A Near-Exact Defragmentation Scheme to Improve Restore Performance for Cloud Backup Systems

Rongyu Lai, Yu Hua, Dan Feng, Wen Xia, Min Fu, and Yifan Yang

Wuhan National Laboratory for Optoelectronics (WNLO)
Huazhong University of Science and Technology, Wuhan, China
{lairongyu,csyhua,dfeng,xia,fumin}@hust.edu.cn,dickyfyang@gmail.com

Abstract. Cloud backup systems leverage data deduplication to remove duplicate chunks that are shared by many versions. The duplicate chunks are replaced with the references to old chunks via deduplication, instead of being uploaded to the cloud. The consecutive chunks in backup streams are actually stored dispersedly in several *segments* (the storage unit in the cloud), which results in fragmentation for restore. The segments that are referred will be downloaded from the cloud when the users want to restore the chunks of the latest version, and some chunks that are not referred will be downloaded together, thus jeopardizing the restore performance. In order to address this problem, we propose *a near-exact defragmentation scheme*, called NED, for deduplication based cloud backups. The idea behind NED is to compute the ratio of the length of chunks referred by current data stream in a segment to the segment length. If the ratio is smaller than a threshold, the chunks in the data stream that refer to the segment will be labeled as fragments and written to new segments. By efficiently identifying fragmented chunks, NED significantly reduces the number of segments for restore with slight decrease of deduplication ratio. Experiment results based on real-world datasets demonstrate that NED effectively improves the restore performance by 6%∼105% at the cost of 0.1%∼6.5% decrease in terms of deduplication ratio.

1 Introduction

Cloud backup is an important application of cloud storage service. Stefan et al. [9] proposed a data backup system, called Cumulus. Cumulus is a cloud backup system with high portability by only using four basic cloud storage interfaces (GET, PUT, LIST, DELETE) to manage data in the cloud. YuruBackup is a cloud backup system that uses fingerprint servers to obtain highly-efficient deduplication and explores a highly scalable architecture for fingerprint servers to cope with increasing number of clients [12]. Dropbox is a file synchronization tool based on Amazon S3 and it is one of the world's most popular cloud storage applications. In addition, there are many other cloud backup systems, such as Jungle Disk, Brackup.

X.-h. Sun et al. (Eds.): ICA3PP 2014, Part I, LNCS 8630, pp. 457–471, 2014.

Some cloud backup systems are designed under a *thin cloud* assumption that the remote data center only provides minimal interfaces (i.e, uploading and downloading complete files). Cumulus [9], Brackup, YuruBackup [12] and Duplicity are this kind of system. The *thin cloud* design ensures that the systems are able to back up data to almost any remote storage. We focus on the restore performance of thin cloud based backup systems in this paper.

In order to improve the backup speed and reduce the storage space, the backup systems employ data deduplication [9], [12] and delta compression [11] due to their salient features of data compression performance. Only deduplication is discussed in this paper. In the backup process, the input chunks are duplicate detected, and duplicate chunks will not be written to the cloud. Instead, the system only keeps references to the stored chunks. Through deduplication, only new chunks will be written to segments and uploaded to the cloud. Therefore, deduplication improves storage utilization and saves backup time for thin cloud based backup systems.

In practice, the deduplication possibly causes degradation of restore performance. Deduplication removes duplicate chunks and keeps references to old chunks stored in the cloud. The deduplication leads to logically consecutive chunks in a data stream not being consecutively stored in *segments* (the storage unit in the cloud). Some referred segments contain lots of chunks that are not referred by the data stream. If the length of chunks referred by a data stream in a segment is smaller than a threshold, the chunks of the data stream that refer to the chunks of the segment are fragments. To restore these fragments, the segments that contain them will be downloaded. Chunks that are not referred in these segments will be downloaded together. But these chunks are useless for the restore and lengthen the restore time, since the segments are downloaded through WAN and the speed of segments reading is much faster than that of segments downloading. The bottleneck in restore is the segments downloading process. Thus the fragments exacerbate the restore performance.

