Leveraging Data Deduplication to Improve the Performance of Primary Storage Systems in the Cloud

Bo Mao, *Member, IEEE*, Hong Jiang, *Fellow, IEEE*, Suzhen Wu, *Member, IEEE*, and Lei Tian, *Senior Member, IEEE*

Abstract—With the explosive growth in data volume, the I/O bottleneck has become an increasingly daunting challenge for big data analytics in the Cloud. Recent studies have shown that moderate to high data redundancy clearly exists in primary storage systems in the Cloud. Our experimental studies reveal that data redundancy exhibits a much higher level of intensity on the I/O path than that on disks due to relatively high temporal access locality associated with small I/O requests to redundant data. Moreover, directly applying data deduplication to primary storage systems in the Cloud will likely cause space contention in memory and data fragmentation on disks. Based on these observations, we propose a performance-oriented I/O deduplication, called POD, rather than a capacity-oriented I/O deduplication, exemplified by iDedup, to improve the I/O performance of primary storage systems in the Cloud without sacrificing capacity savings of the latter. POD takes a two-pronged approach to improving the performance of primary storage systems and minimizing performance overhead of deduplication, namely, a request-based selective deduplication technique, called Select-Dedupe, to alleviate the data fragmentation and an adaptive memory management scheme, called iCache, to ease the memory contention between the bursty read traffic and the bursty write traffic. We have implemented a prototype of POD as a module in the Linux operating system. The experiments conducted on our lightweight prototype implementation of POD show that POD significantly outperforms iDedup in the I/O performance measure by up to 87.9 percent with an average of 58.8 percent. Moreover, our evaluation results also show that POD achieves comparable or better capacity savings than iDedup.

Index Terms—I/O deduplication, data redundancy, primary storage, I/O performance, storage capacity

1 Introduction

Data deduplication has been demonstrated to be an effective technique in Cloud backup and archiving applications to reduce the backup window, improve the storage-space efficiency and network bandwidth utilization. Recent studies reveal that moderate to high data redundancy clearly exists in virtual machine (VM) [1], [2], enterprise [3], [4], [5], [6] and high-performance computing (HPC) [7], [8] storage systems. These studies have shown that by applying the data deduplication technology to large-scale data sets, an average space saving of 30 percent, with up to 90 percent in VM and 70 percent in HPC storage systems, can be achieved [1], [5], [8]. For example, the time for the live VM migration in the Cloud can be significantly reduced by adopting the data deduplication technology [9].

The existing data deduplication schemes for primary storage, such as iDedup [5] and Offline-Dedupe [6], are capacity-oriented in that they focus on storage capacity savings and only select the large requests to deduplicate and bypass all the small requests (e.g., 4 KB, 8 KB or less). The rationale is that the small I/O requests only account for a tiny fraction of the storage capacity requirement, making deduplication on them unprofitable and potentially counterproductive considering the substantial deduplication overhead involved. However, previous workload studies have revealed that small files dominate in primary storage systems (more than 50 percent) and are at the root of the system performance bottleneck [4], [8], [10], [11]. Furthermore, due to the buffer effect, primary storage workloads exhibit obvious I/O burstiness [10], [12]. From a performance perspective, the existing data deduplication schemes fail to consider these workload characteristics in primary storage systems, missing the opportunity to address one of the most important issues in primary storage, that of performance.

With the explosive growth in data volume, the I/O bottleneck has become an increasingly daunting challenge for big data analytics [13] in terms of both performance and capacity. Recent International Data Corporation (IDC) studies indicate that in past five years the volume of data has increased by almost nine times to 7 ZB per year and a more than 44-fold growth to 35 ZB is expected in the next 10 years [14]. *Managing the data deluge on storage to support (near) real-time data analytics becomes an increasingly critical challenge for Big Data analytics in the Cloud, especially for VM platforms where the sheer number and dominance of small files overwhelm the I/O data path in the Cloud [11].*

- *B. Mao is with the Software School of Xiamen University. E-mail: maobo@xmu.edu.cn.*
- *H. Jiang is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE. E-mail: jiang@cse.unl.edu.*
- *S. Wu is with the Computer Science Department of Xiamen University, Xiamen, Fujian, China. E-mail: suzhen@xmu.edu.cn.*
- *L. Tian is with Tintri, Mountain View, CA, USA. E-mail: leitian.hust@gmail.com.*

Manuscript received 27 June 2014; revised 16 Apr. 2015; accepted 21 June 2015. Date of publication 12 July 2015; date of current version 16 May 2016. Recommended for acceptance by G. Min.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2455979

Moreover, our workload analysis, detailed in Section 2.1, shows that data redundancy on the critical I/O path is much more pronounced than on the storage devices, largely due to the high temporal locality of small I/O requests. This suggests that, if such redundant I/Os can be removed from the critical I/O path, the performance bottleneck of a primary storage system may be significantly alleviated, if not removed. Thus, we argue that, for primary storage systems in the Cloud, it may be at least as important, if not more so, to deduplicate the redundant I/Os on the critical I/O path for the sake of performance as to deduplicating redundant data on storage devices for the sake of capacity savings.

On the other hand, our experimental studies suggest that directly applying data deduplication to primary storage systems will likely cause space contention in the main memory and data fragmentation on disks. This is in part because data deduplication introduces significant index-memory overhead to the existing system and in part because a file or block is split into multiple small data chunks that are often located in non-sequential locations on disks after deduplication. This fragmentation of data can cause a subsequent read request to invoke many, often random, disk I/O operations, leading to performance degradation. Our preliminary evaluations on the VM disk images reveal that the restore times with deduplication are much higher than those without deduplication, by an average of 2.9× and up to 4.2× [15]. These two problems will be particularly acute with the deployment of the data deduplication technology into the primary storage systems for big data analytics in the Cloud.

To address the important performance issue of primary storage in the Cloud, and the above deduplication-induced problems, we propose a Performance-Oriented data Deduplication scheme, called POD, rather than a capacity-oriented one (e.g., iDedup), to improve the I/O performance of primary storage systems in the Cloud by considering the workload characteristics. POD takes a two-pronged approach to improving the performance of primary storage systems and minimizing performance overhead of deduplication, namely, a request-based selective deduplication technique, called Select-Dedupe, to alleviate the data fragmentation and an adaptive memory management scheme, called iCache, to ease the memory contention between the bursty read traffic and the bursty write traffic.

More specifically, Select-Dedupe takes the workload characteristics of small-I/O-request domination into the design considerations. It deduplicates all the write requests if their write data is already stored sequentially on disks, including the small write requests that would otherwise be bypassed from by the capacity-oriented deduplication schemes. For other write requests, Select-Dedupe does not deduplicate their redundant write data to maintain the performance of the subsequent read requests to these data. iCache dynamically adjusts the cache space partition between the index cache and the read cache according to the workload characteristics, and swaps these data between memory and back-end storage devices accordingly. During the write-intensive bursty periods, iCache enlarges the index cache size and shrinks the read cache size to detect much more redundant write requests, thus improving the write performance. During the read-intensive bursty periods, on the other hand, the read cache size is enlarged to cache more hot read data to improve the read performance. Thus, the memory efficiency is maximized.

The prototype of the POD scheme is implemented as an embedded module at the block-device level and a subfile deduplication approach is used. To examine the net effect of the POD scheme, in our trace-driven evaluation we use the block-level traces that were collected beneath the memory buffer cache so that the caching/buffering effect of the storage stack is already fully captured by the traces. In other words, all the small I/O requests in our evaluation are issued from the buffer cache to the block devices after the former has processed the filesystem-issued requests. The extensive trace-driven experiments conducted on our lightweight prototype implementation of POD show that POD significantly outperforms iDedup in the I/O performance measure of primary storage systems without sacrificing the space savings of the latter. Moreover, as an application of the POD technology to a background I/O task in primary cloud storage, it is shown to significantly improve the online RAID reconstruction performance by reducing the user I/O intensity during recovery.

The rest of this paper is organized as follows. Background and Motivation are presented in Section 2. We describe the POD architecture and design in Section 3. The performance evaluation is presented in Section 4. We review the related work in Section 5 and discuss insights we obtained through this work and future work in Section 6. The conclusion of this paper is presented in Section 7.

2 Background and Motivation

In this section, we present some important observations drawn from previous and our workload analysis of primary storage, the cache partition strategy and read amplification problem associated with HDD-based data deduplication to motivate the POD study.

