Read-Performance Optimization for Deduplication-Based Storage Systems in the Cloud

BO MAO, Xiamen University
HONG JIANG, University of Nebraska-Lincoln
SUZHEN WU, Xiamen University
YINJIN FU, National University of Defense Technology
LEI TIAN, University of Nebraska-Lincoln

Data deduplication has been demonstrated to be an effective technique in reducing the total data transferred over the network and the storage space in cloud backup, archiving, and primary storage systems, such as VM (virtual machine) platforms. However, the performance of restore operations from a deduplicated backup can be significantly lower than that without deduplication. The main reason lies in the fact that a file or block is split into multiple small data chunks that are often located in different disks after deduplication, which can cause a subsequent read operation to invoke many disk IOs involving multiple disks and thus degrade the read performance significantly. While this problem has been by and large ignored in the literature thus far, we argue that the time is ripe for us to pay significant attention to it in light of the emerging cloud storage applications and the increasing popularity of the VM platform in the cloud. This is because, in a cloud storage or VM environment, a simple read request on the client side may translate into a restore operation if the data to be read or a VM suspended by the user was previously deduplicated when written to the cloud or the VM storage server, a likely scenario considering the network bandwidth and storage capacity concerns in such an environment.

To address this problem, in this article, we propose SAR, an SSD (solid-state drive)-Assisted Read scheme, that effectively exploits the high random-read performance properties of SSDs and the unique data-sharing characteristic of deduplication-based storage systems by storing in SSDs the unique data chunks with high reference count, small size, and nonsequential characteristics. In this way, many read requests to HDDs are replaced by read requests to SSDs, thus significantly improving the read performance of the deduplication-based storage systems in the cloud. The extensive trace-driven and VM restore evaluations on the prototype implementation of SAR show that SAR outperforms the traditional deduplication-based and flash-based cache schemes significantly, in terms of the average response times.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management; D.4.8 [**Operating Systems**]: Performance

General Terms: Design, Performance

Additional Key Words and Phrases: Storage systems, data deduplication, virtual machine, solid-state drive, read performance

---

This work is supported by the China National Science Foundation no. 61100033, the US NSF under Grant No. NSF-CNS-1116606, NSF-CNS-1016609, NSF-IIS-0916859, the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, and the Huawei Innovation Research Program.
An earlier version of this article appeared in *Proceedings of the 7th IEEE International Conference on Networking, Architecture, and Storage (NAS'12)*.
B. Mao was a postdoc researcher at the University of Nebraska-Lincoln when this work was done.
Authors' addresses: B. Mao and S. Wu (corresponding author), Xiamen University, Xiamen 361005, China; email: suzhen@xmu.edu.cn; H. Jiang and L. Tian, University of Nebraska-Lincoln; Y. Fu, National University of Defense Technology, Changsha 410073, China.

DOI: http://dx.doi.org/10.1145/2512348

ACM Reference Format:
Mao, B., Jiang, H., Wu, S., Fu, Y., and Tian, L. 2014. Read-performance optimization for deduplication-based storage systems in the cloud. *ACM Trans. Storage* 10, 2, Article 6 (March 2014), 22 pages.
DOI: http://dx.doi.org/10.1145/2512348

1. INTRODUCTION

With the explosive growth of digital content, the demand for storage capacity has been mounting, along with an increasing need for more cost-effective use of storage capacity. Data deduplication, as a space-efficient method, has been proposed for optimizing applications such as data backup and archiving, and even for primary storage for the virtual machine (VM) servers by reducing the amount of storage space consumed by the current datasets. In the deduplication process, the duplicate data chunks are deleted, leaving only a single instance of each unique data chunk to be stored. Data deduplication has been proven to be very effective in saving storage space and network bandwidth in high-performance, data-intensive environments [Muthitacharoenand et al. 2001; Nath et al. 2008; Tan et al. 2011]. Different applications have different levels of data redundancy [Meyer and Bolosky 2011]. For example, virtual machines can benefit greatly from data deduplication because most OS-specific binaries are similar across different guest instances, implying a high level of data redundancy. In many cases, the VM servers with data deduplication can obtain more than 80% storage reduction [Clements et al. 2009]. Backup and archiving applications can generally benefit significantly from data deduplication due to the nature of repeated full backups of an existing file system [Tan et al. 2011].

Since the unique data chunks that are actually stored after data deduplication are shared among all original sources of those data chunks, the duplicate data chunks are replaced by pointers to the data chunks that may well be located in many different disks. This can result in a file or block being physically fragmented into multiple small data chunks located in multiple disks. Thus, each of the subsequent read requests issued to the file or block will likely result in many expensive disk seeks. The performance of these reads in fact directly reflects that of the restore operation in the deduplication-based systems. In other words, the read performance will suffer significantly from the data fragmentation caused by the deduplication process. For example, it was acknowledged that the read performance declines noticeably on the deduplicated data in EMC's DL3D system, and there are other similar deduplication solutions that cause the read performance to decline by more than 60% over time.[1] Our preliminary experimental study on the virtual machine disk images reveals that the restore time is more than doubled when the deduplication is used.

Moreover, storage system reliability can be measured by MTTDL (mean time to data loss) and is inversely proportional to MTTR (mean time to repair). Since the data restore operation is a significant part of MTTR, its performance will not only affect the user-perceived response time, but also directly impact the reliability of storage systems. Therefore, the deduplication-based backup should not only shorten the backup window and save network bandwidth and storage space, but and more importantly, also shorten the recovery window. Unfortunately, the issue of read performance has received very little attention in the literature. For example, in backup recovery, only 33% of recovery cases are successful [Xiao and Yang 2008], and 48% of organizations need to reduce recovery times [ESG 2008]. In the VM servers, read performance is arguably even more important, particularly in primary storage systems and the cloud environment where the restores from the VM suspensions and migrations are very

[1]Deduplication and restore performance.
http://www.aboutrestore.com/2008/08/08/deduplication-and-restore-performance/.

frequent and directly reflect the quality of the cloud services as perceived by clients. Moreover, the reduced read performance can also lead to SLA (service-level agreement) and SLO (service-level objective) violations, which in many environments, such as Amazon S3 (Simple Storage Service)[2], are also perceived as reduced revenues for service providers.

Thus, the read efficiency in deduplication-based storage systems is critically important from the viewpoints of system performance and reliability. There are several proposed solutions that attempt to overcome the aforementioned problem, including forward referencing (used by SEPATON's DeltaStor technology[3]) and built-in defragmentation [Rhea et al. 2008; Yang et al. 2010]. However, these solutions increase either system overhead or fragmentation of the stored data. To address this problem more comprehensively and effectively, in this article, we propose a new solution, SSD Assisted Read (SAR), to improve the read performance in deduplication-based storage systems. SAR selectively stores in SSDs unique data chunks with high reference count, small size, and nonsequential workload characteristics to effectively exploit the unique data-sharing characteristic of deduplication-based storage systems and the high random-read performance of SSDs. As a result of SAR, many read requests to multiple HDDs are replaced by read requests to SSDs, thus significantly improving the read performance.

More specifically, SAR has the following salient features.

First, SAR significantly improves read performance in deduplication-based storage systems by transforming many small HDD-bound read IOs to SSD-bound IOs to fully leverage the significant random-read-performance and energy-efficiency advantages of the latter over the former.

Second, SAR improves system reliability and availability by significantly shortening the restore window, a substantial part of MTTR.

Third, SAR is independent of and orthogonal to any existing deduplication methods and is simple to understand and implement, requiring very few changes to existing data deduplication methods. Thus it can be easily embedded into any existing deduplication methods to improve read performance.

To evaluate the effectiveness of the proposed SAR, we have implemented a prototype of SAR by integrating it into the open-source data deduplication project called SDFS.[4] Our extensive trace-driven evaluations of the SAR prototype show that SAR outperforms the traditional deduplication-based system significantly in read operations by a speedup factor of up to 28.2×, with an average factor of 5.8× in terms of the average response time. The VM restore evaluations of the prototype implementation of SAR show that SAR reduces the user response time of the traditional deduplication-based storage system by an average of 83.4% and up to 176.6%.