Several defragmentation schemes were proposed to improve restore performance of deduplication based backup systems, such as Capping [5], CFL [6] and CBR [4]. However, they are designed for defragmentation in deduplication-based traditional backup systems. After defragmentation, every duplicate chunk obtains a defragment state, "fragment" or "not fragment". Existing schemes can not guarantee that the duplicate chunks that refer to one segment are in the same state. Some duplicate chunks referring to the chunks of one old segment are possibly determined as fragments and rewritten to new segment. Meanwhile, others are not fragments and keep reference to chunks of the old segment. For restoration, both new segments and the old segments will be downloaded. The fragments will be downloaded twice. Existing schemes mistakenly identify some duplicate chunks to be fragments, and it incurs two main challenges: (1) cloud storage space will be wasted and thus the backup cost will be increased; (2) the backup time will be significantly extended in a low-bandwidth network environment.

In this paper, we propose *a near-exact defragmentation scheme for deduplication based cloud backup (NED)*. It computes the ratio of length of chunks referred by current data stream in a segment to the segment length. If the ratio is smaller than a threshold, the chunks in the data stream that refer to the segment will be labeled as fragments. Thus, the duplicate chunks that refer to a segment will get the same defragment state. If a segment is referred by a fragment, the segment will not be downloaded when the data stream is restored. No chunks will be downloaded twice. Compared with existing defragmentation schemes, NED is able to rewrite fewer chunks than existing schemes while achieving near-exact restore performance of cloud backup system without deduplication. In addition, we need to upload smaller amount of data to the cloud than existing defragmentation, which helps save backup time and storage space.

The rest of this paper is organized as follows. Section 2 presents the previous work of improving restore performance of deduplication based backup system. The background and motivation of our work are proposed in section 3. In section 4, we present design and implementation of NED. Section 5 presents and discusses the experiment results of NED. Finally, section 6 concludes the paper.

2 Related Work

Researchers have proposed some schemes to improve restore performance in deduplication based storage systems. These schemes are divided into two categories according to the principles, *"defragmentation"* and *"deduplication"*. iDedup [7], CFL [6], CBR [4], and Capping [5] are *"defragmentation"* schemes that aim to identify and rewrite the fragments. CABdedup [8] is a *"deduplication"* scheme, and it employs data deduplication in the restore process.

iDedup is a deduplicaiton solution for primary workloads which are latency-sensitive. iDedup efficiently reduces the latency of deduplication, but it is not appropriate for cloud backup because it rewrites too much data and cloud backup workloads are not latency-sensitive in the deduplication stage.

CFL-SD selectively deduplicates the input chunks based on chunk fragment level (CFL) to improve restore performance. CBR rewrites the fragmented chunks based on stream context and disk context to improve restore performance. Capping improves restore performance by analyzing the fragmented chunks in a much larger buffer of input chunks. Specifically, Capping inserts chunks to a fixed-length (for example, 20MB) buffer, and computes the count of segments referred by the chunks in the buffer. Capping only deduplicates the chunks in the top 10 referred segments and rewrites other chunks for defragmentation.

CABdedup points out that deduplication not only can be applied to improve backup speed and save storage space in the backup process, but also can be used in the restore process. However, it doesn't address the fragmentation problem caused by deduplication.

Some cache techniques on chip-multiprocessors [15], [14], [13] are used to speed up the fingerprint computing, which helps improve the backup speed. However, the restore doesn't consume much computing time, and the techniques don't have obvious effect on restore.

Table 1. Table of related work. This table shows how *NED* is positioned relative with some other relevant work.

Name	Usage Scenario	Design Goal & Solution
iDedup [7]	Primary storage	It executes selective deduplication to improve restore performance of deduplication system
CFL-SD [6]	Traditional backup	It executes defragmentation based on chunk fragment level to improve restore performance of deduplication system
CBR [4]		It executes defragmentation based on stream context and disk context to improve restore performance of deduplication system
Capping [5]		It executes defragmentation based on buffer analysis to improve restore performance of deduplication system
CABdedupe [8]	Cloud storage	It executes deduplication in the restore process to improve restore performance of deduplication system
NED		It executes defragmentation based on segment reference analysis to improve restore performance of deduplication system

Table 1 summarizes the above restore performance optimization schemes. There is not a *"defragmentation"* scheme for cloud backup. We analyze the reason in detail in section 3. Therefore, we propose NED to remove fragments in deduplication based cloud backup systems.

3 Background and Motivation

3.1 Cloud Backup

Some nomenclatures are explained below to help describe how cloud backup system works clearly.

Data set and data stream. A data set includes many versions of file collection. Files in a collection are divided into chunks, and the chunks form a data stream. For example, chunks A $\sim$ L form a data stream in Figure 2.