2.1 Workload Characteristics: Domination of Small Files and I/Os in Primary Storage

Knowing the workload characteristics is important for system design, which explains the many recent studies conducted by researchers to analyzing primary workloads [4], [8]. These studies reveal that more than 50 percent of files are smaller than 4 KB [4], [16] and 30 to 62 percent of I/O requests seen at the block level are 4 KB. In primary storage data sets, small files are the most common file size and up to 62 percent files are smaller than 4 KB [8]. These results indicate that small I/O requests dominate the primary storage workloads, which is quite different from backup and archiving applications. Moreover, these studies also found that small files have high deduplication rates by a ratio of over 20 percent, up to 80 percent for the primary data sets [8]. Our own analysis on primary workloads also finds that small I/O redundancy (i.e., 4 KB or 8 KB) dominates, as illustrated in Fig. 1 that shows the distribution of the I/O redundancy among requests of different sizes on the 15th day of the three traces collected from Florida International University (FIU traces [17]). We can see that small I/Os dominate the total requests and have the highest redundancy.

These observations and new findings suggest that the existing deduplication schemes are not suitable for the need

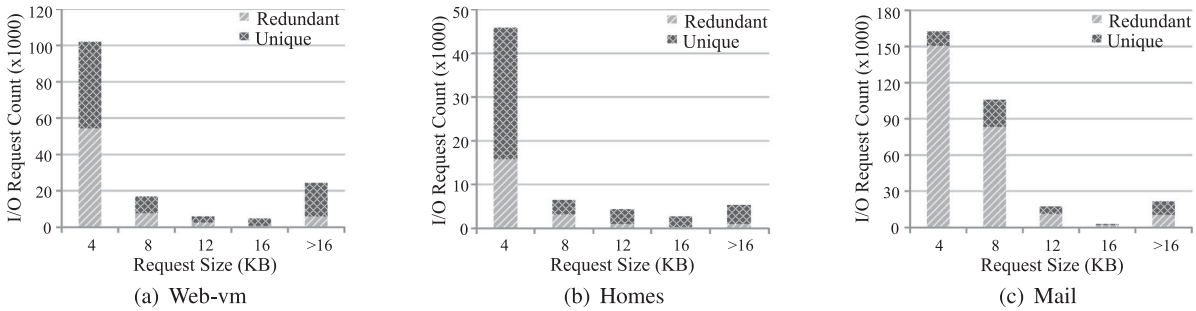

Fig. 1. Distribution of I/O redundancy among requests of different sizes on the 15th day of the traces.

of primary storage systems in the Cloud. Capacity-oriented deduplication systems, such as iDedup [5], do not deduplicate the small I/O requests because deduplicating them contributes little to the overall capacity savings. However, from the perspective of performance, small I/O requests are extremely important because they are the root cause of the performance bottleneck. Moreover, large I/O requests are mostly partially redundant. Deduplicating these partially redundant large I/O requests will cause the data fragmentation problem [5], [15], [18]. Previous studies on deduplication-based backup and archiving systems do not pay much attention to the data fragmentation problem because read requests are rare in backup and archiving environments. The data fragmentation problem can result in significantly increased read response time, a particularly detrimental side effect for primary storage where reads are common events, as elaborated in Section 2.3.

Moreover, we also found that I/O redundancy is noticeably higher than capacity redundancy. *I/O redundancy* in this context means that data blocks accessed by different write requests on the critical I/O path contain the same content. It is therefore different from data redundancy in that the latter is analyzed from the static data stored on the storage devices [4], [6], [8]. For the redundant write data, only the write data addressed to different locations may contribute to capacity savings. Fig. 2 shows the percentages of write data that are addressed to the same locations and to the different locations with the same content. While the latter indicates the data redundancy targeted by capacity-oriented deduplication schemes, it is the combination of the former and the latter that signifies the I/O redundancy. It is clear that I/O redundancy is noticeably higher than capacity redundancy, by an average of 21.9 percent among the three traces, due to the additional repeated accesses to the same locations on the storage devices by user requests as a result of their temporal locality. This new finding implies that on-line deduplication is much more effective in reducing I/O traffic than off-line deduplication [6] for primary storage workloads.

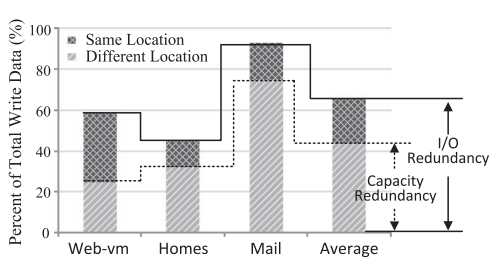

Fig. 2. I/O redundancy versus capacity redundancy for the three applications.

2.2 Index Cache versus Read Cache

Intuitively, applying the data deduplication technique to primary storage systems will introduce the necessary memory overhead required for the hash index. For example, when deduplicating 1 TB data with an average chunk size of 4 KB, the space required by the hash index table is about 8 GB. Typical servers cannot keep such a huge hash index table in memory that is also used to cache the popular data blocks to exploit the access locality. Consequently, most of the hash index entries must be stored on disks, where the in-disk index-lookup operations can become a severe performance bottleneck in deduplication-based storage systems [19].

Fig. 3 shows the read and write performances in average response time as a function of the percentage of the memory space allocated to the index cache in a deduplication-based storage system driven by the original mail trace, where the memory space is assumed to be divided between the index cache and the read cache. The experiment setup is described in Section 4. We can see that with a larger index cache, the write performance is improved because most indices are stored in memory, thus reducing the number of in-disk index-lookup operations on the write path. However, when the memory space for the read cache is reduced, the read performance is degraded due to the increased read miss ratio. Similarly, with a smaller index cache, the write performance is degraded and read performance is improved. Thus, the index cache is beneficial to the write performance and the read cache is beneficial to the read performance.

On the other hand, previous workload studies revealed that the I/O burstiness is a common characteristic in

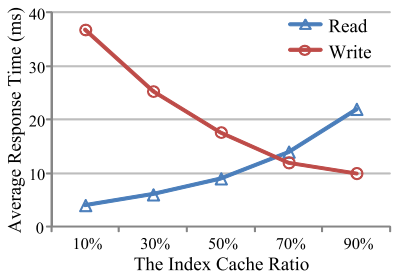

Fig. 3. Read and write performances as a function of the percentage of the memory space allocated to the index cache in a deduplication-based storage system, where the memory space is assumed to be divided between the index cache and the read cache.

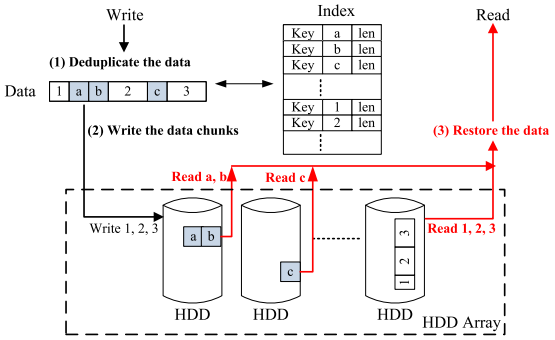

Fig. 4. An example showing the read amplification problem in a deduplication-based storage system.

primary storage systems, suggesting that read-intensive periods are interactive with write-intensive periods [16], [20]. Given that the index cache is important in improving the write performance and read cache is critical for the read performance, the I/O burstiness characteristic will likely render ineffective the fixed cache partition between the read cache and the index cache. Thus, we believe that a dynamic and adaptive cache partition strategy based on the workload characteristics seems to be a solution to the above mentioned problems.

2.3 The Read Amplification Problem

Data deduplication reduces the write traffic of primary storage systems by replacing the redundant data chunks with pointers, which often renders originally sequential data non-sequential on disks as a side effect. Thus, a subsequent read request to the data will incur many small disk I/O operations, a phenomenon we call *read amplification phenomenon* [21]. Fig. 4 shows the processing workflow in a traditional deduplication storage system. When data arrives, it will be split into multiple data chunks, each of which is associated with a calculated hash value, also known as fingerprint, that uniquely identifies and represents this data chunk. The index-lookup process then tries to find the redundant data chunks from the fingerprint index table according to the hash values. When a redundant data chunk is found, it is replaced with a pointer in the metadata. Only the unique data chunks are written to the disks. For example, as shown in Fig. 4, data chunks *a*, *b* and *c* are unique data chunks that are already stored on disks. Only data chunks *1*, *2* and *3* are written to the disks. Later, a read request to the data will need three separate disk I/O operations to reconstruct the original data, as shown in Fig. 4. However, if the storage system does not deduplicate the redundant data, the data will be sequentially laid out on the disk, which allows a subsequent read request to fetch the data with a single disk I/O operation. Consequently, while data deduplication saves storage space, the read performance will be significantly degraded due to extra disk I/O operations.

