DASM: A Dynamic Adaptive Forward Assembly Area Method to Accelerate Restore Speed for Deduplication-based Backup Systems

Chao Tan[1], Luyu Li[1], Chentao Wu[1(✉)], and Jie Li[1,2]

[1]Shanghai Key Laboratory of Scalable Computing and Systems
Dept. of Computer Science and Engineering, Shanghai Jiao Tong University,
Shanghai 200240, China
[2]Department of Computer Science
University of Tsukuba, Tsukuba, Ibaraki 305-8577, Japan
(✉) Corresponding Author: wuct@cs.sjtu.edu.cn

Abstract. Data deduplication yields an important role in modern backup systems for its demonstrated ability to improve storage efficiency. However, in deduplication-based backup systems, the consequent exhausting fragmentation problem has drawn ever-increasing attention over in terms of backup frequencies, which leads to the degradation of restoration speed. Various Methods are proposed to address this problem. However, most of them purchase restore speed at the expense of deduplication ratio reduction, which is not efficient.
In this paper, we present a Dynamic Adaptive Forward Assembly Area Method, called DASM, to accelerate restore speed for deduplication-based backup systems. DASM exploits the fragmentation information within the restored backup streams and dynamically trades off between chunk-level cache and container-level cache. DASM is a pure data restoration module which pursues optimal read performance without sacrificing deduplication ratio. Meanwhile, DASM is a resource independent and cache efficient scheme, which works well under different memory footprint restrictions. To demonstrate the effectiveness of DASM, we conduct several experiments under various backup workloads. The results show that, DASM is sensitive to fragmentation granularity and can accurately adapt to the changes of fragmentation size. Besides, experiments also show that DASM improves the restore speed of traditional LRU and ASM methods by up to 58.9% and 57.1%, respectively.

Keywords: Data Deduplication; Restore Speed; Reliability; Cache Policy; Performance Evaluation

1 Introduction

Data deduplication is an effective technique used to improve storage efficiency in modern backup systems. [13] [2]. A typical deduplication-based backup system partitions backup streams into variable-size or fixed-size chunks. Each data chunk is identified by fingerprints calculated using cryptographic methods, such as SHA-1 [11]. Two chunks are considered to be duplicates if they have identical fingerprints. For each chunk, deduplication system employs Key-Value Store,

also referred to fingerprint index, to identify possible duplicates. Only new fresh chunks are physically stored in containers while duplicates are eliminated.

However, in backup systems, the deviation between physical locality and logical locality increases when backup frequencies are improved, which leads to the physical dispersion of subsequent backup streams. The consequent exhausting fragmentation problem has drawn ever-increasing attention, which results in the degradation of restoration speed and expensive garbage collection operations [8] [10].

In the past decade, various methods are proposed to address this problem [7] [9]. Most of them purchase restore speed by sacrificing the deduplication ratio, which is costly [12] [6]. Traditional restoration method employs LRU algorithm to cache chunk containers. In many scenarios, LRU buffers a large amount of unuseful chunks within containers. Besides, unlike some other applications, we have perfect pre-knowledge of future access informations during restoration, which cannot be well utilized by LRU. A few advanced approaches like ASM [6] employs a forward assembly area to prefetch data chunks within a same container. Nevertheless, ASM is a chunk-level restoration method which limits its read performance due to the one-container cache regulation.

In this paper, we present a Dynamic Adaptive Forward Assembly Area Method, called DASM, to accelerate restore speed for deduplication-based backup systems. DASM exploits the fragmentation information within the restored backup streams and dynamically trades off between chunk-level cache and container-level cache. DASM is a pure data restoration module which pursues optimal read performance without sacrificing deduplication ratio. Meanwhile, DASM works well under different memory footprint restrictions.

The main contributions of this paper are summarized as follows,

- We propose a novel Dynamic Adaptive Forward Assembly Area method (DASM) for deduplication-based backup systems, which outperforms prior approaches in terms of restoration speed. DASM performs well in different scenarios, such as various memory restrictions and backup workloads.
- We conduct several experiments to demonstrate the effectiveness of DASM. The results show that DASM sharply improves the restore speed in various workloads.

The remainder of the paper is organized as follows. Section 2 reviews the background and the motivation of DASM. Section 3 illustrates the details of our DASM. Section 4 evaluates the performance of our scheme compared with other popular restoration cache algorithms. Finally we conclude our work in Section 5.