The rest of this article is organized as follows. Background and motivation are presented in Section 2. We describe the SAR architecture and design in Section 3. Performance evaluations are presented in Section 4. We review the related work in Section 5 and conclude in Section 6.

2. BACKGROUND AND MOTIVATION

In this section, we present the necessary background about data deduplication and important observations drawn from our preliminary experimental study on the data

[2]http://aws.amazon.com/s3/

[3]http://sepaton.com

[4]http://www.opendedup.org/

fragmentation phenomenon in deduplication-based storage systems, followed by the description of flash-based SSDs to motivate our SAR research.

2.1. Data Deduplication Basics

Data deduplication is a specific type of data compression. It splits files into multiple data chunks that are each uniquely identified by a fingerprint that usually is a hash signature of the data chunk and removes the duplicate data chunks by checking their fingerprints, which avoids a byte-by-byte comparison. Data deduplication has been an essential and critical component in cloud backup and archiving storage systems. It not only reduces the storage space requirements but also improves throughput of the backup and archiving systems by eliminating the network transmission of redundant data, as well as reduces energy consumption by deploying fewer HDDs. Recently, data deduplication has been applied to primary storage systems, such as the VM servers and ZFS[5] [Clements et al. 2009; Dong et al. 2011; Meyer and Bolosky 2011; Zhang et al. 2010].

Based on how data is placed in disks, data deduplication methods can be classified into *backward-reference deduplication* and *forward-reference deduplication*. In the former, the newest duplicate data chunks are associated with pointers that point backward to older identical data chunks. In the latter, the newest duplicate data chunks are maintained in their entirety and all past identical data chunks are associated with pointers that point forward to the newest data chunks. The forward-reference approach will provide the fastest read performance on the newest backups and is used by SEPATON's DeltaStor.[3] However, it will also introduce much more fragmentation for the old data chunks and induce more index update and metadata update operations, which degrades system performance. Therefore, most existing data deduplication methods are based on the backward-reference deduplication, especially in primary storage platforms, such as VM servers in the cloud.

2.2. Data Fragmentation Phenomenon

Data deduplication reduces the storage space by replacing redundant data chunks with pointers, which often renders originally sequential data nonsequential on disks as a side effect. Thus, a subsequent read request to the data will incur many small disk I/O operations, a phenomenon called *data fragmentation* in deduplication-based storage systems [Jones 2011; Lillibridge et al. 2013; Srinivasan et al. 2012]. However, the data fragmentation phenomenon that is much less understood and rarely studied in the literature can result in increased read response times.

To better understand the impact of data fragmentation associated with data deduplication on restore efficiency, we conduct experiments to evaluate the restore efficiency of a system with and without data deduplication by measuring the response time and power consumption of read requests to the deduplicated data. Figure 1 shows the experimental results of VM restore of a system with and without data deduplication. We can see that the restore times with deduplication are significantly higher than those without deduplication, by an average of 2.9× and up to 4.2×. The power consumption also increases by an average of 2.1× and up to 3.2×. Our trace-driven evaluations with real-world traces show that the average read response time of the system with deduplication is 45.8%, 29.7%, and 49.6% higher than that without deduplication for the web-vm, homes, and mail traces, respectively. The power consumption also increases by an average of 25.5%. The reason behind this read-performance degradation and

[5]http://www.sun.com/software/solaris/zfs.jsp

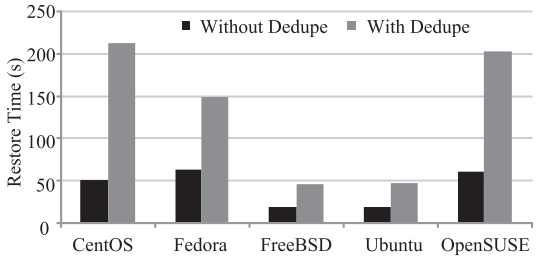


Fig. 1. Comparisons of restore times for VM disk images between systems with and without data deduplication.

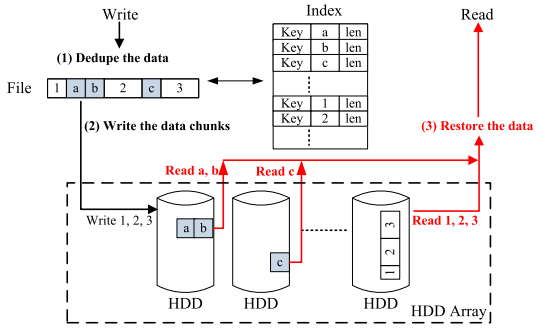
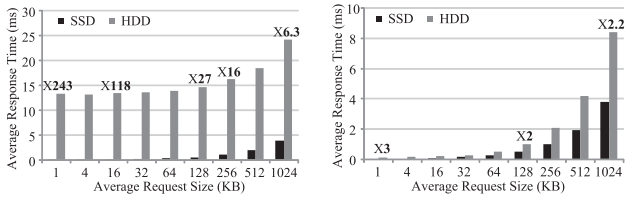


Fig. 2. Process workflow in the traditional deduplication system.

power consumption increase with data deduplication can be explained by reviewing the deduplication process, as follows.

Figure 2 shows the process workflow in the traditional deduplication process. When data arrives, it will be split into multiple data chunks, each of which is associated with a computed hash value (i.e., fingerprint). Then the index-lookup process tries to find the duplicate data chunks from the index table according to the hash values. When a duplicate data chunk is found, the *LBA* (logic block address) value of the data chunk in the index table will be obtained and kept in the metadata. Then only the unique data chunks are written, and the duplicate data chunks are replaced with the pointers in the metadata. For example, as shown in Figure 2, data chunks *a*, *b*, and *c* are the unique data chunks that are already stored. Only data chunks *1*, *2*, and *3* are written to the HDDs. The data's metadata will contain the *LBA* values of data chunks *a*, *b*, *c*, *1*, *2*, and *3*. Later, a read request to the data will first fetch the metadata to obtain the *LBA* values of the data chunks, namely, *a*, *b*, *c*, *1*, *2*, and *3*. Then three I/O requests will be generated to fetch the corresponding data chunks. When all the read requests complete, the data will be reconstructed from the fetched unique data chunks and returned to the upper layer.

However, if there is no data deduplication process enabled during the data storing process, the data will be sequentially laid out on the HDD, which then allows a subsequent read request to fetch the data with a single I/O request. Thus, while the system with deduplication saves storage space, its restore efficiency will significantly degrade due to the extra HDD I/O requests. Statistics from our experiments also indicate that the overall I/O count of the deduplication-based storage system is much higher than that of the system without deduplication, which further confirms that the read performance can be negatively affected by the data deduplication process. Moreover, the increased I/O requests involving many HDDs during system idle periods will cause



(a) Random read. (b) Sequential read.

Fig. 3. A comparison between SSD and HDD in random and sequential read performances as a function of the request size.

many more HDDs to spin up/down, which in turn consumes much more power. As a result, the read performance decreases significantly compared with storage systems without data deduplication.

On the other hand, chunk size is an important factor affecting read-performance degradation in deduplication-based storage systems [Kruus et al. 2010; Srinivasan et al. 2012]. With large chunk size, the read-performance degradation problem is alleviated to some extend, but the deduplication rate (i.e., redundancy elimination) is reduced. A recent study has shown that the deduplication rate drops from 23% to 15% when the chunk size is changed from 4KB to 8KB for engineering departments' data [Srinivasan et al. 2012]. Our analysis on the FIU traces also finds that small I/O redundancy (i.e., 4KB or 8KB) dominates the primary storage systems. Among the 4KB requests, 53.4%, 34.5%, and 92.7% are eliminated by data deduplication for the web-vm, homes, and mail traces, respectively. It has been suggested that the 4KB chunk size might be a good idea for VM platforms [Jin and Miller 2009]. These studies have revealed that small files dominate the enterprise and HPC datasets and have high deduplication rates [El-Shimi et al. 2012; Meister et al. 2012]. Eliminating these small files could not only save storage space, but also significantly improve I/O performance in primary storage systems and VM servers in the cloud. Thus, in this article, we use the 4KB chunk size in our experiments.