Our previous experiments [15], [21] on a RAID5 system consisting of eight disks and 62 pre-made VM disk images validated that the restore latency with deduplicationis significantly higher than that without deduplication, by an average of 2.9× and up to 4.2×. Statistics from the experiments also indicate that the overall I/O count of the deduplication-based storage system is much higher than the system without deduplication, which further confirms that the read performance can be negatively affected by the data deduplication process.

TABLE 1
Comparison between POD and the State-of-the-Art Schemes

Features	I/O Dedup [3]	iDedup [5]	Offline-Dedup [6]	POD
Capacity		✓	✓	✓
Performance	✓			✓
Small I/O				✓
Large I/O		✓	✓	✓
Cache	Static	Static	Static	Dynamic

These important observations described above, combined with the urgent need to address the small-write problem of HDD-based primary storage systems, motivate us to propose POD. POD is designed to retain the desirable advantages of the write-traffic-reducing ability of data deduplication while effectively addressing the deduplication-induced problems. Table 1 briefly compares POD with the state-of-the-art deduplication approaches. In summary, POD improves the I/O performance of primary storage systems by focusing more on small I/Os and files while retaining the capacity savings. Moreover, it manages the storage cache intelligently and dynamically according to the workload to improve the system performance.

3 The Design of POD

In this section, we first outline the main design objectives of POD. Then we present an architecture overview of POD, followed by detailed descriptions of two main POD components, Select-Dedupe and iCache, and the maintenance of data consistency.

3.1 Design Objectives

The design of POD aims to achieve the following three objectives.

- *Reducing small write traffic*-By calculating and comparing the hash values of the incoming small write data, POD is designed to detect and remove a significant amount of redundant write data, thus effectively filtering out small write requests and improving I/O performance of primary storage systems in the Cloud.
- *Improving cache efficiency*-By dynamically adjusting the storage cache space partition between the index cache and the read cache, POD efficiently utilizes the storage cache adapting to the primary storage workload characteristics.
- *Guaranteeing read performance*-To avoid the negative read-performance impact of the deduplication-induced read amplification problem, POD is designed to judiciously and selectively, instead of blindly, deduplicate write data and effectively utilize the storage cache.

3.2 POD Architecture Overview

Fig. 5 shows a system architecture overview of our proposed POD in the context of the system I/O in the Cloud.

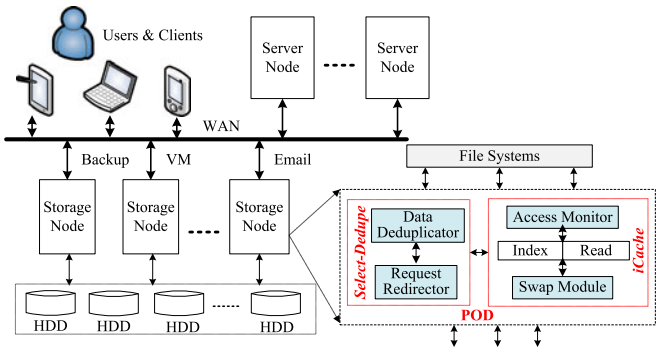

Fig. 5. System architecture of POD.

As shown in Fig. 5, POD resides in the storage node and interacts with the *File Systems* via the standard read/write interface. Thus, POD can be easily incorporated into any HDD-based primary storage systems to accelerate their system performance. Moreover, POD is independent of the upper file systems, which makes POD more flexible and portable than whole-file deduplication [4] and iDedup [5]. It can be deployed in a variety of environments, such as virtual machine images that are mostly identical but differ in a few data blocks [2].

POD has two main components: Select-Dedupe and iCache. The request-based *Select-Dedupe* includes two individual modules: Data Deduplicator and Request Redirector. The *Data Deduplicator* module is responsible for splitting the incoming write data into data chunks, calculating the hash value of each data chunk, and identifying whether a data chunk is redundant and popular. Based on this information, the *Request Redirector* module decides whether the write request should be deduplicated, and maintains data consistency to prevent the referenced data from being overwritten and updated. The *iCache* module also includes two individual modules: Access Monitor and Swap Module. The *Access Monitor* module is responsible for monitoring the intensity and hit rate of the incoming read and write requests. Based on this information, the *Swap* module dynamically adjusts the cache space partition between the index cache and read cache. Moreover, it swaps in/out the cached data from/to the back-end storage. iCache helps request-based Select-Dedupe deduplicate as many redundant data blocks as possible and improves the read performance by expanding the read cache size in face of read bursts.

3.3 Select-Dedupe

The request-based Select-Dedupe works on the write path to effectively reduce the write traffic if the write requests are redundant, and updates the Map_table accordantly. In Select-Dedupe, write requests with redundant data are classified into three categories, as shown in Fig. 6: (I) the fully redundant write requests whose write data are already sequentially stored on disks, (II) the partially redundant write requests whose write data are a mixture of redundant data chunks and new unique data chunks, where the number of redundant data chunks in each request is less than a predefined threshold (for example, 3 in our current design), and (III) the partially redundant write requests of which the number of redundant data chunks per request exceeds the

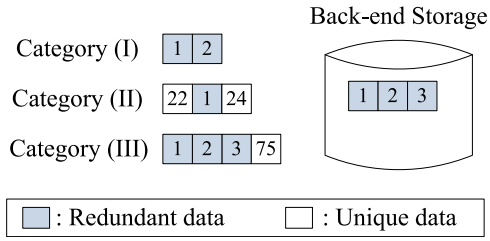

Fig. 6. Categorization of write requests in the request-based Select-Dedupe.

threshold. Select-Dedupe deduplicates the write requests belonging to category (I) and category (II), and ignores any write requests belonging to category (II). For the write requests in category (III), deduplicating the redundant data chunks only reduces the size of the write data, which only slightly improves the write performance because the write requests must still be performed on disks. And most importantly, the performance of the subsequent read requests to these data will be significantly degraded due to the read amplification problem, a conclusion confirmed by our performance evaluations in Section 4. However, for category (III) of write requests, the redundant data blocks are large and sequentially stored on disks, which helps mitigate the seek penalty in the subsequent read requests. Moreover, deduplicating these large and continuous redundant data chunks also contributes to the storage space saving, as validated in Section 4.

Fig. 7 illustrates the process flow of a write request in Select-Dedupe. There are two key data structures supporting Select-Dedupe in deduplicating and redirecting the I/O requests, and identifying the popular hash index entries, namely, *Map_table* and *Index_table*, as shown in Fig. 7. The Map_table keeps all the information of the deduplicated write requests whose write data are already stored on disks. The Index_table maintains the fingerprints of the hot data chunks already stored on disks. The mapping relationship between the items in Map_table and the items in Index_table is *m-to-1*. This means that an LBA (Logical Block Address) can only be linked to a unique and distinctive physical data block but multiple LBAs may be linked to the same physical data block.

In order to reduce the memory space and processing overhead required to store and query the huge hash index

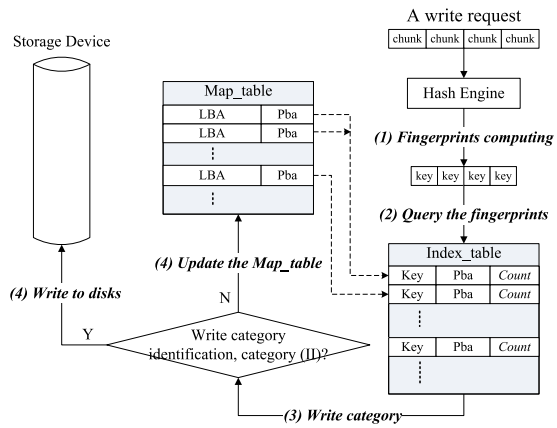

Fig. 7. The write process flow in Select-Dedupe.