2 Background and Motivation

In this section, firstly, we introduce data deduplication briefly. Then we explore the fragmentation problem and how it impacts restoration speed. After that, an elaborate investigation of existing solutions and the motivation of our approach are exhibited.

2.1 Data Deduplication

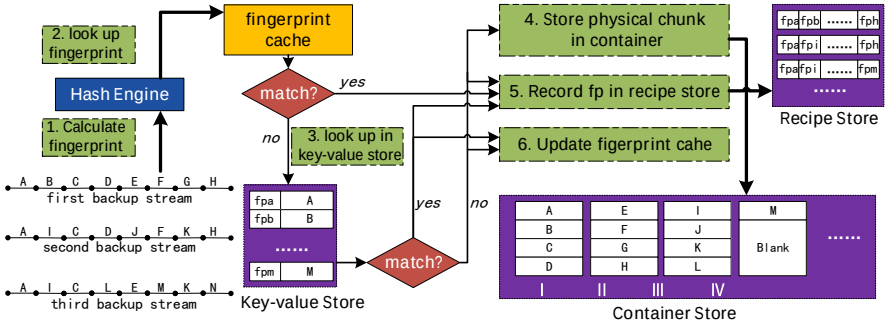


Fig. 1. Data Deduplication System Architecture.

In backup systems, data deduplication can significantly improve storage efficiency [13] [2]. Figure 1 illustrates a typical data deduplication system architecture [4].

Two chunks are considered to be duplicates if they have identical fingerprints. A typical deduplication system employs Key-Value Store, referred to fingerprint index as well, to identify possible duplicates. The key-value store maps each fingerprint to the corresponding physical location of data chunks. The recipe store is used to record logical fingerprint sequences according to the order of the original data stream, in case of future data restoration. And container store is a log-structured storage warehouse. After deduplication, data chunks are aggregated into fixed-sized containers(usually 4MB) and stored in a container store.

As shown in Figure 1, original backup streams are partitioned into several variable or fixed-size chunks. The hash engine calculates a fingerprint for each data chunk using cryptographic methods, such as SHA-1 [11].

Once a fingerprint is produced, the system takes following steps to eliminate duplicates: (1) Look up in the fingerprint cache. (2) If it is a cache hit, it means that a duplicate data chunk has already existed. In this situation, we can only record the fingerprint into recipe store and terminate the deduplication process. (3) If cache misses, it looks up in the key-value store for further identification. (4) If there is a match, jump to step 5. Otherwise, the data chunk is considered to be a new fresh and stored into containers in the container store. (5) Record the fingerprint in the recipe store. (6) Update fingerprint cache to explore data localities.

2.2 Fragmentation Problem and Restoration Speed

Fragmentation is one of the exhausting problems caused by typical deduplication scheme. As we can see in Figure 1, physical chunks are stored sequentially in order of their appearance. In the essence, the physical locality is similar to the logical locality for the first backup stream. However, this deviation increases as

time goes because duplicate chunks are eliminated, which leads to the physical dispersion of subsequent backup streams.

Fragmentation problems is troublesome in many aspects. First, it degrades the restoration speed [8]. In deduplication systems, read and write operations are executes with basic granularity of container. Thus data restoration should be faster with consecutive physical distribution. Besides, chunks could become invalid because of data deletion operations. Physical dispersion results in holes within the containers, which leads to the emergence of inefficient garbage collection [10].

Various methods are proposed to address these problems. On one hand, several methods accelerate the restore speed via decreasing deduplication ratio. [3]. iDedup [6] selectively deduplicates only sequences of disk blocks and replaces the expensive, on-disk, deduplication metadata with a smaller, in-memory cache. These techniques enable it to tradeoff deduplication ratio for performance. The context-based rewriting (CBR) minimizes drop in restore performance for latest backups by shifting fragmentation to older backups, which are rarely used for restore [5].

Traditional restoration method employs LRU algorithm to cache chunk containers. However, LRU is a container-level cache policy which buffers a large amount of unused chunks within containers. Besides, unlike other data access pattern, we have perfect pre-knowledge of future access information during restoration, which is not utilized by LRU. ASM [6] employs a forward assembly area to prefetch data chunks within a same container. Nevertheless, ASM is a chunk-level restoration method which limits its read performance since the one-container cache regulation.