2.3. Understanding Flash-Based SSDs

Different from HDDs, flash-based SSDs are made of silicon memory chips and have no moving parts. Two characteristics of flash-based SSDs underlie the efficiency of the SAR design.

— *High random-read performance*. Figure 3 shows comparisons of the random and sequential read performance between SSD and HDD as a function of request size. From Figure 3(a), we can see that the random-read response times of SSD are significantly less than those of HDD, and the difference narrows as the request size increases. On the other hand, from Figure 3(b), we can see that the gap of sequential-read response times between HDD and SSD is insensitive to the request size and remains at roughly 2×. The high random-read performance of SSD is one of its main advantages over HDD [Polte et al. 2008].
— *Low power consumption*. The power consumed by SSD is only 150mW in the active mode,[6] which is significantly less than HDD which consumes about 10W in the active mode [Caulfield et al. 2009; Zhu et al. 2005]. Flash-based SSD is over two orders of magnitude more efficient than HDD in terms of I/O counts per joule [Andersen et al. 2009].

[6]Intel X25-M mainstream SATA SDD. `http://www.intel.com/design/flash/nand/mainstream/index.htm`.

On the other hand, flash-based SSDs also have two limitations: limited write-erase cycles and low random-write performance. However, in SAR, the backup data are sequentially laid out in SSDs and almost unchanged due to characteristics of content-aware storage, to be elaborated upon in Section 3. Thus, the two issues are avoided. Moreover, the cost per gigabyte of current-generation SSDs is about 20 times more than that of HDDs. It is not cost effective to replace all the HDDs with SSDs. SAR, similar to the existing optimizations [Guerra et al. 2011; Koltsidas and Viglas 2008], utilizes an appropriate and cost-effective amount of SSD capacity to significantly improve the read performance of existing deduplication-based storage systems.

2.4. Motivation and Rationale

Data deduplication has been demonstrated in the literature and by commercial products to be an effective technique in applications such as backup and archiving, and even in primary storage systems, such as VM servers that host hundreds of thousands of VMs in the cloud environment [Armbrust et al. 2009; Hansen and Jul 2010]. Thus, data deduplication has become a necessity in these applications, particularly in light of the explosive growth in the volume of digital contents and availability requirements in the cloud storage[7] [Himelstein 2011]. On the other hand, we also observe that data deduplication has the unintended effect of degrading read performance. Yet, the whole point of backup (or write in the cloud) is to restore (or read) in case of disaster (or on-demand) and reduce the storage space to save cost and network bandwidth. This makes data deduplication a two-edged sword. While its advantages have been extensively exploited in the literature and by commercial products, its disadvantages, particularly its negative effect on read performance, have been largely ignored thus far.

On the other hand, a recent study of the 113 VMs from 32 VMware ESX hosts reveals that a high degree of similarity among the VM disk images results in more than 80% reduction in storage space. Moreover, about 6% of the unique data chunks are referenced more than 10 times, and some data chunks are referenced over 100,000 times at the 4KB block size [Clements et al. 2009]. Our analysis of the mail workload, a real-world trace, indicates that the unique data chunks referenced more than 5 times amount to about 4.5% of all data chunks but account for over 42.2% of the total accesses. These unique data chunks with high reference count are likely to be accessed much more frequently, because they are shared by many files (or data blocks). Moreover, the access latency of these unique data chunks with high reference count affects the read performance directly.

These important observations, combined with the urgent need to address the read-performance problem associated with data deduplication, motivate us to propose SAR, an SSD-assisted read scheme. SAR is designed to overcome the key bottleneck of read efficiency by effectively exploiting the high random-read performance nature of SSDs and the unique data-sharing characteristic of deduplication-based storage system in the cloud.

3. SAR

In this section, we first outline the main principles guiding the design of SAR. Then we present a system overview of SAR and the key data structures in SAR, followed by a description of the selective data promotion, data demotion, and data restore processes in SAR.

[7] `http://riverbed.com/us/` and `http://www.symantec.com`

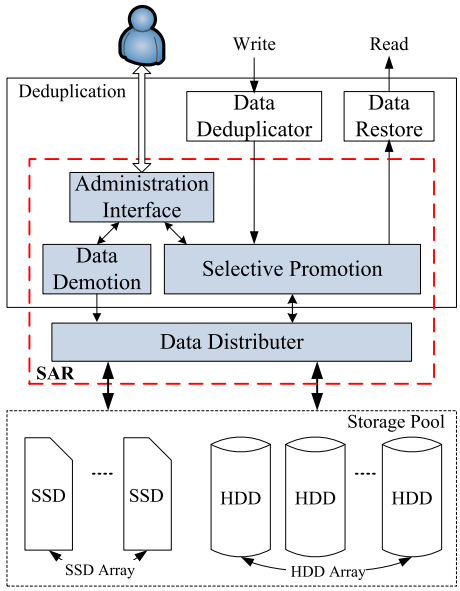


Fig. 4. System overview of SAR.

3.1. Design Principles

SAR aims to achieve high performance, high reliability, and high portability, following the design principles outlined here.

— *High performance*. To avoid any degradation in read performance and any violation of SLAs, the data restore performance must be significantly improved. SAR strives to achieve this goal by significantly reducing the expensive random-read IOs originally to the magnetic disks, by selectively storing the unique data chunks in SSDs. Read performance is improved as a result of the high random-read performance of SSDs.
— *High reliability*. With data deduplication, the backup window and storage space are reduced for the backup and cloud storage applications. However, the recovery window and restore time are also increased due to the random scattering and placement of the unique data chunks. SAR improves reliability by significantly improving the data random-read performance in order to shorten the recovery window and restore time.
— *High portability*. SAR has very little negative impact on the efficiency of the data deduplication process and is independent of and orthogonal to any particular data deduplication scheme being used. Thus it can be easily deployed in any deduplication-based storage system.

3.2. System Overview

Figure 4 shows a system overview of our proposed SAR. As shown in Figure 4, SAR interacts with the *Data Deduplicator* module and can be incorporated into any existing deduplication scheme. In SAR, the storage pool is composed of an SSD array and an HDD array. Data reliability is maintained by the redundancy schemes adopted by the arrays. When the data is deduplicated, only the unique data chunks are stored, and

the duplicate data chunks are replaced by the pointers to the unique data chunks that are already stored in the HDD array. As shown in Figure 4, SAR is located below the Data Deduplicator module and only handles the unique data chunks that are already deduplicated.

SAR has four key functional modules incorporated into the existing deduplication-based storage system: Administration Interface, Selective Promotion, Data Demotion and Data Distributor. The *Administration Interface* module provides an interface for system administrators to configure the SAR design parameters and monitor the SAR runtime status. The *Selective Promotion* module is responsible for monitoring and identifying the chunk size, sequentiality, and the reference counts of the unique data chunks that have been processed by the Data Deduplicator module, and the access popularity of the unique data chunks. Based on the information, SAR selectively stores the "hot" unique data chunks in SSD array. The Data Demotion module is responsible for evicting the "cold" unique data chunks from the SSD array when there is no free space available in the SSD array. The Data Distributor module is also responsible for appropriately directing I/O requests to either the SSD array or the HDD array.

In SAR, it is obviously not cost-effective to store all the unique data chunks in SSDs, because the cost/GB ratio of current-generation SSDs is much higher (e.g., 20X) than that of HDDs. To make the SAR design cost effective and optimally trade off between efficiency and cost, SAR exploits the characteristics of unique data chunks and the performance difference between SSDs and HDDs. In other words, SAR only stores in SSDs a small percentage of the unique data chunks but allows SSDs to absorb most small and hot random-read requests, by selecting the unique data chunks with high reference count, small size, and nonsequential in access patterns. The cost/performance analysis of SAR is discussed in Section 4.5.1.