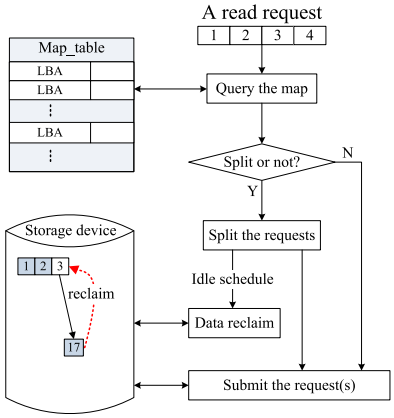

Fig. 8. The read process flow in Select-Dedupe, showing the optimization on the read path.

table, POD only stores the hot hash index entries in memory. The Index_table in our POD design is organized in an LRU form and maintains the frequency of write requests by using the *Count* variable (initialized to "0"), as shown in Fig. 7. When a write request hits the Index_table, the count value of the corresponding hash index entry is incremented, capturing the temporal locality and frequency of write requests. The *Count* variable is also used to prevent the referenced data blocks from being modified or deleted (see Section 3.5). To prevent data loss in case of a power failure, the Map_table data structure is stored in non-volatile RAM.

When a write request arrives, Select-Dedupe splits the write data into chunks that are each fingerprinted with their hash values by a hash engine (a dedicated embedded processor or the host processor). Each fingerprint is queried in the Index_table. If a match is found, meaning that the data chunk is redundant, the corresponding *Count* value is incremented. Otherwise, a new hash index entry is inserted into the Index_table. After calculating the fingerprints of the data, Select-Dedupe classifies the write request into one of the three categories described above and depicted in Fig. 6. If the write request belongs to category (II), the data is directly written to the disks without deduplicating the redundant data. Otherwise, Select-Dedupe only updates the Map_table for the redundant data blocks, and the non-redundant data blocks are written to the disk as usual. The data consistency is also checked to make sure that the referenced data is not overwritten, which is elaborated in Section 3.5.

As shown in Fig. 8, when a read request is filtered by the cache and arrives at the block device layer, Select-Dedupe first checks whether the read request hits the Map_table. If the read request does not hit the Map_table, the read request is directly submitted to the device layer. Otherwise, the address of the request will be replaced by one or more addresses according to the Map_table. Then, the newly generated read request(s) is(are) submitted to the device layer. After the completion of these read requests, the read data is reconstructed and returned to the upper layer. In our design, if a read request that hits the Map_table is split into multiple small read requests by the Request Redirector module, Select-Dedupe will reorganize these data chunks to their original sequential positions and update the Map_table during the system idle time. Thus, the performance of the subsequent read requests to these data chunks will be guaranteed.

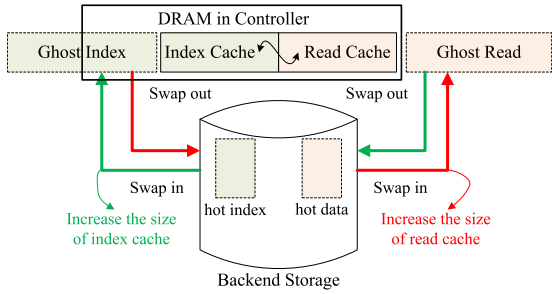

Fig. 9. The iCache structure.

3.4 iCache

The design of iCache is based on the rationale that the I/O workload of primary storage changes frequently with mixed read and write burstiness. As observed from the preliminary results in Section 2.2, we need to dynamically adjust the storage-cache space partition between the index cache and read cache adapting to the characteristics of user accesses to obtain the best overall performance. To maximize the performance of the storage cache in deduplication-based primary storage systems, the type of data that provides the largest performance benefit should be stored in storage cache. Fig. 9 shows the structure of iCache. The DRAM size in the storage controller is fixed while the index cache size and the read cache size can be dynamically resized. The maximum size of an actual cache and its ghost cache is set to be equal to the total size of the DRAM in the storage controller. iCache maintains the ghost index and ghost read caches that store only metadata whose actual data are stored on the back-end storage devices. When a victim data item is flushed from the index cache or the read data cache, its metadata is inserted into the corresponding ghost cache and the actual data is flushed to the back-end storage device. Through these metadata of the ghost caches, the cost-benefit of their actual caches can be estimated. Based on the cost-benefit values, iCache swaps in the actual data of the ghost cache with the larger cost-benefit value into the memory and swap out the data of the other cache to the back-end storage device.

The cost-benefit values are generated by read and write hits in the actual caches and the ghost caches. For a predefined interval, iCache calculates the cost-benefit values of the ghost caches and adjusts the allocation ratio between the actual index cache and read cache. Table 2 summarizes the parameters that are used to analyze the cost-benefit values and adjust the cache space size in iCache. Algorithm 1 shows the pseudo-code of the dynamic cache allocation scheme in iCache.

TABLE 2
Parameters for Analyzing the Cost-Benefit Values and Adjusting the Cache Space Partition in iCache

Notation	*Description*
N_{req}	Request number of the access sequence
N_{int}	Interval to adjust the cache allocation size
H_{gi}	Hit counts in the ghost index and index cache
H_{gr}	Hit counts in the ghost read and read cache
S_{ci}	Current size of the index cache size
S_{cr}	Current size of the read cache size

Algorithm 1. Dynamical cache allocation

```
1.1  if (N_req mod N_int) == 0 then
1.2    C_gi = H_gi/S_ci; C_gr = H_gr/S_cr;
1.3    if (C_gi > C_gr) then
1.4      S_turn = S_unit * (C_gi/C_gr);
1.5      S_i_max = S_ci + S_turn;
1.6      S_r_max = S_cr - S_turn;
1.7      if (S_r_max < S_cr) then
1.8        Inc_index_cache(S_turn);
1.9      end
1.10   end
1.11   else
1.12       S_turn = S_unit * (C_gr/C_gi);
1.13       S_r_max = S_cr + S_turn;
1.14       S_i_max = S_ci - S_turn;
1.15       if (S_i_max < S_ci) then
1.16         Inc_read_cache(S_turn);
1.17       end
1.18     end
1.19     H_gi = 0; H_gr = 0;
1.20     Update(S_ci, S_cr);
1.121 end
```

We calculate the cost-benefit value of increasing the index cache space (C_{gi}) by means of dividing the hit counts of the index cache and ghost index (H_{gi}) by the current size of the index cache (S_{ci}). The calculation of the cost-benefit value of increasing the read cache space uses the same method. At every interval (N_{int}), iCache adjusts the cache space partition between the index cache and read cache. For example, as shown in Fig. 9, the index cache occupies a larger part in the DRAM and some read data (i.e., the ghost part) is swapped out to the back-end storage device. However, if the cost-benefit value of increasing the read cache (C_{gi}) is larger than that of increasing the index cache (C_{gr}), iCache increases the read cache size by swapping in the ghost part of the read data and swapping out the ghost part of the index cache.

The two functional components in iCache, the Access Monitor and the Swap Module, work together to accomplish the iCache functions. The Access Monitor in iCache determines which cache, index cache or read cache, should be increased in size according to the current access pattern. The access pattern is measured by counting the numbers of the read and write requests that hit the corresponding caches (H_{gi} and H_{gr}). When the storage-cache space is repartitioned, the Swap Module in iCache will swap in/out the particular data from/to the memory and the back-end storage device. The swapped out index data and read data are stored on a reserved space on the back-end storage device. As described in Section 2.1, the data fragmentation problem degrades the read performance in deduplication-based primary storage systems based on HDDs. In primary storage systems, the read requests are time- and performance-critical because they are synchronized and can block the progress of applications.

In POD, the read performance is guaranteed in the following three ways. First, on the write path, Select-Dedupe does not deduplicate the write requests with scattered redundant data, such as category (II) shown in Fig. 6. The operations of the subsequent read requests to these data will be identical to those without data deduplication. Moreover, the read operations will also be optimized on the read path, as shown in Fig. 8. Second, by eliminating the redundant small write requests, such as category (I) shown in Fig. 6, the I/O queue length is reduced. The read requests on the disks can be serviced much more efficiently. Third, in the storage cache, iCache improves the read performance by dynamically increasing the read cache size in face of read bursts to improve the read cache hit ratio.