2.3 Forward Assembly Area

Forward Assembly Area (ASM) is a chunk-level restoration method which caches chunks rather than containers for better cache performance. Figure 2 depicts the basic functions of ASM.

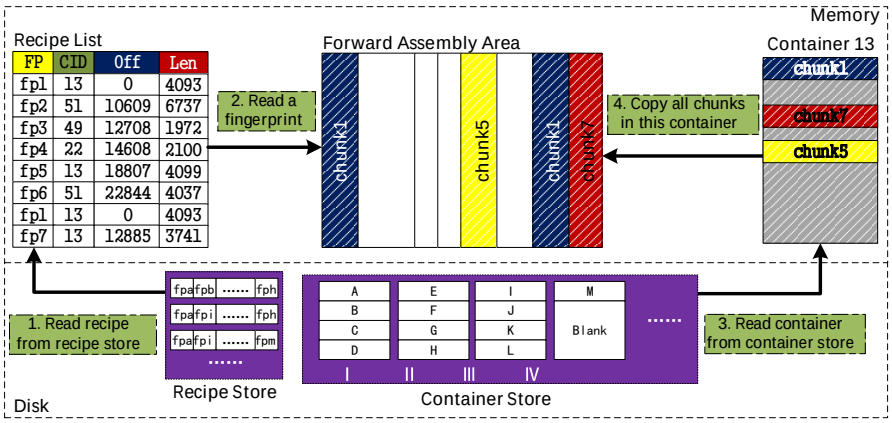


Fig. 2. Forward Assembly Area(ASM) Method.

ASM employs a large forward assembly area to assemble M bytes of the restored backup. Meanwhile, it utilizes two dedicated buffers, a recipe buffer to cache recipe streams and a chunk-container-sized buffer to cache proper containers that is being used.

In a simple case shown in Figure 2, firstly, ASM reads the recipes from their related recipe store preserved in disk into the recipe buffer. Then, ASM obtains the top n recipes which hold at most M bytes of the restored data. After that, ASM finds the earliest unrestored chunk and loads the corresponding container into its I/O buffer. For $chunk1$ with fingerprint $fp1$, the corresponding container $container13$ is loaded into the container buffer. Then, all the chunks belongs to $container13$ in the top n recipe items will be restored ,which are $chunk1$, $chunk5$,$chunk7$ in this case. ASM repeats this procedure until the M bytes are completely filled. Finally, ASM flushes these M bytes of restored data into disk.

ASM restores M bytes every time and only one container is loaded into memory during each recovery, which improves the cache performance. However, the one-container regulation may degrade the read performance since each container have to wait until the last one finishes restoration of all chunks.

2.4 Our Motivation

Many previous literatures attempt to figure out the fragmentation problem through deduplication procedures, such as rewriting. These methods purchase restore speed at the expense of deduplication ratio reduction, which is unworthy. From the restoration's perspective purely, traditional LRU cache policy ignores the perfect pre-knowledge of future access information during restoration and holds plenty of useless chunks in memory, which is a big resource waste. ASM is a chunk-level restoration method which limits its read performance since the one-container cache regulation.

To address this problem, we propose a novel Dynamic Adaptive Forward Assembly Area method (DASM) for deduplication-based backup systems, which arms ASM with a multiple-container-sized cache. DASM adaptively adjusts the size of the forward assembly area and the container cache according to the fragmentation level of the restored backup streams to pursue optimal restoration performance. DASM is resource independent and cache efficient, which outperforms prior approaches under different memory footprint restrictions and various backup workloads.

3 Design of DASM

To overcome the shortages of the forward assembly area method, we present a Dynamic Adaptive Forward Assembly Area method (DASM) which arms ASM with a multiple-container-sized cache. Figure 3 exhibits the overall architecture of DASM.

