could detect more duplicate data, it also produces much metadata. It greatly increases the data management difficulty such as the data fragmentation problem. Thus, the deduplication ratio should also consider the metadata factor. which is equal to $\frac{unprocessed\quad whole\quad storage\quad size}{processed\quad storage\quad size+metadata\quad size}$. Furthermore, the situation will be worse in resemblance data deduplication. The state-of-the-art resemblance detection work induces extensive computation and produces much more metadata, including the chunk feature, validation code, and chunk sequence information.

In summary, data deduplication is an essential technology in the cloud storage scenario. Removing the duplicate part among the resemblance data greatly saves the network bandwidth and improves cloud storage utilization. However, existing resemblance data deduplication schemes fail to consider the metadata factor in deduplication. Therefore, we propose a fast variable-grained resemblance data deduplication for cloud storage by considering the metadata factor in resemblance detection. The essential idea of our work is to avoid unnecessary resemblance detection computation and decrease the metadata size through variable-grained resemblance chunk detection. Thus, we split the data with a high probability of redundancy in fine-grained during the resemblance data deduplication while processing the data with a low probability of redundancy in coarse-granularity. The main contributions in this paper are summarized as follows:

- Firstly, we analyze the limitation in existing resemblance data deduplication schemes. The metadata size is critical in deduplication performance. Dynamically combining the adjacent resemblance chunks or unique chunks in resemblance data deduplication may significantly decrease the metadata size.
- Secondly, we propose a fast variable-grained resemblance data deduplication based on our observations. It avoids unnecessary resemblance detection computation and decreases the metadata size through variable-grained resemblance chunk detection.
- Finally, we conduct extensive simulations to evaluate our design. The experimental results show that our method outperforms the state-of-the-art work in metadata size and the deduplication ratio.

The rest of the paper is organized as follows. The related work about resemblance data deduplication is summarized in Section II. In Section III, we analyze the problem of the latest resemblance detection work in deduplication. Then, we propose a fast variable-grained resemblance data deduplication for cloud storage in Section IV. Finally, we present experimental results and conclude the paper in Section V and Section VI, respectively.

II. Related Work

With the prevalence of cloud storage technology, data deduplication becomes a critical technology to achieve high storage utilization and save network bandwidth. The deduplication technique benefits attract lots of researchers. In this section, we category the related work into two parts based on whether the deduplication supports the resemblance detection or not.

A. Traditional Deduplication

The main idea of Traditional deduplication is to remove the duplicate data by comparing the hash value for each chunk [10]. Only the unique chunk is stored. It was utilized into various cloud storage scenario such as primary and backup-level storage [11]–[14] to remove the redundancy. Other systems use different approaches to improve the effectiveness of data deduplication, such as integrating block-level deduplication with compression [15] and global deduplication method [16]. Moreover, data deduplication has also attracted considerable attention for virtual machine images [17]–[19]. However, based on our observation, there are lots of redundancy among non-duplicate but highly similar chunks. The existing traditional deduplication can hardly detect and remove the duplicate part among the resemblance chunks.

B. Resemblance Data Deduplication

To detect and remove the duplicate part among resemblance chunks, some researchers utilize delta compression, a data reduction technique, to maximize the compression ratio [6], [20]. It takes the delta algorithm with delta format to record the difference between similar chunks. Only the differences are recorded in delta files. Nevertheless, delta compression can hardly answer which chunks should be treated as the candidates for delta compression. Therefore, resemblance detection is the first step for delta compression to determine the resemblance among various chunks. Aronovich el at. [21] propose a novel type of similarity signatures serving in the deduplication system. It combines similarity matching schemes with byte by byte comparison or hash-based identity schemes. Xu el at. [22] propose a similarity-based deduplication system for database management systems by using byte-level encoding to achieve greater savings. There are also some other coarse-grained resemblance detection approaches [6], [23], [24]. These methods extract features from non-overlapped chunks and may suffer from high false positives.

As the latest work, such as N-transform [6] and Finesse [7], attempts to improve the resemblance detection accuracy and performance using the grouping features mechanism. Both of these methods are chunk-level resemblance detection. In N-transform, it extracts the top k largest Rabin fingerprint values of a chunk. Then a super-feature of this chunk can be calculated by several such features. Furthermore, Finesse extracts the largest Rabin fingerprint value in each subchunk and groups the Rabin fingerprint values as the features based on these values' ranking. However, the metadata factor is critical in deduplication system performance based on previous research [9]. The smaller the average chunk size is , the larger the metadata it generates. The situation will be worst when we consider resemblance detection in data deduplication. The state-of-the-art resemblance detection work induces extensive computation and produces much more metadata, including the chunk feature, validation code, and chunk sequence information. The existing resemblance detection method does not consider the metadata during the data deduplication processing.

III. Problem statement

In this part, we mainly discuss the limitations of existing resemblance data deduplication. To remove the redundancy among similar chunks, most researchers measure the similarity among chunks by extracting each chunk content's features. The most common feature extraction method is to select the top k largest Rabin fingerprint values in a chunk as the feature. Then, the similarity distance could be measured by these features. Only the different parts are stored according to the most similar pair of chunks.