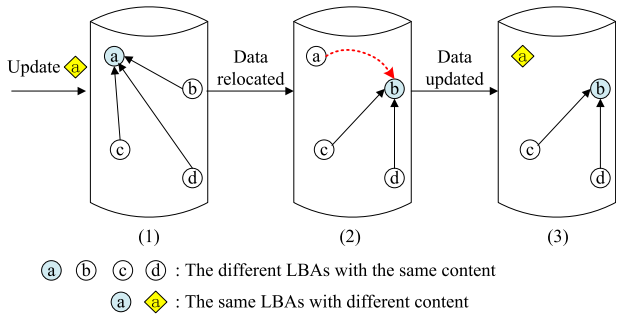

Fig. 10. Update process in POD.

3.5 Maintaining Data Consistency

Data consistency in POD mandates that: (1) the referenced data be reliably stored on disks; and (2) the key data structures not be lost in case of a power failure.

First, in order to avoid data loss caused by the update or delete operations, all the referenced data on disks must be prevented from being modified or deleted. To avoid the circular effect of the pointers, if a piece of the referenced data (the count value is not "0") needs to be updated with new data content, the original data is migrated to an alternative location that also points to the data according to the Map_table. In the meantime, all the LBAs that point to the referenced data must be updated in the Map_table. Then the original location is updated with the new data content. Thus, the referenced data is not lost and reliably stored on disks.

Fig. 10 illustrates an example of how a referenced data chunk is updated reliably. In order to update a data chunk with LBA *a* whose physical/unique data chunk is stored on the disk and referenced by three redundant data chunks with LBAs *b*, *c* and *d*, the physical data chunk is first relocated from LBA *a* to LBA *b* so that the redundant data chunks with LBAs *c* and *d* are now pointed to the data chunk with LBA *b*. Then, the updating can be safely performed in the LBA *a* with the new data. As we can see from Fig. 10, updating a data chunk results in two extra I/O operations on HDDs, which will degrade the system performance. To address this problem, such update operations are usually processed in the background during system idle periods to alleviate the performance overhead for online applications [5].

Second, since the contents of the Map_Table for Select-Dedupe must never be lost in case of a power failure, the Map_Table could be stored in an NVRAM which will incur the NVRAM space overhead. In POD, each entry in the Map_Table consumes 16 bytes. Assuming a worst case scenario of a workload of 1,000,000 I/O requests with a 50 percent deduplication ratio on average, the NVRAM space overhead will be 1,000,000 * 50% * 16 bytes = 8 MB. Moreover, the I/O requests with the same content and locations only consume one entry in the Map_Table, which will further reduce the memory overhead in real applications.

TABLE 3
The Characteristics of the Three Traces

Traces	*Write ratio*	*I/Os*	*Average Request Size (KB)*
Web-vm	69.8%	154,105	14.8
Homes	80.5%	64,819	13.1
Mail	78.5%	328,145	40.8

Our evaluations on the FIU traces in Section 4.5 also validate that the Map_Table is in general sufficiently small and thus will not incur significant hardware overhead. To further reduce the memory space consumed by the Map_Table, data compression technique can be used to compress the mapping information for the cold data blocks. Moreover, since the performance of a battery-backed RAM, a *de facto* standard form of NVRAM for storage controllers, is roughly the same as that of the main memory, the write penalty due to the Map_Table updates can be negligible.

4 Performance Evaluations

In this section, we first describe the experimental setup and methodology. Then we evaluate the performance of POD through extensive trace-driven experiments.

4.1 Experimental Setup and Methodology

We have implemented a prototype of POD as a module in the Linux operating system and use the trace-driven experiments to evaluate its effectiveness and efficiency. In this paper, we compare POD with the HDD-based storage systems without data deduplication (short for *Native*) and with traditional full data deduplication (short for *Full-Dedupe*), and the capacity-oriented scheme iDedup [5]. All the experiments were conducted on a storage node with an Intel Xeon X3440 CPU and 4 GB memory. In this system, a SAMSUNG HE253GJ SATA HDD (250 GB) is used to host the operating system (Ubuntu Linux kernel 2.6.35), Linux software RAID module (i.e., MD) and other software. Two HighPoint RocketRAID 2640 SAS/SATA controllers are used to connect eight SATA HDDs (WDC WD1600AAJS).

In the experiments, we used three traces for our trace-driven evaluation. The three traces were collected from three production systems, a virtual machine running two web-servers (web-vm), a file server (homes) and an email server (mail), covering a duration of three weeks [17]. Because the original request data have been split into several small data chunks with a fixed size (e.g., 4 KB or 512 B), the original requests are reconstructed according to their timestamp, LBA and length. In order to simulate the hash computing overhead of each data chunk, we added a 32us fingerprint-computing delay to each process of writing a 4 KB data chunk, which is an overestimation for the processors in modern controllers [22], [23]. The cache for storing the hash index table was warmed up by the first 14 days' traces. We used the 15th day of the three traces for our evaluations and the characteristics of the three traces are summarized in Table 3. Due to the different footprints of the three traces, the total memory size is set to be 100, 500, and 500 MB for the web-vm, homes, and mail traces, respectively [3].

We replayed the three traces at the block level and evaluated the user response times. The hash values of the data chunks are also included with other attributes of replayed requests. In the experiments, we also evaluate statistics of average response times for the read and write requests separately to understand the impact of different schemes on read requests and write requests.

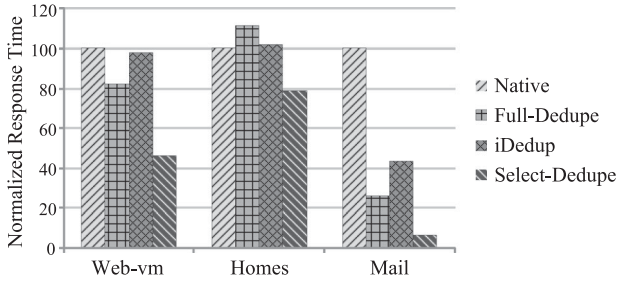

Fig. 11. The response-time performance of the different deduplication schemes normalized to the Native system, driven by the three traces for a 4-disk RAID5.

4.2 Select-Dedupe Performance

We first conducted experiments on a 4-disk RAID5 system with a stripe unit size of 64 KB. In this experiment, Full-Dedupe, iDedup and Select-Dedupe all use the fixed cache partition that allocates equal spaces to the index cache and read cache. Fig. 11 shows the response-time performance of the different deduplication schemes normalized to that of the Native system, driven by the three traces. Full-Dedupe degrades the Native system performance for the homes trace, indicating that directly applying data deduplication to primary storage systems may introduce extra performance overhead. iDedup improves the performance of Native system slightly, which indicates that capacity-oriented deduplication schemes are not effective in improving system performance. In contrast, Select-Dedupe improves the performance of the Native system by 53.9, 21.2, and 88.6 percent for the web-vm, homes, and mail traces, respectively. In order to understand the reasons behind the performance improvement of Select-Dedupe, we separate the write response times from the read response times and plot them separately in Fig. 12.

Fig. 12a shows that Select-Dedupe reduces the write response times of the Native system by 47.2, 20.2, and 91.6 percent, and those of the iDedup system by 40.2, 18.8, and 81.5 percent, for the web-vm, homes, and mail traces, respectively. The significant write-performance advantage of Select-Dedupe stems from the fact that a large number of write requests are eliminated by Select-Dedupe, as shown in Fig. 15. For the mail trace, Select-Dedupe removes 70.7 percent of write requests from the Native system, thus directly reducing the average write response time. On the other hand, iDedup reduces write response times of the Native system by 11.6, 1.7, and 54.5 percent for the web-vm, homes, and mail traces, respectively, which are far less significant than Select-Dedupe. This is because iDedup focuses on large I/O requests and thus eliminates much fewer write requests than Select-Dedupe, as indicated in Fig. 15, resulting in much smaller performance improvement. Although Full-Dedupe reduces much more write requests than Select-Dedupe, as shown in Fig. 15, its write-performance improvement is less than that of Select-Dedupe. In particular, for the homes trace, Full-Dedupe actually increases the

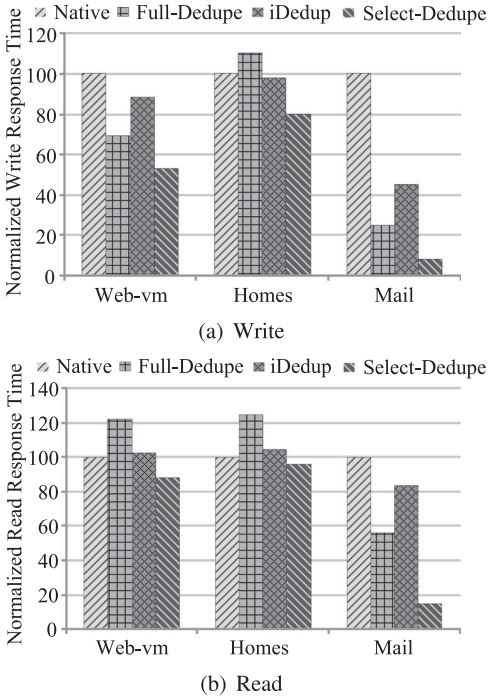

Fig. 12. The average response times of write requests and read requests of the different deduplication schemes normalized to those of the Native system.

write response time of the Native system by 10.1 percent. The reason is that there are many partially redundant write requests in the homes trace. Deduplicating the redundant data of these write requests cannot reduce the write response time because these write requests still must be performed on disks. More importantly, deduplicating these write data will induce the read amplification problem that directly degrades the read performance (see Fig. 12b and its explanation next), thus indirectly affecting the write performance. Furthermore, the on-disk index-lookup operations also degrade the write performance.