Different from ASM, DASM carries a multiple-container-sized cache called Container Cache, which buffers multiple containers. To reduce the resource de-

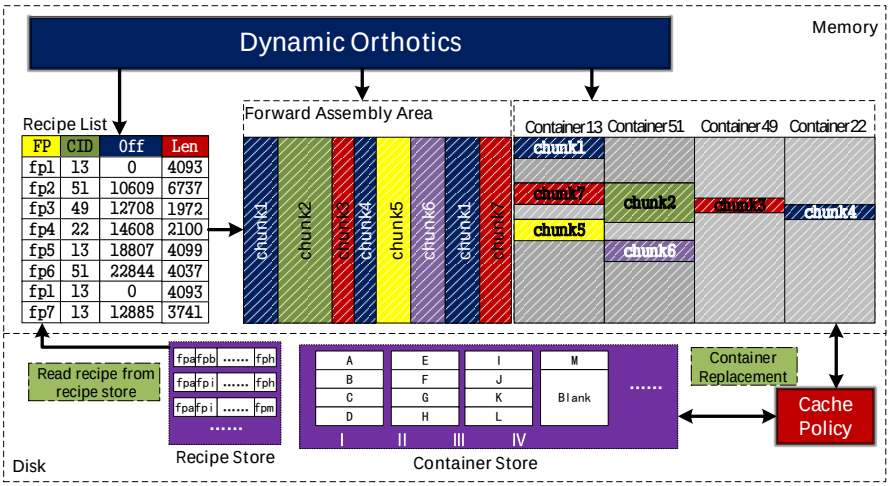


Fig. 3. Overview of Dynamic Adaptive Forward Assembly Area Method(DASM).

pendencies and increase cache efficiencies, we impose restrictions on memory footprint by bounding the overall size of the ASM area and the container cache.

Besides the original ASM area and the new joint container cache, DASM involves other two main components as below,

- **Dynamic Orthotics**
  Dynamic Orthotics is a calculation model which exploits future access information to pursue optimal restoration performance within the restriction of finite memory footprint. It regards the fragmentation rate as a major criterion and dynamically adjusts the proportion of the ASM area to the container cache. Details are discussed in Section 3.1.
- **Cache Policy**
  Cache Policy module depicts the cache replacement policies carried out in the container cache. It works as a centralized scheduler which dispatches containers legitimately to reduce cache miss and minimize the bandwidth requirement.

DASM abides by the following steps,

1) Tradeoff the ASM area and the container cache according to the fragmentation rate, which are typically done by Dynamic Orthotics module. After Dynamic Orthotics completes the tradeoff, the top n chunks within C containers are restored at this time.

2) Load the required C containers into the container cache. If there is a cache miss, DASM selects appropriate cache policies for container replacement.

3) Execute the ASM restoration schemes policies repeatedly: find the earliest unrestored chunk and copy all the chunks pertains to its corresponding container to the forward assembly area.

4) Flush all the restored data to disks and restart from step 1.

3.1 Dynamic Orthotics

Restoration speed is strongly depended on the fragmentation granularity of the restored data stream. We consider two extreme situations,

(1) Lowest fragmentation granularity for the first backup: as for the first backup. in this scenario, we can shrink container caches to a single container size since almost all consecutive chunks belong to a same container.

(2) Highest fragmentation granularity that almost all unique chunks are placed in different containers with each other: Under this circumstance, we have to shrink ASM area to acquire as large container cache area as possible.

Based on these observations, we propose a mathematical model called Dynamic Orthotics, which exploits future access information to pursue optimal restoration performance within the restriction of finite memory footprint. It regards fragmentation rate as the major criterion and dynamically adjusts the proportion of the ASM area to container cache.

For clarity, we define a few symbols as shown in Table 1.

Table 1. Notations of DASM

Notation	Description
ASM	Forward assembly area whose size is donated by $\|ASM\|$
CC	Container cache whose size is donated by $\|CC\|$
CS	Container Size
MFP	Overall memory footprint whose size is donated by $\|MFP\|$
$chunksize_i$	Chunk size of the ith chunk in the recipe stream
C_i	The overall number of containers required by the top i chunks
n	The number of chunks waiting to be restored at a time
$rate$	Fragmentation Rate

Suppose there are n chunks waiting to be restored in the forward assembly area and they belong to C_n containers. To obtain optimal restoration performance, ASM area should hold exactly $\sum_{i=1}^{n} chunksize_i$ bytes while container cache has the ability to load accurately C_n containers. The relationships between these notations can be formulated by the following equations:

$$|ASM| = \sum_{i=1}^{n} chunksize_i \tag{1}$$

$$|CC| = C_n * CS \tag{2}$$

$$|ASM| + |CC| \leq |MFP| \tag{3}$$

We define fragmentation rate by the following equation,

$$fragmentation\ rate = \frac{|CC|}{|MFP|} \tag{4}$$

As shown in Algorithm 1, Dynamic Orthotics reads chunk information sequentially from the recipe buffer. For each chunk, dynamic orthotics calculates whether it can be pushed into ASM. If Equation 1 2 3 still holds, then add the

chunk into ASM. Otherwise, terminate and adjust |ASM| and |CC| according to the returned value.