However, the existing resemblance data deduplication scheme [6], [7] fails to consider the metadata factor, which may deteriorate the performance and increase the management complexity. The Metadata in a traditional deduplication system describes the basic chunk hash value and the relationship among chunks and the original data. The metadata size will be larger in resemblance data deduplication compared with the traditional one. Besides the traditional metadata elements, it also includes each chunk feature, a set of hash values. All the features should be recorded in the metadata. As the feature dimension rising, the metadata size will increase dramatically.

This increasing metadata seriously deteriorates the performance of the uploading and downloading processes in the deduplication system. Specifically, the metadata size is related to the chunk number in deduplication. To explore much more duplicate data, the fine-grained average chunk size is setted. The negative impact of this setting is increasing the metadata size and deteriorates the uploading performance even though it has the chunk container scheme. It is because that too many small chunks of uploading degrade the system's performance. Similarly, the recovery process in deduplication also needs to assemble these chunks based on metadata. The situation will be worsen in resemblance data detection. Moreover, the existing resemblance data deduplication schemes hardly consider the metadata factor.

Furthermore, the chunks following the similar chunk have a high chance to be resemble based on data locality property, and vice versa. What is more, if we defined two resemblance chunks' delta encoding result as a compression result, the compression results of each adjacent resemblance chunks are larger than the compression result that we treat these chunks as a whole. Moreover, treating these chunks at the fine-grained level without distinction also increases the metadata size. Thus, we suppose that dynamically combining the adjacent resemblance chunks or unique chunks in resemblance data deduplication may significantly decrease the metadata size. Furthermore, breaking those chunks, located at the transition region between resemblance chunk and unique chunk, is good for removing much more duplicate data. Inspired by this observation, we attempt to design a Fast variable-grained resemblance data deduplication scheme.

IV. System design

In this section, we first introduce the high-level workflow about our resemblance data deduplication design. Next, a fast variable-grained resemblance detection scheme is elaborated. It aims to decrease the metadata size and keep the high deduplication ratio. Finally, we briefly describe the implementation of our design.

A. the Overview of Workflow

Based on our observation, our design must avoid unnecessary resemblance detection computation and decrease the metadata size through variable-grained resemblance chunk detection. Thus, we first process the data slices in coarse-granularity with a low probability of duplicate data. Then, we further split the data slices with a high probability of redundancy in fine-grained. To simplify the description of our design, we introduce the workflow of our design in two processes: uploading and downloading, which is shown in Figure 1.

In the uploading process, the target uploading data is split into chunks based on coarse-grained average chunk size. Then, these chunks are categorized into three kinds of chunks, such as Transition region chunk, duplicate chunk, and non-duplicate chunk through the interaction with the server. It is because these chunks are in coarse-grained levels. The client can easily achieve these chunks' categories. After achieving this classified information, the client calculates the duplicate chunk ratio for the target data based on the divisor value between the duplicate chunk count and total chunk count. Then, the client decides whether further divide the whole chunks or part chunks into sub-chunks in fine-grained level. Finally, the client extracts the features of non-duplicate chunks or sub-chunks. Similar chunks are detected based on these features through the interaction with the server. Only the unique non-resemblance chunks or sub-chunks, delta file, and metadata are stored at the server-side.

When the client initiates a remote file request, the server first locates the file's corresponding metadata. Then, the related chunks, sub-chunks, and delta files are located based on metadata information. Next, the server recovers the resemblance chunks or sub-chunks based on the inverse operations of delta encoding. Then, based on the recovery sequence in metadata, the server put these related data together and sent the requested file to the client. It is noted that variable grained resemblance chunk detection greatly avoids unnecessary resemblance computation and significantly decreases the metadata size detailed in the following subsection.