3.3. Data Structures in SAR

There are three key data structures in SAR, namely, the index, the LRU_list, and the Map_table, as shown in Figure 5. The *index* is an important data structure in the data deduplication process. After the data is deduplicated, the index is used to identify the duplicate data chunks based on the hash values (i.e., fingerprints) of the chunks, by using hash functions such as SHA-1 and SHA-256. Since each unique data chunk has a fingerprint in the index, the space required by the index increases with the number of incoming new unique data chunks. For example, with an average chunk size of 4KB, the space required by the index is about 8GB for 1TB storage. The server cannot keep such a huge structure in memory, making it necessary to store part of the index in HDDs, where in-disk index-lookup operations can become a performance bottleneck in deduplication-based storage systems. Most existing research on data deduplication focuses on alleviating this problem [Debnath et al. 2010; Meister and Brinkmann 2010; Xia et al. 2011; Zhu et al. 2008]. However, since SAR tries to improve the restore efficiency, the in-disk index-lookup problem is out of the scope of this article and arguably orthogonal to SAR. We leave this problem as part of our future work.

In order to improve reliability and support garbage collections, exploitation of the reference management becomes popular and common in studies on deduplication-based storage systems [Bhagwat et al. 2006; Guo and Efstathopoulos 2011]. Figure 5 shows the index structure used in SAR. The *count* variable indicates the reference count of the unique data chunk. When a unique data chunk is stored, its count variable is initialized to be 1 and increased by 1 when it is referenced again. When the count variable reaches a threshold (e.g., 5), the corresponding unique data chunk will be promoted to the SSD array. The reference count is tracked on the write path.

As shown in Figure 5, another data structure used in SAR is the *LRU_list* that tracks the read-access popularity on the read path. LRU_list is an LRU list that stores the

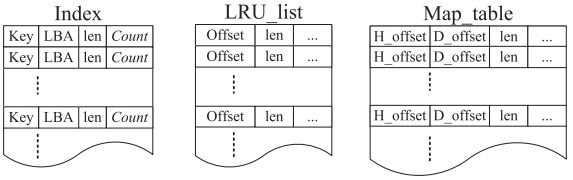


Fig. 5. Key data structures in SAR.

information of the most recent read requests, that is, the *Offset* and *Length* of the unique data chunks. Based on the LRU_list, the popular unique data chunks can be identified and promoted to the SSD array. The third data structure is the *Map_table*, as shown in Figure 5, which stores the mapping information for the unique data chunks that are promoted from the HDD array to the SSD array. The memory overhead incurred by SAR to store the data structures is acceptable and discussed in Section 4.5.2.

3.4. Selective Data Promotion

The cost per GB of current-generation SSDs is about 20 times more than that of HDDs. Based on the observations from Figure 3(a), the SAR design aims to use the SSD array and HDD array in a cost-effective way so that the unique data chunks are kept in the SSD array that can absorb most small random-read requests. The Selective Promotion module is responsible for selecting the unique data chunks to store on the SSD array. First, unique data chunks with small sizes (e.g., smaller than 128KB) and nonsequential access characteristics are selected. The size characteristics of these unique data chunks are extracted from the *len* variable of the index structure, and the nonsequential characteristics are determined by both the *len* and *LBA* variables. Then the Selective Promotion module stores the unique data chunks with high reference count in SSDs to exploit the reference locality of the unique data chunks and to reduce the expensive random I/O requests to HDDs. This is because, in deduplication-based storage systems, unique data chunks with high reference count are shared by many files and have a high probability of being accessed frequently [Clements et al. 2009; Koller and Rangaswami 2010]. On the other hand, to exploit the access locality, the unique and popular read data chunks are also promoted to the SSD array according to the LRU_list.

Figure 6 shows the process workflows of read and write requests in SAR. When a write request arrives, the Data Deduplicator module first splits the data into multiple data chunks, identifies the duplicate data chunks by comparing the fingerprints of these data chunks with those in the index structure, and increments by one the corresponding reference counts of the unique data chunks that hit in the index lookups. Then the Selective Promotion module decides whether the unique data chunks of the data (i.e., those that hit the index lookups) should be promoted to the SSD array. Note that the new unique data chunks with the reference count of 1 will be written to the HDD array directly. If the unique data chunk is selected to be promoted to the SSD array and is not in the Map_table, the unique data chunk will be written to the SSD array, and a new item will be added to the Map_table to maintain the mapping information.

When a read request arrives, the Selective Promotion module first checks whether it hits the Map_table. If it does, the read data is fetched from the SSD array. Otherwise, the Selective Promotion module checks whether the read data hits the LRU_list. If the read data is considered popular, it will be promoted to the SSD array after it is fetched from the HDD array. Once the read data is promoted to the SSD array, the

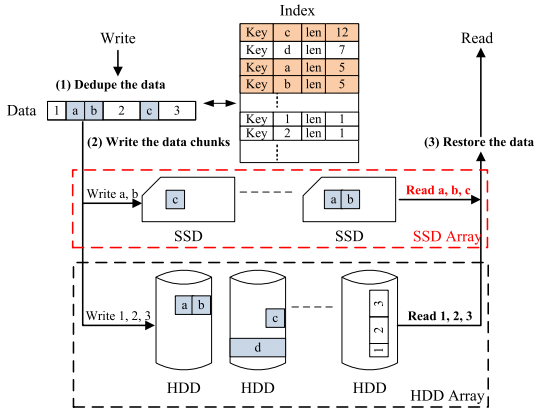


Fig. 6. Read/write process workflows in SAR.

corresponding item in the LRU_list is deleted and a new item is added to the Map_table to maintain the mapping information.

For example, as shown in Figure 6, the unique data chunks a and b are selected to be stored in the SSD array, since the corresponding reference count is 5 and their total size is small, although a and b are sequential. However, unique data chunk d with a high reference count is stored in the HDD array because of its large size.

There is little overhead for the Selective Promotion module to migrate the unique data chunks from the HDD array to the SSD array. On the write path, when the unique data chunk is chosen to be promoted, the unique data chunk is directly written to the SSD array without reading the HDD array, because the unique data chunk is already included in the write data. On the read path, if the fetched read data chunk is considered popular, it is written to the SSD array directly. Moreover, the data chunks are sequentially written to the SSD array to accelerate the write process by exploiting the high sequential-write performance of SSDs. Though the read requests to the SSD array may be random, the SSDs can achieve high random-read performance, as discussed in Section 2.3. Therefore, the performance advantages of SSDs can be fully exploited in SAR.

3.5. Data Demotion

When the number of data blocks continues to increase, the SSD array will not have free space to accommodate the newly promoted unique data chunks. Moreover, if the administrator changes the parameters, such as the reference count threshold, some data chunks will be evicted from the SSD array. Thus, a data migration module is required to evict the unique data chunks from the SSD array. The Data Demotion module in SAR is responsible for evicting the unique data chunks from the SSD array. Different from the data promotion process that needs to write the unique data chunks to the SSD array, the Data Demotion module removes the unique data chunks from the SSD array by only deleting the corresponding items in the Map_table. Thus, the Data Demotion process adds very little performance overhead to the overall system.

The eviction rules by the Data Demotion module for determining which data chunk to demote from the SSD array are based on two conditions. First, if the eviction process is triggered by the system administrator, the evicted unique data chunks are selected according to the changed system parameter values. For example, if the reference count threshold is changed from 5 to 10, all the unique data chunks with reference count below 10 will be evicted. Second, if there is no free space available in the SSD array,

a unique data chunk with a larger size and lower reference count will be evicted first. When the evicted list of data chunks is determined, the Data Demotion module evicts the data chunks from the SSD array. At the same time, the corresponding space in the SSD array will be marked as invalid for erasing.

3.6. Data Restore

When a read request arrives, the metadata (also called inode) is first checked to locate the corresponding data chunks. In the traditional deduplication process, all the data chunks are stored in the HDD array. In SAR, however, the data chunks may be located in the SSD array if they have been promoted to the SSD array, so the Data Distributor module will redirect the read request to the SSD array according to the Map_table. If the request hits the Map_table, it will be served by the SSD array. When all the data chunks have been retrieved, the Data Restore module reconstructs them and returns the restored data to the upper layer.

For example, as shown in Figure 6, reading data will require three I/O requests, two to the SSD array and one to the HDD array (after merging). Since reading data from the SSD array is very fast, the overall read response time is determined by the read latency of the HDD array, which is the fetch time of data chunks *1*, *2*, and *3*. However, in the traditional deduplication process, all three requests will be forwarded to the HDD array, which results in significantly longer user response time.