Segment. Segment is the storage unit in the cloud [9], [8], [12]. Cloud backup systems write chunks into larger units called segments. The chunk that makes a segment be longer than the specified maximum segment (4MB default) will be written to a new segment, and the previous segment will be closed. A segment is identified by a segment ID.

To get high portability, the cloud backup systems only employ simple interfaces (e.g, get, put, delete, list etc). They don't depend on the ability to read or write arbitrary byte ranges within a file. The interfaces are simple enough that they can be implemented on various protocols, such as Amazon S3, FTP, SFTP and network file systems [9]. Thus, a complete segment is uploaded and downloaded.

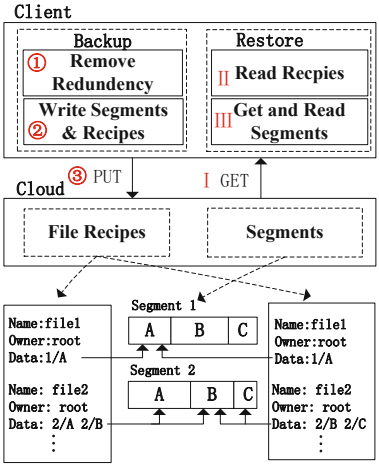


Fig. 1. The backup and restore process

Recipe. Each job has a recipe. The main contents of a recipe are the list of files that are backed up in the job and the chunks' addresses of the files. In addition, it contains job information, such as backup time. Figure 1 shows two simplified recipes. Recipes are written for recovery jobs. The client gets the addresses of chunks from the recipe and gets segments from remote storage according to the addresses.

Backup process. The left part of Figure 1 shows the 3 steps of backup. The first step is removing redundancy which includes dividing files into chunks, computing fingerprints and searching fingerprints. This step is the same as that of traditional backup system (we treat backup system that doesn't use cloud storage as traditional backup system). The fingerprint searching is done in the client or in the metadata server [10]. The system writes new chunks to segments and writes chunk addresses to the recipe in the second step. In the third step, the recipe and segments are uploaded to the cloud.

Restore process. The right part of Figure 1 shows the restore process of the cloud backup system. The restore process includes 3 steps. First, the client downloads the recipe of the job from the cloud. Second, the chunk addresses (segment ID and chunk ID) are read from the recipe. Third, segments are downloaded from the cloud according to the addresses and chunks are read from the segments to construct the files. However, a segment is possibly referred by multiple files, and repetitive downloading of segments severely slows the restore speed when the system restores different files. Therefore, the client employs a buffer on the disk to save the segments that are already downloaded. When a segment is needed, the client checks the buffer. If the segment already exists in the buffer, the client reads it directly from the buffer. Otherwise, the client downloads the segment from the cloud and then inserts it to the buffer. Thus, a segment will be downloaded from the cloud only once.

Table 2. The comparison between cloud backup and traditional backup

	Cloud Backup	Traditional Backup
General User	Individual	Enterprise
Size of Backup Set	Small	Big
Bandwidth	Low	Hight
Storage Location	Cloud	Data Server
Storage Unit	Segment	Container
The Bottleneck of Restore	WAN	Disk
Representative Systems	Cumulus[9], Dropbox Jungle Disk, Brackup duplicity, YuruBackup[12]	HydraStor[1], DDFS[16] Symantec[3] GreenBytes

Table 2 summarizes the comparison between cloud backup and traditional backup. In traditional backup, chunks are sent to the client after being read from the containers in the data servers in restore process[2]. If a data server is running multiple jobs, reading data from disk will become a bottleneck in traditional backup. Since thin cloud system employs interface of reading a complete file (i.e., segment)[9], data is read in the client after the segments are downloaded from the cloud in cloud backup [12]. In general, the speed of reading data from disk is much faster than that of translating data through WAN. Therefore, the bottleneck of restore process in cloud backup is the segment downloading process.

3.2 Definition of Fragment in Cloud Backup System

Chunks are shared by new version and old version after deduplication. The system replaces duplicate chunks with the references to old chunks instead of uploading the chunks to the cloud. The consecutive chunks in backup stream are actually stored dispersedly. The lengths of referred chunks in some segments are possibly longer than those of other segments. In order to accurately measure the length of data referred by the data stream in a segment, we introduce two new terms, *segment reference length (SRL)* and *segment reference ratio (SRR)*. SRL is the total length of data referred by a data stream in a segment, and SRR is the ratio of SRL to the length of the segment. Due to the different distributions of data streams, the SRR of a segment differs in different data streams. If a duplicate chunk is restored in cloud backup system, the complete segment that is referred must be downloaded. Some chunks that are not referred by any chunks of the data stream will possibly be downloaded together.