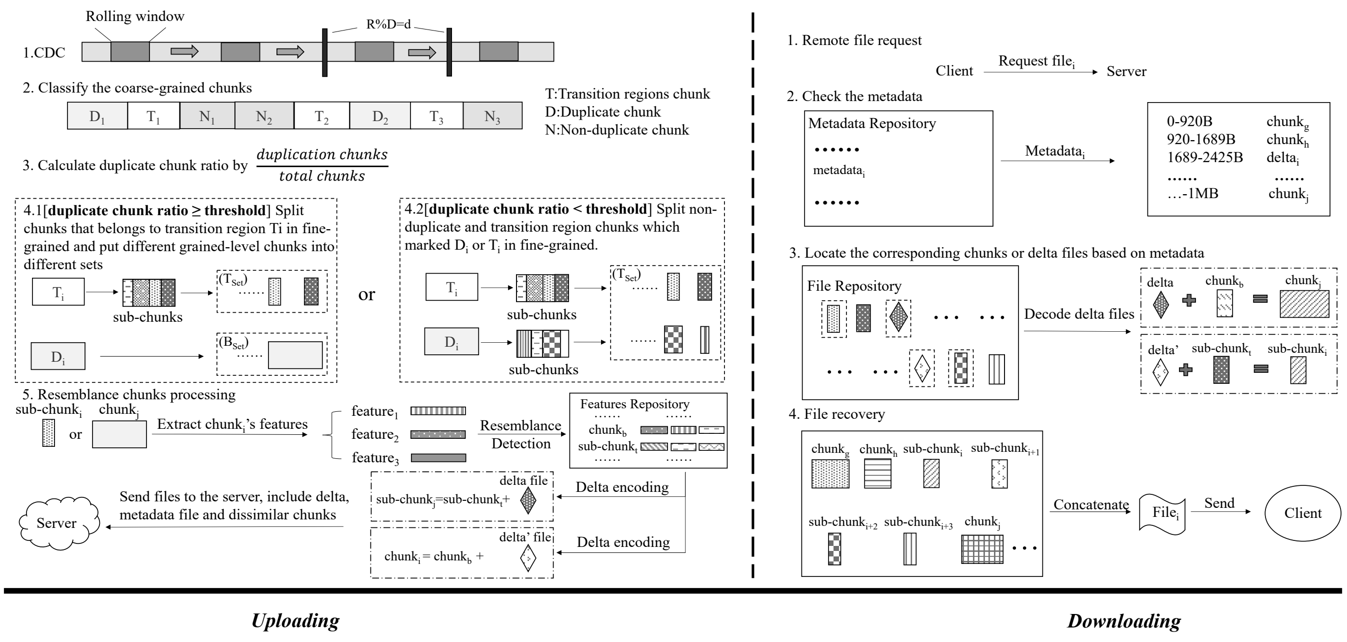
Rolling window
R%D=d
1.CDC
2. Classify the coarse-grained chunks
D1 T1 N1 N2 T2 D2 T3 N3
T:Transition regions chunk
D:Duplicate chunk
N:Non-duplicate chunk
3. Calculate duplicate chunk ratio by duplication chunks / total chunks
4.1[duplicate chunk ratio ≥ threshold] Split chunks that belongs to transition region Ti in fine-grained and put different grained-level chunks into different sets
Ti
sub-chunks
(TSet)
Di
(BSet)
or
4.2[duplicate chunk ratio < threshold] Split non-duplicate and transition region chunks which marked Di or Ti in fine-grained.
Ti
sub-chunks
(TSet)
Di
sub-chunks
5. Resemblance chunks processing
sub-chunki chunkj
or
Extract chunki's features
feature1
feature2
feature3
Resemblance Detection
Features Repository
chunkb
sub-chunkt
delta file
sub-chunkj=sub-chunkt+
Delta encoding
delta' file
chunki = chunkb +
Delta encoding
Send files to the server, include delta, metadata file and dissimilar chunks
Server
1. Remote file request
Client
Request filei
Server
2. Check the metadata
Metadata Repository
metadatai
Metadatai
0-920B chunkg
920-1689B chunkh
1689-2425B deltai
...-1MB chunkj
3. Locate the corresponding chunks or delta files based on metadata
File Repository
Decode delta files
delta chunkb chunkj
delta' sub-chunkt sub-chunki
4. File recovery
chunkg chunkh sub-chunki sub-chunki+1
sub-chunki+2 sub-chunki+3 chunkj
Concatenate
Filei
Send
Client
Uploading
Downloading

Fig. 1. The Overview of Fast Variable-Grained Resemblance Data Deduplication

B. Fast Variable-Grained Resemblance Detection Scheme

In this subsection, the fast variable-grained resemblance detection scheme is elaborated as shown in Algorithm 1. To dynamically detect the duplicate data in resemblance data detection, we introduce two average chunk size. One is the coarse-level average chunk size. The other is the fine-grained average chunk size. For each target data f_p, the fast variable-grained resemblance detection scheme split f_p into chunks based on the coarse-level average chunk size avg_chunk_size. Next, the client can determine whether each chunk is unique or not. Once the cloud server did not store the chunk's hash value, it will be treated as a unique chunk. Next, the client marks the duplicate chunk location based on these chunks' sequence.

Then, the client gets a series of coarse-level chunks in sequence. To faciliate the later fine-grained process, each unique chunk adjacent to the duplicate chunk is split into sub-chunks based on fine-grained average chunk size, $grain\_size$. We defined the sub-chunk set as T_Set. Furthermore, the unique chunk that does not adjacent with duplicate chunks is directly put into the B_Set set. Next, the client calculates each coarse-level chunk's hash value and marks the duplicate chunk's location. After the coarse-level chunking, the client statistics the duplicate chunks' proportion $init\_dup\_ratio$ in the target data.