4. PERFORMANCE EVALUATIONS

In this section, we first describe the prototype implementation of SAR, followed by an introduction to the experimental setup and methodology. Then we evaluate the performance of SAR through both extensive trace-driven and real VM-disk-image restore experiments.

4.1. Prototype Implementation

We have implemented a prototype of SAR by integrating it into the open-source data-deduplication project called SDFS.[4] SDFS is a file system that provides the inline deduplication for applications. It is applicable for services such as backup, archiving, VM primary, and secondary storage. SDFS is composed of three basic components: SDFS volume, SDFS file system service, and DSE (deduplication storage engine). The SDFS volume is a mounted file system presented to the operating system and can be shared by SAMBA or NFS. The SDFS file system service provides a typical POSIX interface of the deduplicated files and folders to volumes. It stores the metadata for the files and folders and manages the file mappings that identify the data locations to the deduped/undeduped chunk mappings. Within SAR, the file mappings are also related to the SSD array in addition to the HDD array. DSE is responsible for storing, retrieving, and removing all deduplicated data chunks.

In traditional deduplication-based storage systems such as SDFS, the unique data chunks are stored on HDDs and indexed for retrieval with an in-memory hash table. In SAR, we extend the store rules by selectively promoting some unique data chunks to the SSD array. Since SDFS does not maintain the reference count for each unique data chunk, the hash index structure *HashChunk* in SDFS is also extended with the *count* variable to indicate the reference counts of the unique data chunks. Setting and retrieving the *count* value are exported by the *setCount* and *getCount* functions.

SDFS has a flexible configuration management file that can set the chunk size, store locations, and other parameters. The configuration management file is valid on the operation of the *mkfs.sdfs* command. In order to achieve a high deduplication ratio [Clements et al. 2009; Gupta et al. 2008; Jin and Miller 2009], we set the chunk

Table I. The Trace Replay Period

Traces	*Start*	*End*	*Length*	*System Size*
Web-vm	1:30	12:25	10.9 Hours	70 GB
Homes	15:50	19:10	3.3 Hours	500 GB
Mail	18:40	23:45	5.1 Hours	470 GB

Table II. The SSD Space Setting of SAR

Trace	**SSD space size**		
	SAR (1)	***SAR (2)***	***SAR (3)***
Web-vm	100MB	500MB	2GB
Homes	100MB	500MB	2GB
Mail	1GB	4GB	8GB

size to 4KB in the VM-disk-image restore experiments and store the hash index in the SSD by setting the *chunk-store-hashdb-location* parameter be the SSD volume. SDFS uses a fixed chunk size, which is sufficient for VM disk images [Jin and Miller 2009].

4.2. Experimental Setup and Methodology

We use both trace-driven and real-application-driven experiments to evaluate the efficiency of our proposed SAR. In real application evaluations, SAR is integrated into SDFS. We compare three systems: (1) an SAR-based deduplication-based storage system (SAR), (2) a deduplication-based storage system without SAR (Dedupe), and (3) a conventional system without deduplication (Native).

In the trace-driven experiments, the three traces were obtained from the SyLab of FIU [Koller and Rangaswami 2010] and cover a duration of three weeks. They are collected from a virtual machine running two Web servers (web-vm), a file server (homes), and an email server (mail), respectively. Each request in the traces includes the hash value of the requested data. In order to obtain the reference count information, we first gather the statistics of the reference count for each data chunk and apply them to the subsequent workload. For example, in order to obtain the reference count of the first request on the eighth day, we analyze the statistics of the first seven days' workloads to find how many times a data chunk has been referenced, and set the corresponding reference count for that request.

We replay the traces at the block level and evaluate the user response time of the storage device. It is time consuming to replay the whole day's trace, so we chose to replay the eighth day's trace with both burst periods and idle periods. The replayed period of the three traces are shown in Table I. Moreover, because the footprints of the three traces are small and different, we adjust the SSD space size to 100MB, 500MB, and 2GB for the web-vm and homes traces, and 1GB, 4GB, and 8GB for the mail trace in the experiments to activate the data-demotion functionality. These three SSD configurations form the basis for the three SAR systems, SAR (1)–(3), used throughout our evaluation in this section, as summarized in Table II. The SSD space is warmed up with the first seven days' traces and part of the eighth day's trace immediately preceding the start of the replay.

For the VM-disk-image restore evaluations, we downloaded 62 pre-made VM disk images from VMware's virtual appliance marketplace[8], Bagvapp's Virtual Appliances[9], and Thoughtpolice[10] with different guest OSs, applications, and versions, such

[8]http://www.vmware.com/appliances/

[9]http://bagside.com/bagvapp

[10]http://www.thoughtpolice.co.uk/vmware/

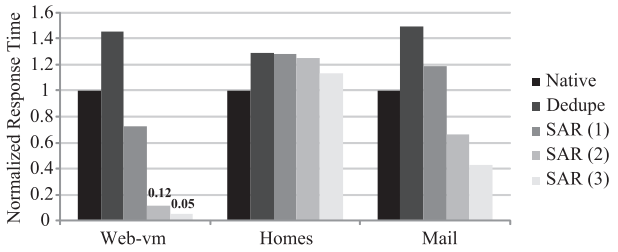


Fig. 7. Normalized read performance for the three traces.

as CentOS, Debian, Fedora, FreeBSD, OpenSUSE, and Ubuntu. The total size is about 178 GB, and the deduplication ratio is 48.9% with 4KB chunk size. In the VM-disk-image restore evaluations, we also use the *iodump* utility to collect disk I/O operations.

All our experiments are conducted on a single system with an Intel Xeon X3440 CPU and two HighPoint RocketRAID 2640 SAS/SATA controllers. In the system, a SAMSUNG 250GB HDD is used to host the operating system (Ubuntu Linux 2.6.35) and other software, and a RAID5 disk array consisting of eight HDDs (WDC WD1600AAJS) serves as the persistent storage for storing data chunks. The SSD device is Intel X25-M 80GB SSD (model SSDSA2MH080G2K5).

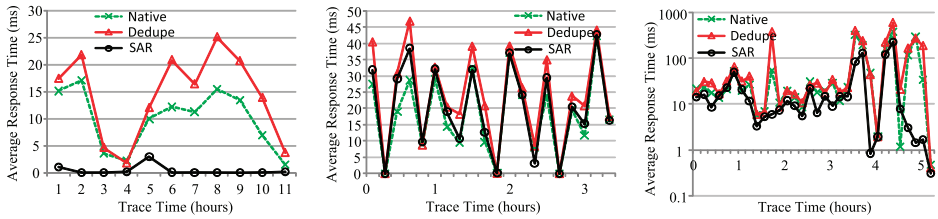
4.3. Trace-Driven Evaluations

Figure 7 shows the normalized average response times of the different schemes under the three traces. First, we observe that Dedupe increases the average response times of the Native system by 45.8%, 29.7%, and 49.6% for the web-vm, homes, and mail traces, respectively. The reason for this performance degradation was explained in Section 2.2. For the three traces, the average read response times increase because read requests require many more disk seeks and processing in the Dedupe system, compared with the Native system.

Second, SAR improves the average response-time performance significantly with a speedup of 28.18, 1.14, and 3.44, respectively, over the Dedupe system under the three traces when SAR uses the largest SSD space size in the SAR configurations (Table II), resulting in an average speedup of 5.8 over the Dedupe system. The significant improvement achieved for the web-vm and mail traces is attributed to the fact that a large portion of I/O requests are performed on the SSD, 95.88% and 49.59% for the web-vm and mail traces, respectively. However, for the homes trace, only 14.51% of I/O requests are performed on the SSD, resulting in a performance improvement of only 14.1%, the reason being that for the homes trace, the most unique data chunks are referenced fewer times, a fact identified in our workload analysis.