Segment reference ratio threshold is proposed to help the system identify fragments. If a duplicate chunk refers to a segment whose reference ratio is smaller than the segment reference ratio threshold, the chunk is a fragment. For example, we assume that the threshold is 0.3. In Figure 2, the reference ratio of segment 37 is 0.75, and it is bigger than the threshold. Therefore,

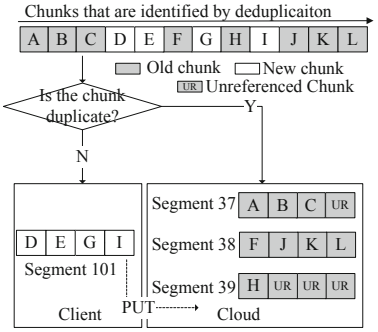


Fig. 2. An example of fragment in cloud backup

chunks A, B and C are not fragments. The same are with chunks F, J, K and L. Segment 39's reference ratio is 0.25 which is smaller than the threshold. Therefor, chunk H is a fragment.

3.3 Motivation

As shown in Table 1, there is not a defragmentation scheme for cloud backup. Existing defragmentation schemes mistakenly identify fragments in cloud backup. After defragmentation, every duplicate chunk gets a "fragment" or "not fragment" state. The existing schemes are not able to ensure that the duplicate chunks that refer to one segment are in the same state. Part of duplicate chunks that refer to an old segment are identified to be fragments and rewritten to new segment, and some other duplicate chunks that refer to the same segment are not fragments and keep the references. Both the new segment and the old segment will be downloaded in the restore process, and the fragments will be downloaded twice.

This sort of rewriting not only wastes the cloud storage space and lengthens the backup time, but also jeopardizes the restore performance. This is demonstrated in Section 5.5 and 5.6. A defragmentation scheme for deduplication based cloud backup makes sense.

4 Design and Implementation

4.1 Near-Exact Defragmentation Scheme for Deduplication Based Cloud Backup

In order to identify and rewrite the fragments in cloud backup, we propose *a near-exact defragmentation scheme for deduplication based cloud backup (NED).* NED is designed as an independent module and provides simple input and output interfaces. Thus, NED can be conveniently added to deduplication systems.

As Section 3.1 describes, the backup process of a typical cloud backup system includes 3 steps. The duplicate chunks are identified in the first step. After the

first step, every chunk gets a deduplication state, "duplicate" or "new". The second step includes writing new chunks to segments and writing the recipe. The recipe and the segments are uploaded to the cloud in the third step. NED works after the first step and before the second step.

Input. The input of NED includes chunk contents and chunk information. The chunk information contains chunk length, deduplication state, chunk address in local file system and duplicate chunk's reference.

Output. The output of NED includes not only the defragment state, but also chunk information and chunk contents that are inputted. Thus, the output keeps consistency with the input, and NED can be added to the system conveniently.

NED is constructed with *Chunk Dedup Result Buffer (CDRB)*, *Segment Reference Buffer (SRB)* and *Rewriting Monitor (RM)*. The chunk information is stored in CDRB. ID of referred segment, the corresponding *Segment Reference Length (SRL)* and *Segment Reference Ratio (SRR)* are stored in the records of SRB. Fragments are identified by RM.

4.2 Workflow

Workflow of NED is divided into *segment reference ratio statistics phase* and *defragment phase*. The defragment phase starts after all the chunks in the data stream are inputted into NED. Figure 3 shows the NED workflow.

Segment reference ratio statistics phrase. After a chunk is inputted into NED, NED inserts the chunk information into CDRB, and checks the deduplication state of the chunk. If the chunk is duplicate, NED searches SRB for the right record by the the segment ID of the chunk reference (segment ID & chunk ID). A match occurs if the segment ID in the record is equal to the segment ID of the chunk reference. If no match occurs, NED creates a new record. NED updates the SRR in the record, and the updating operation needs to add the chunk length to the SRL and compute the SRR. Because the defragment phase doesn't start until all the chunks in the data stream are inputted into NED, all inputted chunks will be stored in CDRB. However, the chunk contents are inputted into NED together with the chunk information, and the RAM consumption will be large if all the chunk contents are stored in RAM. To address this problem, NED frees the chunk contents to save RAM space.