The main reason for statistics is to accelerate the resemblance chunk detection. Once the proportion $init\_dup\_ratio$ is less than the threshold, which is a predefined value, the client will further process the whole unique chunks at a fine-grained level. Specifically, the whole unique chunks in B_{Set} need to be split into sub-chunks based on $grain\_size$ and put them into the T_{Set}. Finally, each element in T_{Set} and B_{Set} are calculate the chunk feature based on Finesse [7]. After receiving the chunk or sub-chunk feature, the server can select a similar chunk and record the different parts by delta encoding. The metadata in resemblance data deduplication includes the unique chunk or sub-chunk's features, chunk hash value, relationship information among chunks, sub-chunks, and the target data.

C. Implementation

In this subsection, we briefly describe the implementation of our design. We use the traditional content-defined chunking algorithm as the splitting algorithm. Each content-defined chunking algorithm has an average chunk size setting parameters. In our design, we take two kinds of average chunk size parameters. One is the coarse-grained average chunk size avg_chunk_size. The other is the fine-grained average chunk size $grain\_size$. It is noted that the threshold in Algorithm 1 is set as 0.01, which is an empirical value. We will further study it in our future work. In addition, the chunk feature extraction is the same with the Finesse [7]. Specifically, it extracts the top k largest Rabin fingerprint values of a chunk. Then a super-feature of this chunk can be calculated by several such features. The client extracts the largest Rabin fingerprint value in each subchunk and groups the Rabin fingerprint values as the features based on these values' ranking. Moreover, we use the VCDiff [25] as the delta encoding implementation.

V. Evaluation

A. DataSets

To evaluate the effectiveness of our method, we choose three distinct real-world workloads. The first one is the macOS workload [26], which collects the snapshots from various

Algorithm 1 Fast Variable-Grained Resemblance detection scheme

```
Input: $f_p$: target data path; $avg\_chunk\_size$: coarse-level average chunk size;$grain\_size$: fine-grained average chunk size;
Output: Delta files, metadata;
 1: split the target data $f_p$ into coarse-grained chunks, such as $ck_1, ck_2, \cdots, ck_i$, based on $avg\_chunk\_size$
 2: for $ck_1$ in $ck_i$ do
 3:     calculate each $ck_i$'s hash value and mark the duplicate chunks' location $\rightarrow$ Set L
 4: end for
 5: for each element in Set L do
 6:     if the i-th element is unique chunk and adjacent with duplicate chunk then
 7:         split the $ck_i$ into sub-chunks such as $sub\_ck_1, sub\_ck_2, \cdots, sub\_ck_j$,, based on $grain\_size$
 8:         Chunk Set $T_{Set} \leftarrow sub\_ck_j$
 9:     else if the i-th element is unique chunk and not adjacent with duplicate chunk then
10:         Chunk Set $B_{Set} \leftarrow ck_i$
11:     end if
12: end for
13: $init\_dup\_ratio = \frac{the \quad duplicate \quad chunk \quad count}{the \quad whole \quad chunk \quad count}$
14: if $init\_dup\_ratio < threshold$ then
15:     $T_{Set} \leftarrow$ split each element in $B_{Set}$ based on $grain\_size$
16:     clear all elements in $B_{Set}$
17: end if
18: for each element in $T_{Set} \& B_{Set}$ do
19:     calculate each element's feature and select out the similar chunks based on server's feature recording history through the vector distances
20:     do delta encoding and record $delta, metadata$ ;
21: end for
```

users on the same day. It represents the low deduplicate ratio workload. The second one is the Kernel dataset [27] comes from the Linux kernel archives website. The third one is the SQL Dump dataset including different periods backup and was reported by [28]. It is noted that these workloads contains various modification patterns from users' daily usage. Thus, we conduct following experiments on these workloads to fairly evaluate our design's benefits.

B. Metrics

We evaluate the efficiency of our method using two metrics: the Delta Compression Ratio(DCR) and the overall time (CPU execution time), where the DCR equals to $\frac{unprocessed \quad whole \quad storage \quad size}{processed \quad storage \quad size+metadata \quad size}$. We take the DCR as the metric to evaluate the benefits by considering the metadata factor. In addition, the speed performance is also critical in resemblance data deduplication. Thus, we also measure the the performance of our design by introducing the overall time cost metric. All the experiments are conducted on a clustered platform. The cloud is simulated by three machines. Each machine is equipped with an Intel(R) Core(TM) i7-4790 @3.60Ghz 8 core CPU, 16GB RAM, a 500GB 5400rpm hard disk and is connected to a 100Mbps network. The OS installed is Ubuntu 18.10 LTS 64-bit System.

C. Numerical Results and Discussions

First of all, as shown in Figure 2, the DCR in our design are 3.71%, 4.68%, 1.64%, higher than Finesse, respectively, when the average chunk size is 2KB, 4KB, 16KB. Our design uses the principle of locality of similar data (that is, it is more likely that similar data surround similar data). The data is divided into granularity processing, focusing on the most likely to contain similar data. Under the condition of ensuring the efficiency of deduplication, reduce Metadata expense caused by the increase in chunks. Finesse makes use of a grouping strategy, scanning chunks without considering metadata and dividing them according to the content, then extracting the feature vector of the chunk, judging similar data, and performing data deduplication. Due to the excessive number of sub-chunks, the size of metadata also increases.