Algorithm 1: Dynamic Orthotics

Aim: Dynamically adjust the proportion of the ASM area to container cache according to the fragmentation rate.

while *(chunk* $\leftarrow$ *RecipeBuffer.head)*$\neq$ *null* **do**
  C'_n = chunk.CID in CC? C_n : $C_n + 1$
  if *(ASM.size+chunk.size)*+C'_n **CS* $\leq$ *MFP.size* **then**
    ASM.size+=chunk.sie
    C_n=C'_n
  else
    return *ASM.size,* C_n

3.2 Near-optimal Cache Policy

To reduce container cache miss penalty as much as possible, we implement a variant of Belady's optimal replacement cache policy [1].

Belady's policy evicts the cached container that will be accessed in the farthest future. With foreseeable container access information, it is oversimplified to achieve optimality. However, with fragmentation rate high enough, we may have to traverse along the recipe list for a long time to find a victim, which is expensive. Therefore we establish several cache policies which could achieve near-optimal performance under framework of DASM.

- If a container has already existed in the container cache and it will be reused during the next round, keep it in the cache.
- If there exists no such containers, evict all containers in the container cache.

DASM dynamically adapts the size of container cache and reserve exactly C_n containers' size for the containers required. Since containers which are not reused during the next round will be evicted no matter how, we just evict them without considering replacement order.

4 Performance Evaluation

To demonstrate the effectiveness of DASM, we conduct several experiments using various backup datasets to evaluate the performance of DASM and compare it with traditional restore methods such as LRU and ASM.

4.1 Evaluation Methodology

1) Evaluation Platform We utilize an open-source deduplication platform called Deduplication Framework (DeFrame) [4], for comprehensive data deduplication evaluation. DeFrame implements the entire parameter space discussed

in a modular and extensible fashion; this enables apple-to-apple comparisons among both existing and potential solutions [4].

We implement and evaluate a DASM prototype on DeFrame under the Ubuntu 14.10 operating system, with linux kernel version 3.13.0, running on a quad-core Intel i5-4430 processor at 3.00 GHz, with a 12GB RAM, a 1TB 7200RPM hard disk drive.

2) Traditional restoration methods for comparison This evaluation employs LRU, ASM as the contrasts. As illustrates in Section 2, traditional restoration method employs LRU algorithm to cache chunk containers. ASM employs a forward assembly area to prefetch data chunks within a same container.

3) Evaluation Metrics In this paper, we exploit restoration speed as our major evaluation metric. Restore speeds are measured in term of encoding time in seconds.Besides, we explores fragmentation rate to show that DASM is sensitive to fragmentation degree and can adapt to fragmentation change accurately. The fragmentation rate is defined in Section 3.1 as:

$$fragmentation\ rate = \frac{|CC|}{|MFP|} \tag{5}$$

4) Evaluation Datasets We choose three real-world files as our datasets: ubuntu-14.04.3, ubuntu-14.10 and ubuntu-15.04. These datasets are firstly stored as the 1st, 2nd and 3rd backup streams in the deduplication systems, respectively. After that, we evaluates performance of LRU, ASM and our DASM methods by restoring each original backup streams. Table 2 lists some detailed information about these datasets.

Table 2. Detailed Information about Datasets

Backup data	Bytes	Chunks	Average Chunk Size	Containers
ubuntu-14.04.3	1054867456	228791	4610	255
ubuntu-14.10	1186529280	257945	4599	378
ubuntu-15.04	1150844928	250609	4592	480

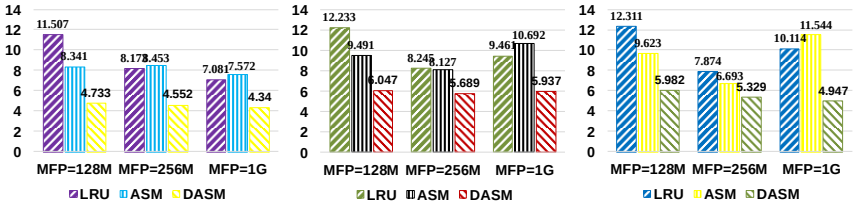
4.2 Experimental Results

In this section, we give the experimental results of DASM and compare it with LRU and ASM under three backup streams to illustrate its benefits.