Defragment phase. The goal of this phase is to identify the fragments within the chunks that are stored in the CDRB. Before discussing this phase, we assume that the segment reference ratio threshold is p. RM traverses all the chunks in the CDRB. If there comes a duplicate chunk, RM searches the SRB for the corresponding record by the segment ID of the chunk's reference. The chunk will be marked as a fragment if the SRR in the matched record is smaller than p. In order to keep the RAM overhead as low as possible, the chunk contents were freed at the first phase. In order to keep the format of output compatible with the input and meet the requirements of underlying storage, NED reads the chunk contents from local file system according to the local file system addresses of chunks in this phase.

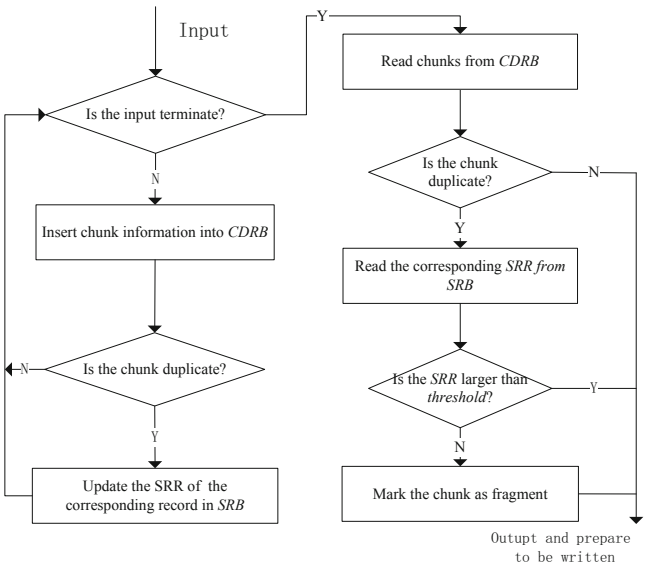


Fig. 3. The workflow of NED

As shown in Figure 4, we illustrate the working process of cloud backup system which employs NED. Chunks identified by deduplication are inputted into NED module. At segment reference ratio statistics phase, the coming chunks are sequentially inserted into CDRB and SRRs of referred segments in SRB are updated at the same time. After the first phase, segments 37, 38, and 39 are referred and the SRRs are 0.75, 1, 0.25 respectively. We assume that segment reference ratio threshold is 0.3. At the defragment phase, RM successively obtains chunks A∼L from CDRB, and determines whether they are fragments or not. Chunks A, B and C are duplicate chunks and they refer to old chunks in segment 37. The reference ratio of segment 37 is bigger than the threshold. Therefore, chunks A, B and C are not fragments. Similarly, chunks F, J, K and L are not fragments. The SRR of segment 39 is smaller than the threshold, and chunk H is a fragment.

NED analyzes the entire data stream. Duplicate chunks that refer to one segment will get the same defragment state. As the experiment results show, when the threshold is 0.9, NED near-exactly identifies the fragments in the data stream and the average restore performance is almost equal to that of backing up without deduplication.

5 Performance Evaluation

5.1 Experimental Setup

In order to test the performance of NED, we develop a simulator which is a 5,000-line C program. The simulator employs rabin fingerprint algorithm for chunking,

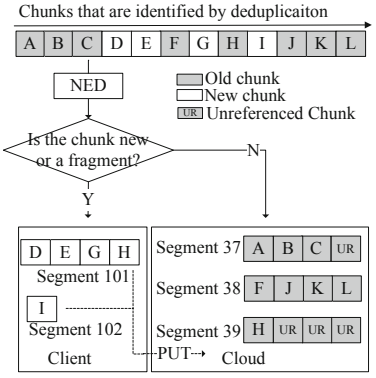
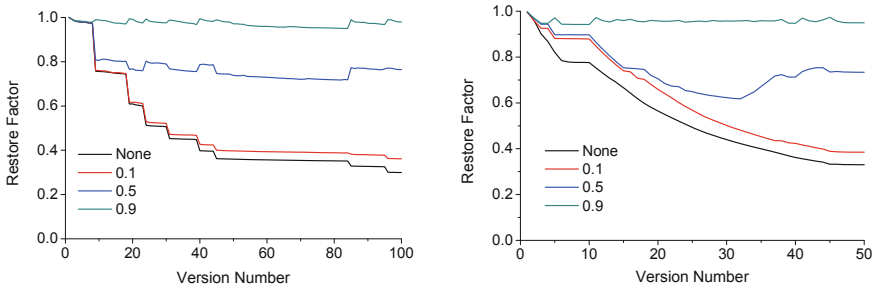