Fig. 12b indicates that Full-Dedupe underperforms the Native system in read performance by 22.1 and 24.7 percent for the web-vm and homes traces respectively, but outperforms the Native system by 44.2 percent for the mail trace. The read performance degradation is caused by the read amplification problem for the web-vm and homes traces. Since the mail trace has a significant percentage of fully redundant write requests, the positive effect of reducing a large number of write requests far overshadows the less serious read amplification problem, thus improving the read performance. iDedup achieves comparable read performance to the Native system by only deduplicating redundant large I/O requests to reduce the HDD seek overhead. In contrast, Select-Dedupe consistently outperforms the Native system in read performance by 11.7, 4.3 and 85.3 percent for the web-vm, homes and mail traces, respectively. Select-Dedupe improves the read performance indirectly by reducing the write traffic and avoiding the read amplification problem as much as possible. The significant number of reduced write requests in Select-Dedupe greatly shortens the length of the disk I/O queue and relieves its pressure, thus allowing the read requests to be serviced more quickly.

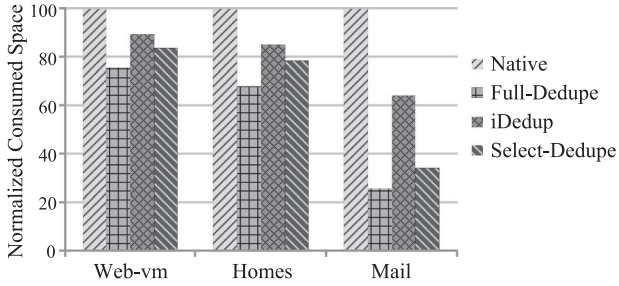

Fig. 13. The normalized storage capacity used by the different deduplication schemes.

We also plot the normalized storage capacity used by the different schemes, as shown in Fig. 13. Full-Dedupe saves the largest amount of storage capacity among all the deduplication-based schemes because it deduplicates all redundant write data, which is not the case for iDedup and Select-Dedupe. Select-Dedupe achieves comparable or better capacity savings than the capacity-oriented deduplication scheme iDedup, especially for the mail trace. This is because, while iDedup only deduplicates large I/O requests, Select-Dedupe deduplicates both large and small I/O requests. When the small I/O requests become a major part of the total requests, the capacity saving is also increased accordantly.

4.3 POD: Select-Dedupe with iCache

Fig. 14 traces the average response times (on the left *Y* axis) of the three deduplication-based systems as the three traces are being replayed, along with the I/O intensity/burstiness (on the right *Y* axis). In this experiment, iDedup and Select-Dedupe use the fixed cache partition between the index cache and read cache while POD uses the dynamic cache partition (iCache). POD outperforms Select-Dedupe by 17.2, 6.4 and 18.7 percent for the web-vm, homes and mail traces, respectively. With iCache, POD can dynamically adjust the size of the index cache and the read cache adapting to the access patterns. Therefore, during the read or write bursty periods, POD performs better than Select-Dedupe. Second, during the write bursty periods, both Select-Dedupe and POD outperform iDedup in terms of user response time. This is because both Select-Dedupe and POD deduplicate more redundant small-write data than iDedup, which effectively shortens the I/O queues and allows the remaining write requests to be serviced on the disks more quickly. The shortened I/O queues also indirectly improve read performance, especially for the mail trace where the read burstiness and write burstiness are mixed. Moreover, as small-write requests are very expensive for parity-based disk arrays due to the *read-modify-write* requirement, reducing small-write requests has the effect of freeing up more disk resources to service the read requests.

Fig. 15 shows the percentage of write requests removed from the Native system by Full-Dedupe, iDedup, Select-Dedupe, and POD under the three traces in a 4-disk RAID5 system. Full-Dedupe removes more write requests than the other three deduplication schemes. The reason is that Full-Deudpe eliminates all redundant write data blocks and uses the full hash index table. In particular, iDedup reduces the fewest write requests because it only focuses on large-write requests and ignores all small-write requests. However, large-write requests account for only a small proportion of

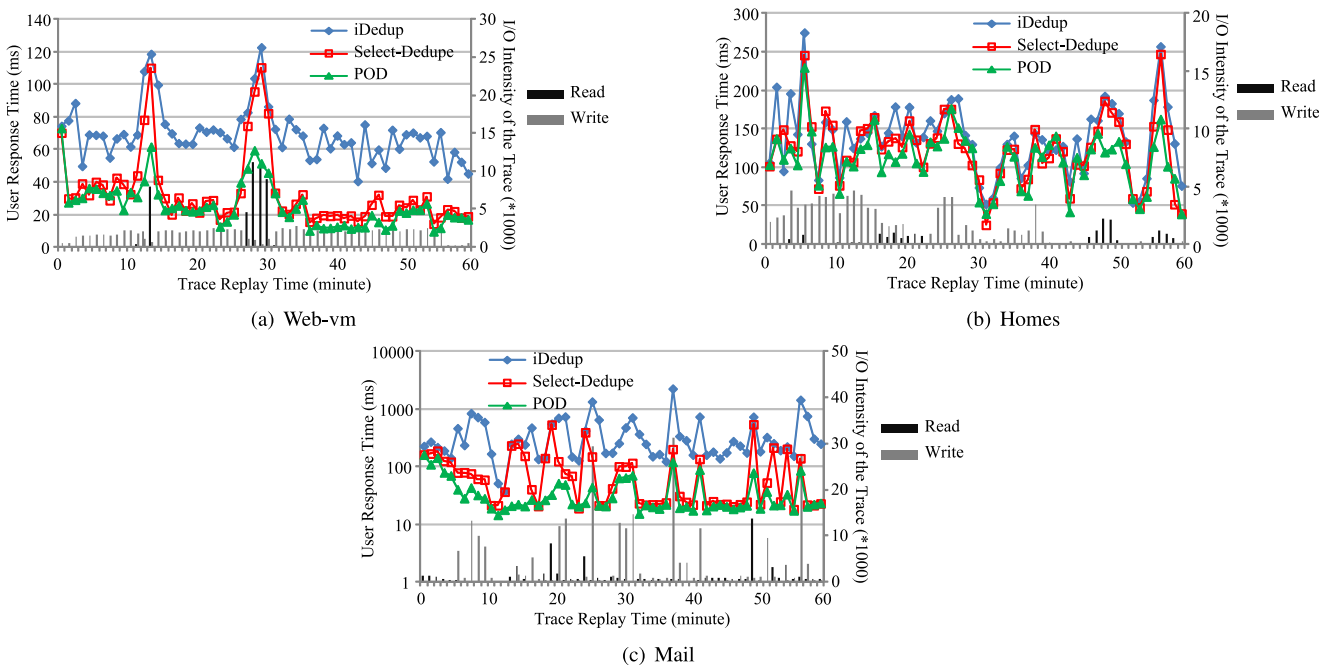

Fig. 14. The average response times (the left *Y* axis) and the I/O burstiness (the right *Y* axis) as a function of the trace-replaying time. Note: For the mail trace, the left *Y*-axis is in log scale.

the total write requests. In contrast, POD and Select-Dedupe reduce much more write requests than iDedup. This is because POD checks the small-write requests that account for a majority of all write requests, as shown in Fig. 1 and described in Section 2.1. Moreover, these small-write requests also exhibit high data redundancy. POD removes slightly more write requests than Select-Dedupe because POD enlarges the index cache size and shrinks the read cache size during write intensive periods, thus detecting more redundant write requests to deduplicate.