1) Restore Speed Under Different Cache Size Table 3 and Figure 4 describe the comparison between restore speed among LRU, ASM and DASM under different cache size. Experiments show that DASM improves the restore speed of traditional LRU and ASM methods by up to 58.9% and 57.1%, respectively.

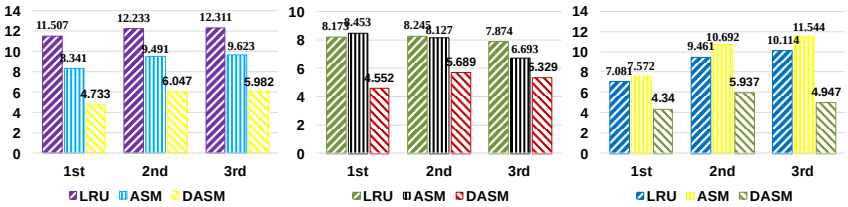
Table 3. Comparison among LRU, ASM and DASM under different cache size in terms of restore speed.

	128M			256M			1G		
	1st	2nd	3rd	1st	2nd	3rd	1st	2nd	3rd
DASM	4.733	6.047	5.982	4.552	5.689	5.329	4.34	5.937	4.947
LRU	11.507	12.233	12.311	8.173	8.245	7.874	9.081	9.461	10.114
Improvement	58.9%	50.6%	51.4%	44.3%	31.0%	32.3%	52.2%	57.2%	51.1%
ASM	8.341	9.491	9.623	8.453	8.127	6.693	9.572	10.692	11.544
Improvement	43.3%	36.3%	37.8%	46.1%	30.0%	20.3%	54.7%	44.5%	57.1%



(a) Restore speed for the 1st backup (b) Restore speed for the 2nd backup (c) Restore speed for the 3rd backup

Fig. 4. Comparison among LRU, ASM and DASM under different cache size in terms of restore speed.



(a) Restore speed for 128M MFP (b) Restore speed for 256M MFP (c) Restore speed for 1G MFP

Fig. 5. Comparison among LRU, ASM and DASM under different backup streams in terms of restore speed.

2)Restore Speed Under Different backup streams Figure 5 illustrates the comparison between restore speed among LRU, ASM and DASM under different backup streams. Since the first backup is stored sequentially among the containers, which means that it carries the lowest fragmentation problem, it is restored fastest under arbitrary circumstances. As shown in the Figure, DASM outperforms LRU and ASM under any backup streams.

3) Fragmentation Rate Figure 6 depicts the fragmentation rate changing curves under different backup streams. As we can see in the figure, regardless of the size of memory footprint, as backup time increases, the average fragmentation rate rises with amplitude aggrandizament (3rd > 2nd > 1st). This phenomenon implies that our DASM can effectively and accurately detect fragmentation changes and adjust accordingly.

4.3 Analysis

Traditional LRU cache policy ignores the perfect pre-knowledge of future access information during restoration and holds plenty of useless chunks in memory,