In contrast, the DCR of Finese will be low, and during the recovery of files, metadata detection will increase, which is not conducive to files' recovery. At the same time, we can find that as the chunk length goes from 4K to 8K, the DCR gap between our method and finesse gradually decreases. When the chunk length is 16K, the DCR of finesse is almost as same as that of our method. As the chunk length increases, a file may be processed as a single chunk, and the locality principle of the chunk does not play its role. However, all chunks in Finesse are at a fine-grained level. In addition, as the chunk length increases, the metadata expense gradually decreases. Therefore, in general, the difference in DCR between Finesse and our method gradually decreases as the chunk length increases.

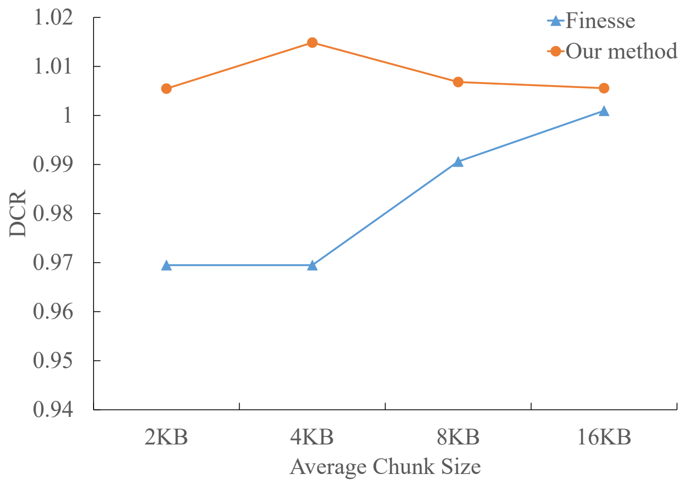

Fig. 2. The DCR in MacOS Workload

There are few similar data in the Kernel dataset, and it is difficult to divide the chunk by the principle of data locality, so the DCR of our method and Finesse are not very good. At the same time, most of the files in the Kernel tarred code dataset are less than 16KB. Most files may be treated as a single chunk and cannot be divided into variable granularity, then causes the gap of DCR between Finesse and our method to reduce gradually. As shown in Figure 3, when the chunk size is 16K, our method and Finesse gap is 9.52% . Figure 3 shows that the DCR of our method is 17.85%, 16.95%, and 13.29%, higher than those of Finesse from 2KB to 8KB. Although as the chunk length increases, the advantages of our method in the locality principle gradually decrease, and the gap with Finesse gradually decreases. Overall, our method's efficiency in data deduplication is still higher than that of Finesse.

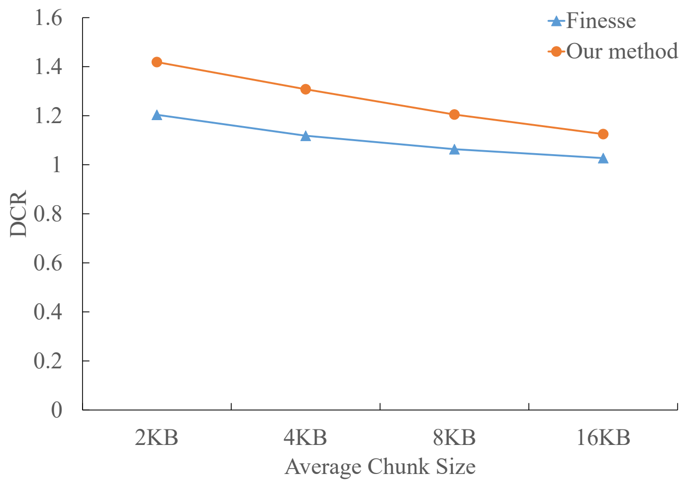

Fig. 3. The DCR in Linux Kernel Workload

There is a large amount of redundant data in the SQL Dump dataset due to the dump workload. Therefore, when the average chunk length is small, the same content will generate more incremental files, occupying more storage space. So as the chunk length increases, the DCR of Finesse gradually increases, while the DCR of our method decreases. The probability of redundant chunks gradually decreases after the chunk length increases, so that it is difficult to divide the data with variable granularity. Most of the data is in a coarse-grained manner. After processing, the accuracy of data deduplication gradually decreases. Although the DCR of our method is gradually decreasing and Finesse is gradually increasing, the DCR of our method is still higher than that of Finesse. The variable granularity processing can effectively reduce the number of chunks, reduce the expense of metadata, and increase DCR. Compared with our method, the effect of our method is better.