It is arguable that with a larger data set the iCache will be much more effective. This is because for a larger data set, the memory space required to store the index cache and read cache will be larger, thus making cache allocation all the more important for and sensitive to performance gains. Our work is underway to collect larger data sets for more indepth POD evaluations.

4.4 Online RAID Reconstruction

Online RAID reconstruction is an important and integral part of parity-based RAID systems of HDDs that POD is primarily design for. With the increasing scale of such systems and stagnant failure rates of HDDs [24], [25], [26], reconstruction is becoming the norm other than the exception [27]. This makes the reconstruction performance increasingly more important. To understand how data deduplication affects the online RAID reconstruction performance, we conducted experiments to measure the online RAID reconstruction performance in terms of reconstruction times and average user response times. Different from DADA [28] that reduces the RAID reconstruction time by only recovering the unique data blocks, POD improves reconstruction performance by leveraging data deduplication to reduce the user I/O operations during online RAID reconstruction [29], [30].

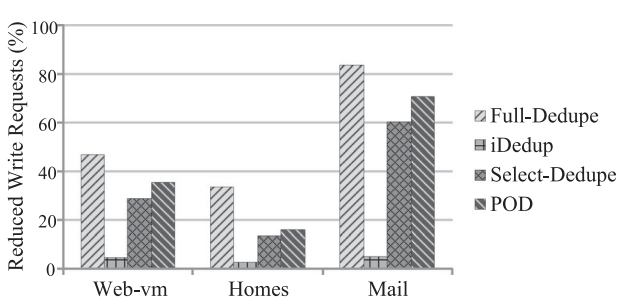

Fig. 15. The percentage of removed write requests by the different deduplication schemes under the three traces in a 4-disk RAID5 system.

Fig. 16 shows the reconstruction performance of deduplication schemes normalized to the Native system under the minimum reconstruction bandwidth of 10 MB/s, driven by the three traces on a RAID5 system consisting of four 10 GB disks. The deduplication-based schemes, i.e., Full-Dedupe, iDedup, and POD, reduce the reconstruction time from the Native system by up to 37.1, 8.4, and 53.5 percent, with an average of 20.1, 3.8, and 29.0 percent, respectively. The reason is that all the three deduplication-based schemes remove many of the user I/O requests addressed to the degraded RAID set, thus freeing more disk resources to service the reconstruction I/O requests to shorten the reconstruction time. The number of I/O requests removed from the degraded RAID set directly affects the reconstruction performance [29]. Although Full-Dedupe removes the largest number of write requests, as shown in Fig. 15, it also introduces more read requests due to the read amplification problem, which will offset the benefit of the write-traffic reduction. iDedup performs the worst among the deduplication-based schemes because it only removes the large I/O requests while ignoring small I/O requests of which many must still be performed on the degraded RAID set.

By introducing data deduplication on the I/O path, redundant write requests can be eliminated, as shown in

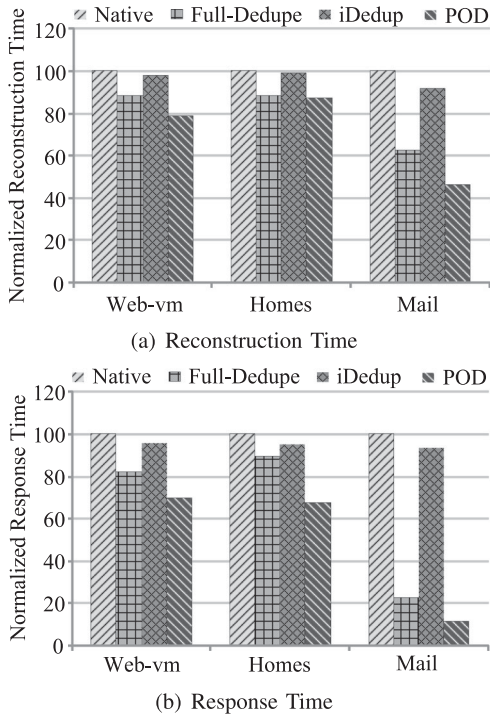

Fig. 16. The online RAID reconstruction performance of the different deduplication schemes normalized to that of the Native system.

Fig. 15. This in turn reduces the user I/O intensity, which can significantly alleviate the I/O workload overhead on the degraded RAID set during reconstruction. Compared with the Native system, Full-Dedupe, iDedup, and POD reduce the user response times by up to 77.0, 6.4, and 88.2 percent, with an average of 35.0, 5.1, and 50.2 percent, respectively. The significant improvements on user response times by Full-Dedupe and POD come from the fact that the length of the request queue is significantly reduced due to the large number of removed write requests. iDedup, on the other hand, improves the user response times only marginally because it only removes a small number of write requests (i.e., large-write requests). Moreover, since the reconstruction period is relatively short, the read performance of Full-Dedupe is not significantly degraded by the read amplification phenomenon, which explains why Full-Dedupe also improves the user response times.

4.5 Overhead Analysis

While data deduplication in POD promises to reduce the write-traffic, it incurs resource overhead. Specifically, there are two main sources of overhead that must be assessed when incorporating POD into HDD-based primary storage systems, that is, computational overhead and memory overhead.

4.5.1 Computational Overhead

Computing the fingerprints of data chunks is time-consuming and directly affects the write performance. The hashing speed depends on the capability of the processors. Using a more powerful processor can effectively reduce the latency of fingerprint calculation. However, this latency, typically in tens of microseconds at most, is insignificant compared to the disk I/O response times that are usually multiple milliseconds. Our extensive experimental results and previous studies [3], [22] have verified that the computing overhead is negligible when compared to the write latency. Moreover, today's multicore processors and Graphic Processing Units (GPUs) make the intelligent storage controllers more powerful, allowing them to extend their capabilities to integrate new techniques, such as data deduplication.

4.5.2 Memory Overhead

Integrating POD into HDD-based primary storage systems needs extra memory space to store the content of the Map_table. To prevent data loss in case of a power failure, POD uses non-volatile memory to store the Map_table. The amount of non-volatile memory required by the Map_table is proportional to the number of reduced write requests. Each entry in Map_table consumes 16 bytes. In the experimental setup of our evaluations, the maximum amounts of non-volatile memory overhead are about 0.8, 0.3 and 1.5 MB for the web-vm, homes and mail traces, respectively. Moreover, with the rapid increase in the memory capacity and decrease in the cost of non-volatile memory, this memory overhead is arguably reasonable and acceptable to the users.

5 Related Work

We briefly review the published research most relevant to our POD approach from the viewpoints of data deduplication in primary storage systems and small-write optimization, respectively.

5.1 Data Deduplication

Data deduplication as a space-efficient technique has received a great deal of attention from both industry and academia. It has been demonstrated to be effective in shortening the backup window and saving the network bandwidth and storage space in backup and archiving applications. For example, D2DRR scheduling scheme is proposed to improve the bandwidth efficiency of Avionics Full DupleX (AFDX) networks with the benefits of low complexity and easy implementation [31]. Most existing studies focus on how to improve the deduplication efficiency by solving the index-lookup-disk-bottleneck problem and how to find the redundant data as much as possible [19], [32]. Recent studies have shown that moderate to high data redundancy also exists in primary storage systems [3], [4], [5], [6], [8].

Among the recent studies that leverage the data deduplication technique to improve the I/O performance of HDD-based primary storage systems, the I/O Deduplication [3] scheme focuses on improving the read performance by exploiting and creating multiple duplications on disks to reduce the disk-seek delay, but does not optimize the write requests [3]. That is, it uses the data deduplication technique to detect the redundant content on disks but does not eliminate them on the I/O path. This allows the disk head to service the read requests by prefetching the nearest blocks from all the redundant data blocks on disk to reduce the seek latency. The write requests are still issued to disks even if their data has already been stored on disks.