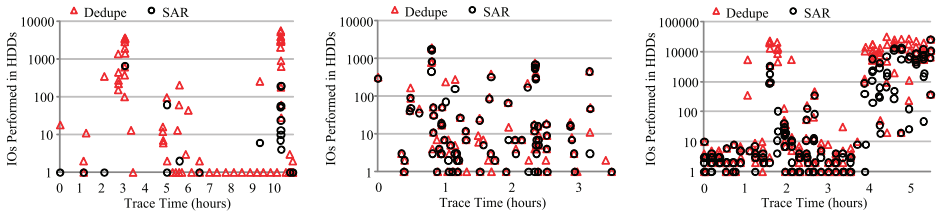
Third, we note that the performance of SAR is very sensitive to the SSD space size, where with the increased SSD space size, the average read response time reduces significantly for the web-vm and mail traces, the reason being that with a larger SSD space size, the more popular unique data chunks can be stored in the SSD, thus allowing the random-read efficient SSD to absorb many more random read requests. For the homes trace, however, the response times remain almost unchanged when increasing the SSD space size, because there is little locality in the trace, and the most unique data chunks are referenced less than five times from our workload analysis.

We also plot the average response times as a function of the trace replay time for the three traces, as shown in Figure 8. In these experiments, SAR uses the largest SSD space size, which is 2GB for the web-vm and homes traces, and 8GB for the mail trace. From Figure 8, we can see that SAR reduces the average response time significantly for the web-vm trace, the reason being that a large number of I/O requests are redirected



(a) Web-vm. (b) Homes. (c) Mail.

Fig. 8. The average response times as a function of the trace replay time. Note Y axes in (c) is log-scaled.



(a) Web-vm. (b) Homes. (c) Mail.

Fig. 9. The number of I/O requests performed in HDDs as a function of the trace replay time. Largest SSD space is used in the experiments.

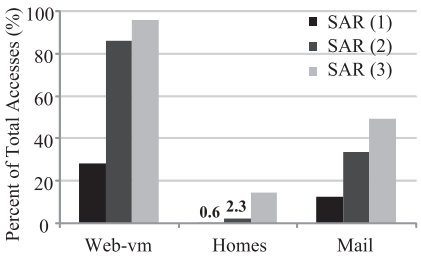


Fig. 10. Percentage of I/O requests performed on the SSD as a function of the SSD space size.

from HDDs to SSDs with SAR. Moreover, SSD has much higher random-read performance than HDD, as shown in Figure 3(a). For example, the average response time of a 4KB-request random-read is about 0.06ms for SSD, much less than the 13.29ms for HDD.

Figure 9 shows the number of I/O requests that are performed in the HDD array as a function of the trace replay time of the three traces for Dedupe and SAR. From Figure 9, we can see that SAR reduces the HDD I/O requests significantly for the web-vm and mail traces. SAR translates many HDD I/O requests to SSD I/O requests, thus significantly reducing the I/O queue length for HDDs, which enables SAR to significantly reduce the average response times of the Dedupe system. Moreover, directly reducing the HDD I/O requests during the system bursty periods improves the average response times. In other words, SAR improves the average read response times by redirecting a large number of HDD-bound I/O requests to SSDs.

To better understand the performance impact of the SSD space size in SAR, we plot the percentage of I/O requests that are performed in the SSD array as a function of the SSD space size, as shown in Figure 10. From Figure 10, we can see that with the increased SSD space size, the number of I/O requests that are performed on the SSD array increases significantly, leading to significant reductions in the average response

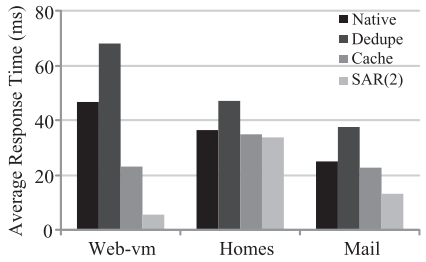


Fig. 11. The performance comparison between Native, Dedupe, Cache, and SAR schemes under the three traces.

Table III. VM Disk Images for Restore Evaluation

VM disk images	*Size (GB)*	*Dedupe. ratio*
Centos-5.1	2.7	81.1%
Fedora-12	3.5	63.3%
FreeBSD-7.0	1.2	56.4%
Ubuntu-10.10	1.0	49.3%
OpenSUSE-11.2	3.4	24.6%

times of the web-vm and mail traces. For example, the percentage of the I/O requests performed on the SSD array is 28.4% when the SSD space size is 100MB and jumps to 95.9% when the SSD space size is increased to 2GB for the web-vm trace. However, for the homes trace, when the SSD space size is less than 500MB, the percentage of I/O requests performed on the SSD array is less than 3%. Thus, the performance impact of the increased SSD space size is insignificant for the homes trace.

In order to compare SAR with the flash-based disk cache scheme, we also set up a system based on an SSD-HDD hybrid architecture, where SSD serves as the LBA- and content-based cache for HDDs (Cache). For the Cache scheme, we set the SSD size to the same as that for SAR(2) in the experiments: 500MB for web-vm and homes traces, 4GB for mail trace. Figure 11 shows the results. We can see that for web-vm and mail traces, SAR is significantly better than the Cache scheme by 75.6% and 41.9% in terms of average response times, respectively, the reason being that SAR captures the data popularity when the data is written to the disks. The Cache scheme needs the access history information to identify the popular data and move them to the SSDs. Moreover, data blocks in backup and archiving applications are typically one-shot to read, so there are no past read traces to go by for the Cache scheme. For homes trace, the performance difference is small, since there is little locality in the homes trace, and the most unique data chunks are referenced few times.

4.4. VM Restore Evaluations

In the VM-disk-image restore experiments, we selected five VM disk images to restore. The five VM disk images are of different sizes and deduplication ratios, as shown in Table III. In order to obtain accurate results, the RAID device is remounted and reformatted after the evaluation for each scheme to prepare for the next experiment. Since the dataset is small in our experiments, SAR puts the unique data chunks with the reference count of 2 in the SSD.

Figure 12 shows the restore performance results among the different schemes for the five VM disk images. From Figure 12, we can see that the deduplication process affects the restore performance significantly. For example, the Dedupe system increases the restore time by up to 4.2× for the Centos-5.1 VM disk image and 3.3× for the OpenSUSE-11.2 VM disk image over the Native system. In contrast, SAR only

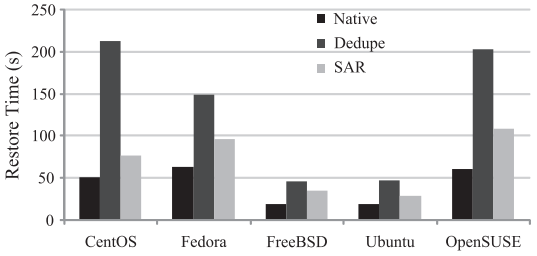


Fig. 12. Restore times of the different schemes in the VM restore evaluations.

increases the restore time by 60.0% on average over the Native system, which is a reduction of 183.4% on average from the traditional deduplication system without SAR. The reason for the longer restore time in SAR than in the Native system is that, for the VM disk image restore, the requests are sequential and the files are large, which is different from the trace-driven experiments where most requests are small. The good sequential-read performance of HDDs enables the Native system to achieve good restore performance. With deduplication both in the Dedupe and SAR systems, a large file is split into multiple small data chunks, and thus reading the large file entails many random-read requests to multiple small nonsequential data chunks.

However, SAR improves the restore performance significantly over the Dedupe system by an average of 83.4% and up to 176.6% for the CentOS-5.1 VM disk image. Thus, SAR improves the overall restore performance over the Dedupe system. For example, the restore time reduces from 213s to 77s for the CentOS-5.1 VM disk image. SAR improves the restore time by significantly reducing the number of read requests that are performed on HDDs. For example, the total number of read requests that are performed on the HDD array are 36,961, 118,509, and 79,365, respectively, for the Native, Dedupe, and SAR systems in the CentOS-5.1 VM-disk-image restore operations. Note that the number of HDD read requests is not consistent with the total number of I/O requests for the deduplication-based system that uses 4KB chunk size, the reason being that the merge functions in the I/O path have coalesced the multiple 4KB sequential I/O requests into a large read request. However, we still notice substantial increase in the large number of HDD read requests, about 2.2×, for the deduplication-based systems over the Native system. On the other hand, SAR redirects some HDD read requests to the SSD array that has high random-read performance.