Fig. 4. An example of NED defragmentation



(a) Linux-ds **(b)** RDB-ds

Fig. 5. The restore performance. *None* denotes the no defragmentation mode, and *0.1*, *0.5* and *0.9* are the segment reference ratios of NED.

and the average chunk length is 8KB. It computes fingerprints via SHA-1. The maximal segment length is 4MB. The simulator employs hash table to store fingerprints. We implement the simulator on a machine running Ubuntu 12.04 and the machine is featured with Intel i7-930 CPU @2.80GHz, 2TB 7200RPM hard drive. We use two data sets as workload.

- Linux-ds. They are the Linux source files, from Linux 2.6.34 to Linux 3.2.04 total of 100 versions, and each version has more than 30,000 small files. The average total length of each version is 400MB. The total size of the data set is about 40GB.
- RDB-ds. We get the second data set from daily backup of Redis database for 50 days. Each version includes a mirror file whose average size is 5GB. The total size of the data set is about 250GB.

5.2 Performance Measurement

We propose *restore factor* and *average restore factor* to measure the restore performance of cloud backup system, and propose *deduplication ratio* to measure the deduplication efficiency.

Restore factor is 1/the length of segment downloaded per MB of data restored. As described in section 3.2, the bottleneck of restore in cloud backup is the segment downloading process. Less data downloaded per MB data restored indicates higher restore performance. Therefore, we propose restore factor to measure the restore performance of a single version. The larger the restore factor is, the higher the restore performance is.

Average restore factor is the average restore factor of many versions. It is used to measure the average restore performance of multiple versions.

Deduplication ratio is the ratio of the length of removed duplicate data to the length of the data stream. Only the compression caused by deduplication is considered in this paper.

5.3 The Restore Performance

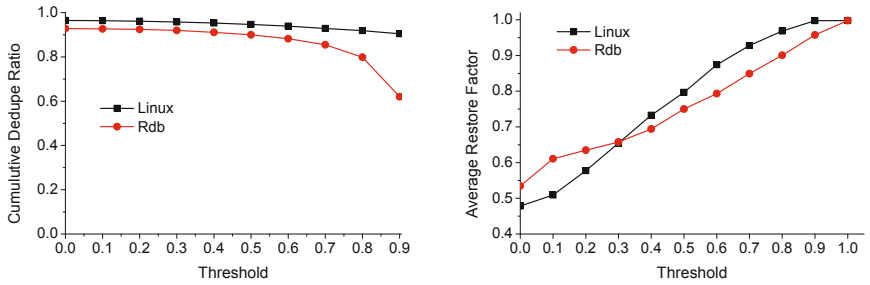
Figure 5 shows the impact of NED on restore performance. The baseline is the restore performance of no defragmentation mode. When the threshold is 0.1, restore performance is slightly improved. The restore performance is approximately 2-fold to the baseline when the threshold is 0.5. When threshold is 0.9, NED near-exactly removes the fragments. Restore factor nearly reaches 1 and is far higher than the baseline. Bigger threshold indicates that more chunks are identified to be fragments. NED significantly improves the restore performance when the threshold is bigger than 0.5.

5.4 The Segment Reference Ratio Threshold

Figure 6 shows the effect of varying segment reference ratio threshold on deduplication ratio and restore factor. No duplicate chunk is identified to be fragment and rewritten when segment reference ratio threshold is 0. Thus, the duplication ratio is the biggest and the restore factor is the smallest. All duplicate chunks are identified to be fragments when the threshold is 1, and the restore factor is the biggest. Larger threshold indicates that more duplicate chunks are marked as fragments. The increase of threshold improves the restore performance and decreases the deduplication ratio. With the growth of threshold, the restore factor increases by 6%∼105%, and deduplication ratio decreases by 0.1%∼6.5% in Linux-ds. In RDB-ds, the restore performance increases by 12%∼75%, and deduplication ratio decreases by 0.1%∼32.5%.