The overall time cost in the Mac dataset is shown in Figure 5. When the chunk size is 2KB, the speed of our method is 11.07 times that of Finesse, mainly because Finesse needs to fingerprint the data in the chunk in units of chunks. The code performs operations such as grouping and sorting. Finally, the feature vector is obtained. Therefore, the time consumption cost is higher under the small chunk condition. While our

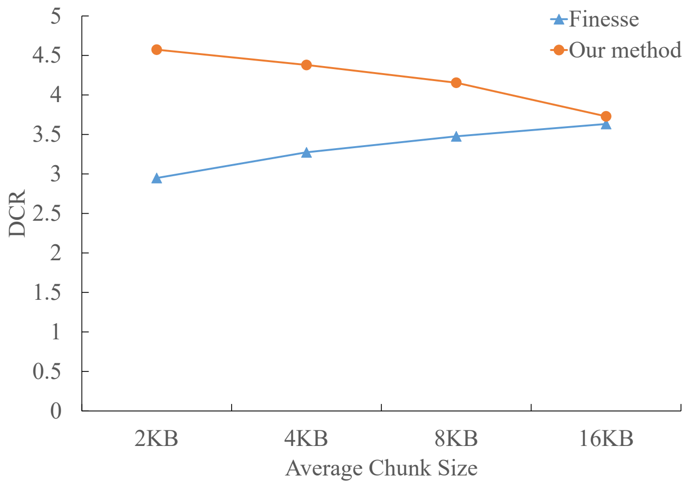

Fig. 4. The DCR in SQL Dump Workload

method performs variable granularity processing on chunks, reduces the number of chunks, reduces metadata overhead, and focuses on calculating chunks that are most likely to contain redundant data. When the chunk length is from 4KB to 16KB, the time overhead of Finesse is 4.43, 3.42, 2.29, and 1.15 times of our method, respectively. It can be seen that as the chunk length increases, the advantage of our method gradually decreases, but it is still faster than Finesse. As the chunk length increases, the proportion of redundant data in each chunk gradually decreases. Our method needs to perform fine-grained processing of most chunks to ensure the accuracy of data deduplication, so the number of chunks increases, and the cost of time gradually becomes the same as the Finesse. Nevertheless, on the whole, as the chunk length increases, the number of chunks decreases in the two methods, and the detection time cost is gradually reduced. However, the time cost of our method is lower under comparison.

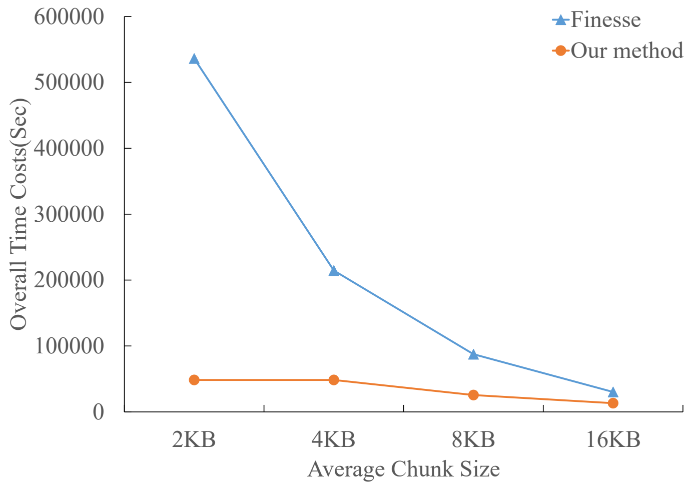

Fig. 5. The Overall Time Cost in MacOS Workload

In the kernel dataset, as shown in Figure 6, when the chunk size is from 2KB to 8KB, the time consumption of our-method

method is lower than that of Finesse, and it is maintained at about 1/5 of Finesse. In contrast, Finesse needs to perform a fine-grained scan of all data and obtain feature vectors for similarity detection through methods such as grouping and sorting, which are time-consuming. So our method is faster than Finesse. However, as mentioned above, when the chunk length increases, the proportion of redundant data in each chunk gradually decreases, and most of the data needs to be fine-grained processing. The speed of our method gradually becomes as same as Finesse. So as shown in Figure 6, when the chunk length reaches 16KB, the time consumption of our method and Finesse is similar, and may even be the same.

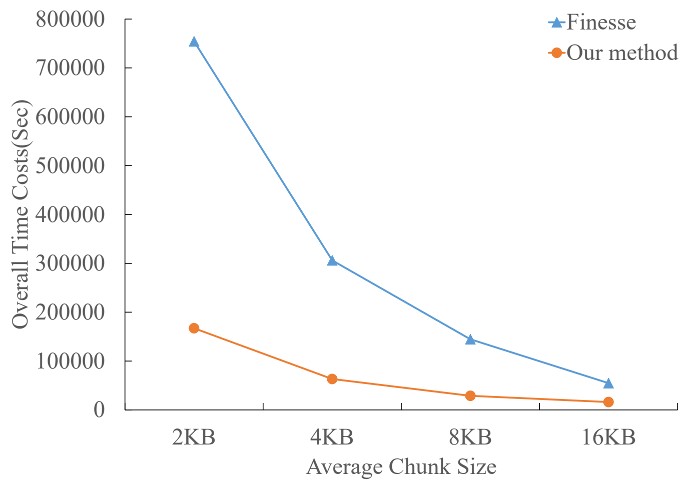

Fig. 6. The Overall Time Cost in Linux Kernel Workload

In the SQL Dump dataset, as shown in Figure 7, in changing the chunk size from 2KB to 16KB, the speed of our method and Finesse both is reduced due to the decreasing number of chunks obtained. So the time consumed in the entire data deduplication process is also reduced. However, in this process, the time consumption of Finesse is still higher than our method, which is 4.31, 2.86, 1.94, and 1.33 times slower than our method, respectively. It can be seen that when chunks are small,variable-granularity chunks used by our method have the potential to reduce time consumption. However, as the average chunk size increases, our method outperforms Finesse.