In the experiments, we also collected statistics on the deduplication throughput measured for different schemes. For the Native system, the throughput is the bandwidth achieved by writing the VM disk images. SAR only reduces the deduplication throughput of the Dedupe system by 4.1%. The main reason for this degradation is the extra index update operations performed on the SSD. However, since most index update operations are processed in memory and updated to the SSD in the background, the performance impact is small in SAR. Moreover, existing deduplication storage systems already integrate the reference count variable in the index structure for reliability and garbage collections [Bhagwat et al. 2006; Guo and Efstathopoulos 2011]. Thus, SAR introduces no additional overhead for the index update operations on top of the Dedupe system.

4.5. Overhead Analysis

There are two overhead issues that must be addressed when integrating SAR into existing deduplication-based storage systems: SSD cost considerations and memory overhead.

4.5.1. SSD Cost Considerations. As shown in the trace-driven experiments, SAR incurs the highest cost when SAR uses the largest SSD space size. For example, the SSD storage space required is about 2.86% of the whole storage for the web-vm trace. Since SSDs are 20 times more expensive than HDDs, the cost of the overall system increases about 1.57 times. However, the corresponding performance improvement of SAR is 28.18 times for the web-vm trace. Thus, the performance/cost ratio of SAR is 17.95 times higher than the Dedupe system for the web-vm trace. We believe that the significant restore performance improvement by SAR justifies the additional investment to upgrade deduplication-based storage systems. Moreover, from these results, it is clear that the overall read performances are both inversely proportional to the SSD storage space used. Though larger SSD storage space is needed at a much higher cost, it buys much more improved read performance. Therefore, it is possible to strike a sensible balance between the performance and cost by appropriately adjusting the SSD space size.

On the other hand, in storage systems, while the storage bandwidth can be improved by purchasing many extra disks, the read response times cannot be improved significantly due to the mechanical characteristics of HDDs. Currently, it is not cost effective to replace all the HDDs with SSDs. SAR improves the read response times significantly by adding a few SSDs with relatively small additional cost. Moreover, in dedupv1 [Meister and Brinkmann 2010] and ChunkStash [Debnath et al. 2010] systems, the SSD is already integrated into the deduplication-based storage systems to accelerate index lookup operations. SAR simply adds another dimension to their approaches to achieve significant improvement of read efficiency.

4.5.2. Memory Overhead. SAR requires extra memory space on top of traditional deduplication-based storage systems in the forms of LRU_list and Map_table. Traditional deduplication-based storage systems also need memory space to store the hash index, so there is no extra memory overhead to store the hash index for SAR.

To prevent data loss, SAR uses nonvolatile memory to store Map_table. The amount of memory consumed is the largest when the SSD space is 8GB with the 4KB request size for the mail trace. Each item in Map_table consumes 24 bytes. In this setting, the maximum nonvolatile memory overhead is 48MB. Different from Map_table, SAR uses volatile memory to store LRU_list, because the loss of LRU_list will not incur data loss. Each item in LRU_list consumes 16 bytes. The volatile memory space overhead for storing LRU_list is about 16MB for one million items, which is sufficient in our evaluations. Thus, the maximal memory overhead in our evaluations is 64MB for SAR. However, with the rapid increase in memory size and decrease in cost of nonvolatile memory, this memory overhead is arguably reasonable and acceptable to end-users.

5. RELATED WORK

Exploiting workload reference locality and the high random-read performance of SSDs to selectively store unique data chunks to speed up restore performance in deduplication-based storage systems is the key contribution of our work. We briefly review the published work most relevant to our SAR research from the following aspects: data deduplication, read performance with deduplication, and SSD-assisted performance optimization.

Data deduplication is an active research area in both the industry and academia. Existing research work on deduplication generally focuses on how to improve deduplication efficiency by solving the index-lookup disk bottleneck problem and how to find the duplicate data blocks. DDFS [Zhu et al. 2008], Sparse Indexing [Lillibridge et al.

2009], Bimodal CDC [Kruus et al. 2010], and SiLo [Xia et al. 2011] focus on accelerating the chunk-lookup performance and reducing the RAM requirement. Venti [Quinlan and Dorward 2002] and Foundation [Nath et al. 2006] focus on a single-node system, while HYDRAstor [Ungureanu et al. 2010] and Super-chunk Routing [Dong et al. 2011] are proposed to apply data deduplication to a global system with multiple storage nodes. Data deduplication technology is widely applied in backup and archiving applications to shorten the backup window and save network bandwidth and storage space, while Difference Engine [Gupta et al. 2008] and DEDE [Clements et al. 2009] apply data deduplication in VM servers to save storage space. Other studies utilize data deduplication technology to improve I/O performance. The I/O Deduplication [Koller and Rangaswami 2010] and LPU [Ren and Yang 2010] schemes use the combined content-based and sector-based cache to improve I/O performance. SAR, being independent of and orthogonal to any specific data deduplication scheme, can be easily incorporated into any of them to improve their read performance.

In contrast, the problem of degraded read efficiency caused by data deduplication has by and large been ignored by the preceding studies. In evaluations of Foundation [Rhea et al. 2008], the authors also noticed that fragmentation induced by data deduplication is a visible problem that slow downs read efficiency, but they left it as future work. They suggested using the defragmentation technique to copy the blocks of one snapshot into a contiguous region. However, this technique typically requires numerous I/O requests as well as frequent updates to the index, which causes significant processing overhead and reduces system performance. HydraFS [Ungureanu et al. 2010] supports high-throughput read performance by sacrificing some write performance. In HydraFS, the block store API allows the caller to specify a stream hint for every block write, and then attempts to co-locate blocks with the same stream hint by delaying the writes until a sufficiently large number of blocks arrive with the same hint. However, this optimization only improves the performance of read requests addressed to the grouped blocks. The read requests addressed to the unique data chunks located on different HDDs still require many expensive disk seeks. In fact, read requests addressed to unique data chunks with high reference count are the key performance bottleneck caused by data deduplication. Our proposed SAR aims to address this key performance bottleneck and can be well incorporated into HydraFS to further improve system performance.

Recently, the data fragmentation problem associated with data deduplication has attracted eyes from the storage research community. Sequence-based deduplication [Jones 2011] and iDedup [Srinivasan et al. 2012] are two capacity-oriented data deduplication schemes that target primary storage systems by exploiting spatial locality to only selectively deduplicate the consecutive file data blocks. Their schemes try to alleviate the data fragmentation problem by selective data deduplication to only deduplicate the sequence and large data blocks. Lillibridge et al. studied three techniques, increasing cache size, container capping, and using a forward assembly area, for alleviating the data fragmentation problem in backup applications [2013]. CABdedupe [Tan et al. 2011] attempts to improve both backup and restore performance for the cloud backup service using data deduplication technology. Different from CABdedupe, SAR targets the read degradation problem induced by data deduplication, thus allowing SAR to potentially coexist and complement CABdedupe to further improve the restore efficiency of the cloud backup services.

There is a large body of research conducted on how to improve storage system performance with the help of SSDs (or flash memory). Gordon [Caulfield et al. 2009] and FAWN [Andersen et al. 2009] are two cluster storage systems that utilize the flash memory as the storage device to improve system performance and energy efficiency. MixStor [Kim et al. 2008] employs both SSDs and HDDs in an enterprise-level storage

system. In MixStor, the SSD arrays and HDD arrays are independent, and requests are distributed based on their characteristics at the application level. Dedupv1 [Meister and Brinkmann 2010] and ChunkStash [Debnath et al. 2010] are deduplication-based storage systems that utilize flash-based SSDs to store the hash index, thus improving write throughput by accelerating the index-lookup process. However, the unique data chunks are stored in HDDs, thus their read performance is also degraded. Different from these, SAR incorporates flash-based SSDs into deduplication-based storage systems for a very different reason of improving read performance. Therefore, SAR is orthogonal to and can be incorporated into dedupv1 and ChunkStash to boost the restore efficiency of deduplication-based storage systems.