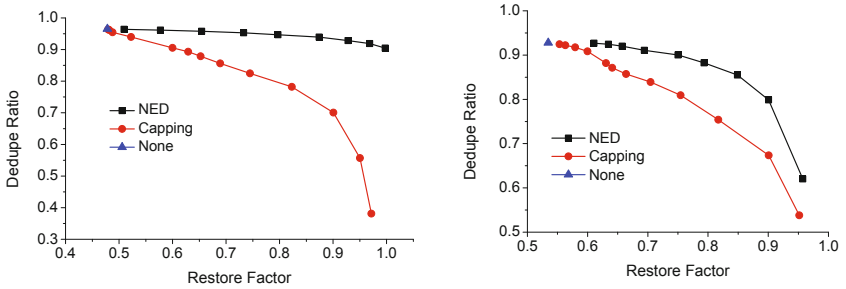
5.5 The Relationship of Restore Factor and Deduplicaton Ratio

Figure 7 shows the relationship between restore factor and deduplication ratio. Bigger restore factor indicates smaller deduplication ratio. Capping is tested as



(a) Dedup Ratio **(b)** Restore Factor

Fig. 6. Effect of varying segment reference ratio threshold on deduplication ratio and restore factor. The deduplication ratios and restore factors are the average values of the last 10 backups



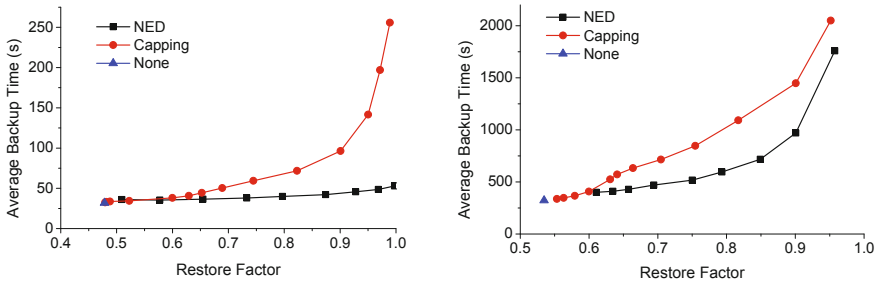
(a) Linux-ds **(b)** RDB-ds

Fig. 7. The relationship of restore factor and deduplication ratio. *None* denotes no deduplication mode. The deduplication ratios and restore factors are the average values of the last 10 backups.

an example of existing defragmentation schemes. In both data sets, the deduplication ratios of NED are larger than those of Capping in the same restore factors. It demonstrates the observation in section 3.3 that the existing defragment schemes mistakenly determine fragments in cloud backup.

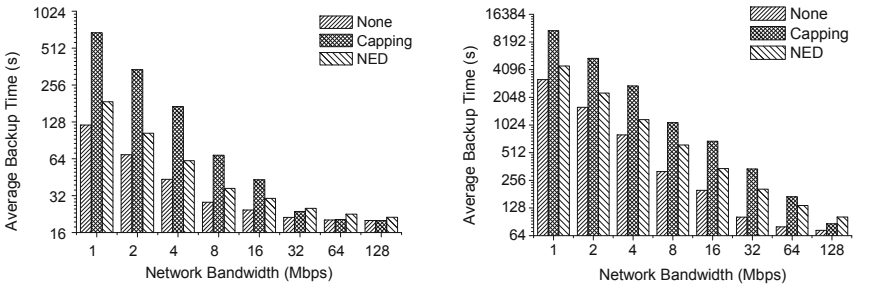
5.6 Backup Time

Figure 8 shows the average backup time of different defragment schemes. From the figure we can see, both Capping and NED bring time overhead. The backup time of NED is shorter than that of Capping in the same restore performance, and the gap between them grows with the restore factor. NED spends more time in identifying fragments than existing schemes, but it identifies fragments more accurately. The deduplication ratio of NED is bigger than that of Capping in the same restore factor. That means fewer chunks are uploaded and the uploading time is shorter. In low-bandwidth WAN environment, the reduced uploading



(a) Linux-ds **(b)** RDB-ds

Fig. 8. The time overhead. *None* denotes no deduplication mode. The restore factors are the average values of the last 10 backups.



(a) Linux-ds **(b)** RDB-ds

Fig. 9. The average backup time in different WAN bandwidths. The threshold of NED is 0.55, and Capping level is 5 segments per 20MB. With this setup, both NED and Capping improve the average restore factor to about 0.82.

time is actually much longer than the defragment time of NED. The result is that the backup time of NED is much shorter than the existing schemes.