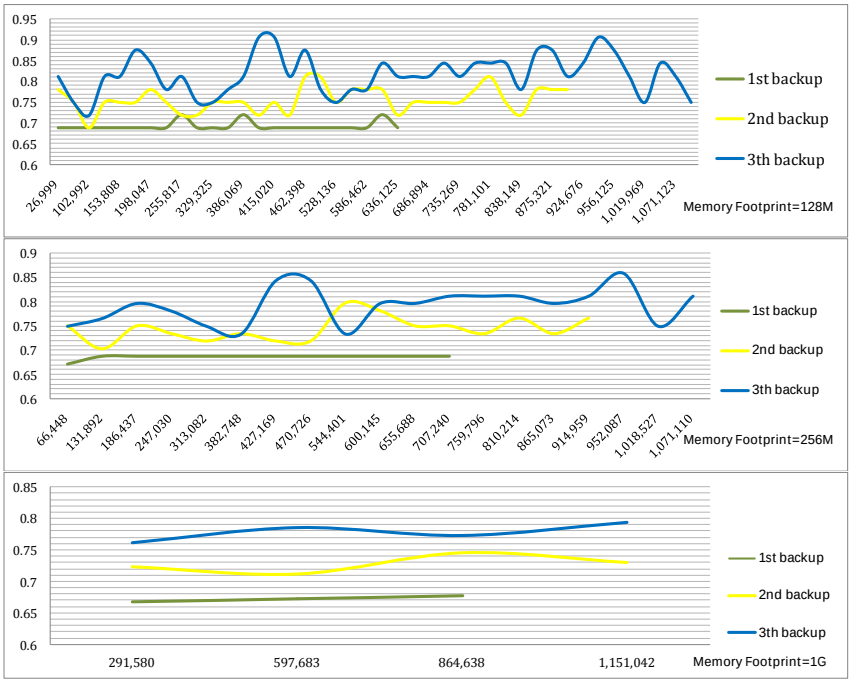


Fig. 6. Fragmentation rate curves under different backup streams

which is a big resource waste. ASM is a chunk-level restoration method which limits its read performance since the one-container cache regulation.

From the results in the previous subsecitons, it is clear that DASM approach has many advantages on restore speed in backup systems compared to the previous restore methods: LRU and ASM. There are several reasons to achieve these gains. First, DASM integrates container cache which can effectively enhance the hit ratio of chunks and reduce read latency. Second, DASM can accurately capture current fragmentation granularit and dynamiclly adjust ASM area size and container cache size to achieve the best performance. Therefore, DASM outperforms LRU and ASM enormously.

5 Conclusions

In this paper, we present a Dynamic Adaptive Forward Assembly Area Method, called DASM, to accelerate restore speed for deduplication-based backup systems. Different from previous approaches, DASM is a pure data restoration module which pursue optimal read performance without sacrificing deduplication ratio. DASM exploits the fragmentation information within the restored backup streams and dynamically trades off between chunk-level cache and container-level cache which leverages both advantages of LRU and ASM. Experiments results show that DASM is sensitive to fragmentation degree and can adapt to fragmentation change accurately. Besides, experiments also show that DASM improves

the restore speed of traditional LRU and ASM methods by up to 58.9% and 57.1%, respectively.

6 Acknowledgement

We thank anonymous reviewers for their insightful comments. This work is partially sponsored by the National 863 Program of China (No. 2015AA015302), the National 973 Program of China No.2015CB352403), the National Natural Science Foundation of China (NSFC) (No. 61332001, No. 61303012, and No. 61572323), the Scientific Research Foundation for the Returned Overseas Chinese Scholars, and the CCF-Tencent Open Fund.

References

1. L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*.
2. T. Asaroand H. Biggar. Data de-duplication and disk-to-disk backup systems: Technical and business considerations. *The Enterprise Strategy Group*.
3. M. Fu, D. Feng, et al. Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information. In *USENIX Annual Technical Conference'14*.
4. M. Fu, D. Feng, et al. Design tradeoffs for data deduplication performance in backup workloads. In *Proceedings of the USENIX FAST'15*.
5. M. Kaczmarczyk, M. Barczynski, et al. Reducing impact of data fragmentation caused by in-line deduplication. In *Proceedings of the 5th Annual International Systems and Storage Conference*.
6. M. Lillibridge, K. Eshghi, and D. Bhagwat. Improving restore speed for backup systems that use inline chunk-based deduplication. In *Proceedings of the USENIX FAST'13*.
7. N. Mandagere, P. Zhou, et al. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*.
8. Y. J. Nam, G. L. Lu, et al. Chunk fragmentation level: An effective indicator for read performance degradation in deduplication storage. In *High Performance Computing and Communications (HPCC)'11*.
9. J. Paulo and J. Pereira. A survey and classification of storage deduplication systems. *ACM Computing Surveys (CSUR)*.
10. W. C. Preston. Restoring deduped data in deduplication systems. *SearchDataBackup.com*, 2010.
11. S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *Proceedings of the USENIX FAST'02*.
12. K. Srinivasan, T. Bisson, et al. idedup: latency-aware, inline data deduplication for primary storage. In *Proceedings of the USENIX FAST'12*.
13. G. Wallace, F. Douglis, et al. Characteristics of backup workloads in production systems. In *Proceedings of the USENIX FAST'12*.