VI. Conclusions

With the prevalence of cloud storage, data deduplication has been a widely used technology by removing cross users' duplicate data and saving network bandwidth. The paper summarizes the state-of-the-art resemblance data deduplication and analyzes their limitations based on our observation. Specifically, the chunks following the similar chunk have a high chance of resembling data locality property and vice versa. Treating these chunks at the fine-grained level without distinction also increases the metadata size. Moreover, existing resemblance data deduplication schemes ignore the performance impact from metadata size. Thus, we propose a fast variable-grained resemblance data deduplication for cloud storage. It dynamically combines the adjacent resemblance chunks or unique chunks into a large chunk to decrease the metadata while keeping the chunks located at the transition region between resemblance chunk and unique chunk for a high deduplication ratio. Finally, we implement a prototype and conduct a serial of experiments on real-world datasets. The results show that our method dramatically reduces the metadata size while achieving the high deduplication ratio.

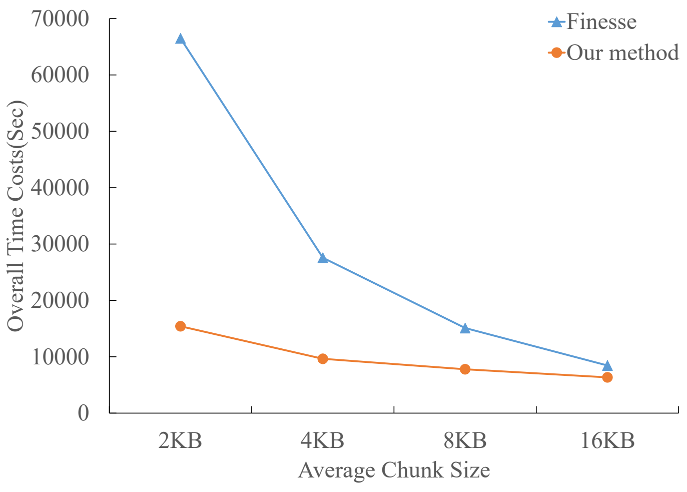

Fig. 7. The Overall Time Cost in SQL Dump Workload

Acknowledgment

This work is supported by the National Key Research and Development Program of China under grants 2016YFB0800402, National Natural Science Foundation of China under grants U1836204, U1936108, University of South China University research foundation under grant 190XQD121, Teaching Reform Foundation under grant 2019YB-XJG30, Natural Science Foundation of Hunan Province of China under grant 2021JJ40468, Hunan Province research foundation under grant 19C1624, and the Ministry of Education Humanities and Foundation on Humanities and Social Sciences under grant 20YJC880027. This work is also partly supported by the National Science Foundation under grant CNS 1526190.

References

[1] P. Sharma, R. Jindal, and D. B. Malaya, "Blockchain technology for cloud storage: A systematic literature review," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 89:1–89:32, 2020.

[2] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the 18th ACM Symposium on Operating System Principles, SOSP 2001, Chateau Lake Louise, Banff, Alberta, Canada, October 21-24, 2001*, K. Marzullo and M. Satyanarayanan, Eds. ACM, 2001, pp. 174–187.

[3] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," *Hewlett-Packard Labs Technical Report TR*, vol. 30, no. 2005, 2005.

[4] W. Tian, R. Li, Z. Xu, and W. Xiao, "Does the content defined chunking really solve the local boundary shift problem?" in *36th IEEE International Performance Computing and Communications Conference, IPCCC 2017, San Diego, CA, USA, December 10-12, 2017.* IEEE Computer Society, 2017, pp. 1–8.

[5] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Q. Liu, and Y. Zhang, "Fastcdc: a fast and efficient content-defined chunking approach for data deduplication," in *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*, A. Gulati and H. Weatherspoon, Eds. USENIX Association, 2016, pp. 101–114.

[6] P. Shilane, G. Wallace, M. Huang, and W. Hsu, "Delta compressed and deduplicated storage using stream-informed locality," in *4th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage'12, Boston, MA, USA, June 13-14, 2012*, R. Rangaswami, Ed. USENIX Association, 2012.

[7] Y. Zhang, W. Xia, D. Feng, H. Jiang, Y. Hua, and Q. Wang, "Finesse: Fine-grained feature locality based fast resemblance detection for post-deduplication delta compression," in *17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019*, A. Merchant and H. Weatherspoon, Eds. USENIX Association, 2019, pp. 121–128.