As shown in Figure 9, lower bandwidth indicates wider uploading gap between NED and Capping. Because Capping rewrites more data than NED in the same restore factor.

NED has different effects on restore performance and deduplication ratio in different workloads. The users should adjust the segment reference ratio to make sure that not only the restore performance meets the requirements but also the decrease of deduplication ratio is acceptable. The restore performance is improved greatly when the threshold is 0.5, and the deduplication ratio decreases slightly. 0.5 is recommended as the default threshold.

6 Conclusion

Due to chunk fragmentation in deduplication based cloud backup systems, the restore performance becomes worse when the version number grows. However,

existing defragmentation schemes fail to identify fragments in cloud backup. We propose NED to identify fragments in cloud backup by precisely indentifying fragmented chunks in each backup. The experiment results show that NED effectively improves the restore performance at the cost of limited decline in terms of deduplication ratio.

Acknowledgment. This work was supported in part by National Natural Science Foundation of China (NSFC) 61173043; National Basic Research 973 Program of China 2011CB302301; NSFC 61025008, 61232004; the Seed Project of Wuhan National Laboratory for Optoelectronics (WNLO).

References

1. Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C., Welnicki, M.: Hydrastor: A scalable secondary storage. In: Proccedings of the 7th Conference on File and Storage Technologies, pp. 197–210. USENIX (2009)
2. Fu, M., Feng, D., Hua, Y., He, X., Chen, Z., Xia, W., Huang, F., Liu, Q.: Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information. In: 2014 USENIX Annual Technical Conference (USENIX ATC 14), pp. 181–192 (2014)
3. Guo, F., Efstathopoulos, P.: Building a high-performance deduplication system. In: Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference. USENIX (2011)
4. Kaczmarczyk, M., Barczynski, M., Kilian, W., Dubnicki, C.: Reducing impact of data fragmentation caused by in-line deduplication. In: Proceedings of the 5th Annual International Systems and Storage Conference, pp. 15:1–15:12. ACM (2012)
5. Lillibridge, M., Eshghi, K., Bhagwat, D.: Improving restore speed for backup systems that use inline chunk-based deduplication. Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 2013), pp. 183–197. USENIX (2013)
6. Nam, Y.J., Park, D., Du, D.H.C.: Assuring demanded read performance of data deduplication storage with backup datasets. In: Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 201–208. IEEE (2012)
7. Srinivasan, K., Bisson, T., Goodson, G., Voruganti, K.: idedup: Latency-aware, inline data deduplication for primary storage. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies. USENIX (2012)
8. Tan, Y., Jiang, H., Feng, D., Tian, L., Yan, Z.: Cabdedupe: A causality-based deduplication performance booster for cloud backup services. In: Parallel & Distributed Processing Symposium (IPDPS), pp. 1266–1277. IEEE (2011)
9. Vrable, M., Savage, S., Voelker, G.M.: Cumulus: Filesystem backup to the cloud. Trans. Storage 5(4), 14:1–14:28 (2009)
10. Xia, W., Jiang, H., Feng, D., Hua, Y.: Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput. In: USENIX Annual Technical Conference (2011)
11. Xia, W., Jiang, H., Feng, D., Tian, L.: Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets. In: Data Compression Conference (DCC 2014), pp. 203–212 (2014)

12. Xu, Q., Zhao, L., Xiao, M., Liu, A., Dai, Y.: Yurubackup: A space-efficient and highly scalable incremental backup system in the cloud. International Journal of Parallel Programming, 1–23 (2013)
13. Zhan, D., Jiang, H., Seth, S.: Exploiting set-level non-uniformity of capacity demand to enhance cmp cooperative caching. In: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), pp. 1–10 (2010)
14. Zhan, D., Jiang, H., Seth, S.: Stem: Spatiotemporal management of capacity for intra-core last level caches. In: 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 163–174 (2010)
15. Zhan, D., Jiang, H., Seth, S.C.: Locality & utility co-optimization for practical capacity management of shared last level caches. In: Proceedings of the 26th ACM International Conference on Supercomputing, pp. 279–290. ACM (2012)
16. Zhu, B., Li, K., Patterson, H.: Avoiding the disk bottleneck in the data domain deduplication file system. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies, pp. 18:1–18:14. USENIX (2008)