6. CONCLUSION

Data deduplication is a two-edged sword in storage systems. On the one hand, it has been demonstrated to be an effective technique in shortening the backup window, and saving network bandwidth and storage space, as evidenced by commercial products in not only backup and archiving but also primary storage systems, such as VM servers in the cloud. On the other hand, the data fragmentation caused by data deduplication degrades read performance. This article proposes a solution, SAR, for improving the read performance by selectively storing in SSDs unique data chunks with high reference count, small size, and nonsequential workload characteristics. SAR effectively exploits high random-read performance properties of SSDs and the unique data-sharing characteristics of deduplication-based storage system. With SAR, many read requests to multiple HDDs are replaced by read requests to SSDs, thus significantly improving read efficiency. Extensive trace-driven and VM-disk-image restore evaluations of the SAR prototype implementation show that SAR outperforms the traditional deduplication-based storage system significantly in terms of the average read response times.

In conclusion, this article makes the following main contributions.

— From the quantitative experimental data, we have verified that read efficiency, much like write efficiency, is critical to overall performance of deduplication-based storage systems, especially for storage systems which use HDDs as storage devices.
— We propose SAR, which selectively stores in SSDs the hot unique data chunks with high reference count, small size, and nonsequential workload characteristics, to effectively exploit the unique data-sharing characteristic of deduplication-based storage system and the high random-read performance of SSDs.
— We conduct comprehensive experiments on our lightweight prototype implementation to evaluate SAR efficiency. The results show that SAR improves read performance significantly.

SAR is an ongoing research project, and we are currently exploring several directions for future work. One is to conduct more detailed experiments to measure the prototype of SAR with larger datasets, multiple disks, and in other applications, such as the backup application. Another is to build a power measurement module to evaluate the energy efficiency of the proposed SAR. Because energy efficiency is also an important system design factor, we believe that SAR will improve energy efficiency in large-scale data centers over traditional deduplication-based storage systems.

ACKNOWLEDGMENTS

We thank the SyLab in FIU for providing us with the I/O traces.

REFERENCES

Andersen, D. G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. 2009. FAWN: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. 2009. Above the clouds: A Berkeley view of cloud computing. Tech. rep. USB/EECS-2009-28, University of California, Berkeley.

Bhagwat, D., Pollack, K., Long, D., Schwarz, T., Miller, E., and Pâris, J. 2006. Providing high reliability in a minimum redundancy archival storage system. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'06)*.

Caulfield, A., Grupp, L., and Swanson, S. 2009. Gordon: Using flash memory to build fast power-efficient clusters for data-intensive applications. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*.

Clements, A. T., Ahmad, I., Vilayannur, M., and Li, J. 2009. Decentralized deduplication in SAN cluster file systems. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'09)*.

Debnath, B., Sengupta, S., and Li, J. 2010. ChunkStash: Speeding up inline storage deduplication using flash memory. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'10)*.

Dong, W., Douglis, F., Li, K., Patterson, H., Reddy, S., and Shilane, P. 2011. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*.

El-Shimi, A., Kalach, R., Kumar, A., Oltean, A., Li, J., and Sengupta, S. 2012. Primary data deduplication - Large scale study and system design. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'12)*.

ESG. 2008. Data protection survey. Enterprise Strategy Group. http://www.esg-global.com.

Guerra, J., Pucha, H., Glider, J., and Rangaswami, R. 2011. Cost effective storage using extent based dynamic tiering. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*.

Guo, F. and Efstathopoulos, P. 2011. Building a high-performance deduplication system. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'11)*.

Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M., and Vahdat, A. 2008. Difference engine: Harnessing memory redundancy in virtual machines. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI'08)*.

Hansen, J. and Jul, E. 2010. Lithium: Virtual machine storage for the cloud. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SOCC'10)*.

Himelstein, M. 2011. Cloudy with a chance of data reduction: How data reduction technologies impact the cloud. In *Proceedings of SNW Spring 2011*.

Jin, K. and Miller, E. L. 2009. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of the Israeli Experimental Systems Conference (SYSTOR'09)*.

Jones, S. 2011. Online de-duplication in a log-structured file system for primary storage. Tech. rep. UCSC-SSRC-11-03, University of California, Santa Cruz.

Kim, Y., Gupta, A., and Urgaonkar, B. 2008. MixedStore: An enterprise-scale storage system combining solid-state and hard disk drives. Tech. rep. CSE-08-017, Department of Computer Science and Engineering, Pennsylvania State University.

Koller, R. and Rangaswami, R. 2010. I/O deduplication: Utilizing content similarity to improve I/O performance. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*.

Koltsidas, I. and Viglas, S. D. 2008. Flashing up the storage layer. *Proc. VLDB Endow. 1*, 1, 514–525.

Kruus, E., Ungureanu, C., and Dubnicki, C. 2010. Bimodal content defined chunking for backup streams. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*.

Lillibridge, M., Eshghi, K., Bhagwat, D., Deolalikar, V., Trezise, G., and Camble, P. 2009. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proceedings of the 7th Conference on File and Storage Technologies (FAST'09)*.

Lillibridge, M., Eshghi, K., and Bhagwat, D. 2013. 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'13)*.

Meister, D. and Brinkmann, A. 2010. dedupv1: Improving deduplication throughput using solid state drives (SSD). In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*.

Meister, D., Kaiser, J., Brinkmann, A., Cortes, T., Kuhn, M., and Kunkel, J. 2012. A study on data deduplication in HPC storage systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*.

Meyer, D. T. and Bolosky, W. J. 2011. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*.

Muthitacharoenand, A., Chen, B., and Mazières, D. 2001. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*.

Nath, P., Kozuch, M. A., O'Hallaron, D. R., Harkes, J., Satyanarayanan, M., Tolia, N., and Toups, M. 2006. Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'06)*.

Nath, P., Urgaonkar, B., and Sivasubramaniam, A. 2008. Evaluating the usefulness of content addressable storage for high-performance data intensive applications. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC'08)*.

Polte, M., Simsa, J., and Gibson, G. 2008. Comparing performance of solid state devices and mechanical disks. In *Proceedings of the 3rd Petascale Data Storage Workshop (PDSW'08)*.

Quinlan, S. and Dorward, S. 2002. Venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*.

Ren, J. and Yang, Q. 2010. A new buffer cache design exploiting both temporal and content localities. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS'10)*.

Rhea, S., Cox, R., and Pesterev, A. 2008. Fast, inexpensive content-addressed storage in foundation. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'08)*.

Srinivasan, K., Bisson, T., Goodson, G., and Voruganti, K. 2012. iDedup: Latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*.

Tan, Y., Jiang, H., Feng, D., Tian, L., Yan, Z., and Zhou, G. 2011. CABdedupe: A causality-based deduplication performance booster for cloud backup services. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS'11)*.

Ungureanu, C., Atkin, B., Aranya, A., Gokhale, S., Rago, S., Całkowski, G., Dubnicki, C., and Bohra, A. 2010. HydraFS: A high-throughput file system for the HYDRAstor content-addressable storage system. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*.

Xia, W., Jiang, H., Feng, D., and Hua, Y. 2011. SiLo: A similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'11)*.

Xiao, W. and Yang, Q. 2008. Can we really recover data if storage subsystem fails? In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS'08)*.

Yang, T., Jiang, H., Feng, D., Niu, Z., Zhou, K., and Wan, Y. 2010. DEBAR: A scalable high-performance de-duplication storage system for backup and archiving. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS'10)*.

Zhang, X., Huo, Z., Ma, J., and Meng, D. 2010. Exploiting data deduplication to accelerate live virtual machine migration. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'10)*.

Zhu, B., Li, K., and Patterson, H. 2008. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)*.

Zhu, Q., Chen, Z., Tan, L., Zhou, Y., Keeton, K., and Wilkes, J. 2005. Hibernator: Helping disk arrays sleep through the winter. In *Proceedings of the ACM SIGOPS 20th Symposium on Operating Systems Principles (SOSP'05)*. ACM, New York, NY, 177–190.

Received December 2012; revised April 2013, June 2013; accepted July 2013