[8] M. Rabin, *Fingerprinting by Random Polynomials*, ser. Center for Research in Computing Technology: Center for Research in Computing Technology. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[9] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 23-26, 2010*, R. C. Burns and K. Keeton, Eds. USENIX, 2010, pp. 239–252.

[10] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized deduplication in SAN cluster file systems," in *2009 USENIX Annual Technical Conference, San Diego, CA, USA, June 14-19, 2009*, G. M. Voelker and A. Wolman, Eds. USENIX Association, 2009.

[11] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *Proceedings of the 11th USENIX conference on File and Storage Technologies, FAST 2013, San Jose, CA, USA, February 12-15, 2013*, K. A. Smith and Y. Zhou, Eds. USENIX, 2013, pp. 183–198.

[12] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *7th USENIX Conference on File and Storage Technologies, February 24-27, 2009, San Francisco, CA, USA. Proceedings*, M. I. Seltzer and R. Wheeler, Eds. USENIX, 2009, pp. 111–123.

[13] M. Lu, D. D. Chambliss, J. S. Glider, and C. Constantinescu, "Insights for data reduction in primary storage: a practical analysis," in *The 5th Annual International Systems and Storage Conference, SYSTOR '12, Haifa, Israel, June 4-6, 2012*. ACM, 2012, p. 17.

[14] 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 *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*, G. Gibson and N. Zeldovich, Eds. USENIX Association, 2014, pp. 181–192.

[15] A. Upadhyay, P. R. Balihalli, S. Ivaturi, and S. Rao, "Deduplication and compression techniques in cloud design," in *Systems Conference*, 2012.

[16] M. Oh, S. Park, J. Yoon, S. Kim, K. W. Lee, S. Weil, H. Y. Yeom, and M. Jung, "Design of global data deduplication for a scale-out distributed storage system," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[17] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proceedings of of SYSTOR 2009: The Israeli Experimental Systems Conference 2009, Haifa, Israel, May 4-6, 2009*, ser. ACM International Conference Proceeding Series, M. Allalouf, M. Factor, and D. G. Feitelson, Eds. ACM, 2009, p. 7.

[18] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "id-edup: latency-aware, inline data deduplication for primary storage," in *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*, W. J. Bolosky and J. Flinn, Eds. USENIX Association, 2012, p. 24.

[19] N. Zhao, H. Albahar, S. Abraham, K. Chen, V. Tarasov, D. Skourtis, L. Rupprecht, A. Anwar, and A. R. Butt, "Duphunter: Flexible high-performance deduplication for docker registries," in *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, A. Gavrilovska and E. Zadok, Eds. USENIX Association, 2020, pp. 769–783.

[20] P. Shilane, M. Huang, G. Wallace, and W. Hsu, "WAN optimized replication of backup datasets using stream-informed delta compression," in *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*, W. J. Bolosky and J. Flinn, Eds. USENIX Association, 2012, p. 5.

[21] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. T. Klein, "The design of a similarity based deduplication system," in *Proceedings of of SYSTOR 2009: The Israeli Experimental Systems Conference 2009, Haifa, Israel, May 4-6, 2009*, ser. ACM International Conference Proceeding Series, M. Allalouf, M. Factor, and D. G. Feitelson, Eds. ACM, 2009, p. 6.

[22] L. Xu, A. Pavlo, S. Sengupta, and G. R. Ganger, "Online deduplication for databases," in *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu, Eds. ACM, 2017, pp. 1355–1368.

[23] F. Douglis and A. Iyengar, "Application-specific delta-encoding via resemblance detection," in *Proceedings of the General Track: 2003 USENIX Annual Technical Conference, June 9-14, 2003, San Antonio, Texas, USA*. USENIX, 2003, pp. 113–126.

[24] G. Forman, K. Eshghi, and S. Chiocchetti, "Finding similar files in large document repositories," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, R. Grossman, R. J. Bayardo, and K. P. Bennett, Eds. ACM, 2005, pp. 394–400.

[25] J. L. Bentley and M. D. McIlroy, "Data compression using long common strings," in *Data Compression Conference, DCC 1999, Snowbird, Utah, USA, March 29-31, 1999*. IEEE Computer Society, 1999, pp. 287–295.

[26] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Generating realistic datasets for deduplication analysis," in *2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012*, G. Heiser and W. C. Hsieh, Eds. USENIX Association, 2012, pp. 261–272.

[27] W. Tian, R. Li, W. Xiao, and Z. Xu, "Pts-dep: A high-performance two-party secure deduplication for cloud storage," in *20th IEEE International Conference on High Performance Computing and Communications; 16th IEEE International Conference on Smart City; 4th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2018, Exeter, United Kingdom, June 28-30, 2018*. IEEE, 2018, pp. 700–707.

[28] M. Beller, G. Gousios, and A. Zaidman, "Travistorrent: synthesizing travis CI and github for full-stack research on continuous integration," in *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, J. M. González-Barahona, A. Hindle, and L. Tan, Eds. IEEE Computer Society, 2017, pp. 447–450.