Sequence-based deduplication [33] and iDedup [5] are two capacity-oriented data deduplication schemes that target at primary storage systems by exploiting spatial locality to only selectively deduplicate the consecutive file data

blocks. They only select the large requests to deduplicate and ignore all small requests (e.g., 4 KB, 8 KB or less) because the latter only occupy a tiny fraction of the storage capacity. However, previous studies and our workload analysis reveal that the large I/O requests only account for a small portion of all requests while the small I/O redundancy on the I/O path is significant (Fig. 1). It implies that it is the performance on the I/O path, rather than capacity efficiency on the storage devices, that stands to potentially gain more from deduplication for primary storage systems. Different from the above schemes targeted at saving storage space, the request-based Select-Dedupe in POD exploits the I/O redundancy to intelligently and selectively deduplicate the write requests, thus improving the small-write performance of HDD-based primary storage systems while avoiding the data fragmentation problem.

Moreover, none of the existing studies has considered the problem of space allocation between the read cache and the index cache. Most of them only use an index cache to keep the hot index in memory [5], leaving the memory contention problem unsolved [6]. The iCache in POD is designed to address the memory contention and the read amplification problem, although the idea of dynamic cache allocation is not original. Patterson et al. [34] proposed a dynamic cache partition between prefetched data and cached data to maximum the cache efficiency. ARC [35] keeps track of frequently used and recently used pages and a recent eviction history for both of them. It uses ghost hits to adapt to the recent change in the resource usage for better performance. In SSD/HDD hybrid storage systems, Yongseok et al. [36] proposed a dynamic scheme to divide the flash memory cache to cache space and over-provisioned space, thus providing better cache performance and GC efficiency inside SSDs. SAR [15], [21] and Nitro [37] combines the flash-based SSD and data deduplication to improve the performance of primary storage systems. The iCache is inspired by these previous studies and designed to collaborate with Select-Dedupe to further eliminate the redundant write requests and address the read amplification problem in primary storage systems. It is orthogonal to and can be incorporated with the existing data deduplication schemes in primary storage systems to further boost the system performance.

5.2 Optimizations for Small Writes

Generally speaking, most existing small write optimizations for HDD-based primary storage systems utilize the logging technique that buffers the write data in temporal locations, such as disks [38], [39], [40], [41], non-volatile RAM [42] and flash [43], and efficiently flushes them to their original locations eventually.

Parity Logging [40] is introduced to overcome the small-write problem of RAID5 by using the logging technique. For a write request, the new data block is written in place, while the XOR result of the old data block and new data block is recorded sequentially in a log disk. Log-Structured Array (LSA) [39] and DCD [38] also log the updated data on the new disk instead of writing it in place to improve the write performance. AFRAID [44] updates the write data immediately, but delays the parity update operations to the next idle period between the bursts of the client activities. It essentially sacrifices some reliability for performance improvement. Eager-writing [41] writes data to the free block that is closest to the current disk head position. However, how to track the free blocks and disk head position is a complicated issue in practice. In contrast, Range Writes [45] improves the write performance by changing both the file system and the disk to make the fine-grained data placement.

Write-back buffer is effective in improving the write performance. STOW [42] is proposed to effectively capture the temporal locality and spatial locality of write requests, and control the destage rate so that the write-back buffer is neither under-utilized nor over-committed. However, the write-back buffer that uses non-volatile RAM is very expensive in cost and small in capacity. Proximal I/O [43] temporarily aggregates random updates in the flash memory and destages them to disk in a single disk revolution.

Most of the aforementioned schemes aim to alleviate the small-write problem of disks or parity-based disk arrays by delaying the write and parity update operations to the system idle period. POD, different from them, improves the small-write performance by eliminating the write requests when their write data is already stored on disks. Thus, with POD, many small-write requests can be processed in memory without disk I/O operations, which significantly improves the system performance.

6 Lessons Learned

The idea of POD is motivated by a simple goal of simultaneously reducing the write traffic and avoiding the read amplification problem through selective data deduplication and intelligent cache management. However, our process of turning this design goal into a practical deployment reveals several interesting insights into the complicated relationship among performance, power consumption and reliability and shows us that deduplication is a double-edged sword for primary storage systems in the Cloud.

First, the interaction between read and write requests in deduplication-based storage systems is complicated. Deduplication reduces the write traffic and directly improves the write performance if there is no index-lookup disk bottleneck. At the same time, the read performance is indirectly improved because the length of the disk I/O queue is reduced. On the other hand, deduplication causes the read amplification problem that directly degrades the read performance and indirectly affects the write performance. POD is designed to retain the desirable advantage of the write-traffic-reducing ability of and effectively avoid the read amplification problem induced by deduplication, thus significantly improving both the read and write performance. However, more comprehensive and in-depth modeling and analysis of the interaction between read and write requests in deduplication-based primary storage systems are needed.

Second, the energy-efficiency impact of deduplication on primary storage systems is also complicated. Deduplication is effective in reducing the storage space and thus the number of the required disks, potentially saving power consumption in data centers. However, from our evaluations, we observe that the read amplification problem caused by deduplication increases the power consumption because more I/O requests and more disks are involved in completing a read request. Moreover, hash computing also

consumes extra power [46]. This problem is largely ignored in the literature since the deduplication technique is initially used in the backup and archiving environments where read requests are relatively rare. However, the read amplification problem becomes particularly performance and power sensitive when the deduplication technique is deployed in the emerging cloud storage systems and the VM platforms in the cloud. In such environments, the read requests are much more common and performance critical, thus deserving much more thoughtful treatment. As a result, another important objective of POD is to improve the energy efficiency, in addition to the performance and capacity improvement. We will build a power measurement module to evaluate the energy efficiency of the proposed POD in our future study of primary data deduplication in the Cloud.

Third, the reliability of deduplication-based storage systems is complex and should not be overlooked. Deduplication has been demonstrated in the backup and archiving environments to shorten the backup window. Our experimental results of the online RAID reconstruction show that data deduplication can reduce the reconstruction time, thus improving the system reliability. On the other hand, keeping only a single instance of data content that is potentially shared by many files poses a threat to the reliability of the deduplication-based storage systems with POD and without POD [47]. Thus, a systemic reliability analysis of deduplication-based storage systems is also needed.

7 Conclusion

In this paper, we propose POD, a performance-oriented deduplication scheme, to improve the performance of primary storage systems in the Cloud by leveraging data deduplication on the I/O path to remove redundant write requests while also saving storage space. It takes a request-based selective deduplication approach (Select-Dedupe) to deduplicating the I/O redundancy on the critical I/O path in such a way that it minimizes the data fragmentation problem. In the meanwhile, an intelligent cache management (iCache) is employed in POD to further improve read performance and increase space saving, by adapting to I/O burstiness. Our extensive trace-driven evaluations show that POD significantly improves the performance and saves capacity of primary storage systems in the Cloud.

Acknowledgments

The authors thank the SyLab in FIU for providing us the I/O traces. The work of Lei Tian was done while he was working at the Department of Computer Science and Engineering, University of Nebraska-Lincoln. This work is supported by the National Natural Science Foundation of China under Grant No. 61100033, No. 61472336 and No. 61402385, the US National Science Foundation (NSF) Grant No. NSF-CNS-1116606 and NSF-CNS-1016609, National Key Technology R&D Program Foundation of China (No.2015BAH34F01 and No. 2015BAH16F02), Fundamental Research Funds for the Central Universities (No. 20720140515). This is an extended version of our manuscript published in the Proceedings of the 28th IEEE International Parallel & Distributed Processing Symposium (IPDPS'14), Phoenix, AZ, May 2014. S. Wu is the corresponding author.

References

[1] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized deduplication in SAN cluster file systems," in *Proc. Conf. USENIX Annu. Tech. Conf.*, Jun. 2009, pp. 101–114.

[2] K. Jinand and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proc. The Israeli Exp. Syst. Conf.*, May 2009, pp. 1–12.

[3] R. Koller and R. Rangaswami, "I/O Deduplication: Utilizing content similarity to improve I/O performance," in *Proc. USENIX File Storage Technol.*, Feb. 2010, pp. 1–14.

[4] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *Proc. 9th USENIX Conf. File Stroage Technol.*, Feb. 2011, pp. 1–14.

[5] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 299–312.

[6] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li, and S. Sengupta, "Primary data deduplication-large scale study and system design," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Jun. 2012, pp. 285–296.

[7] S. Kiswany, M. Ripeanu, S. S. Vazhkudai, and A. Gharaibeh, "STDCHK: A checkpoint storage system for desktop grid computing," in *Proc. 28th Int. Conf. Distrib. Comput. Syst.*, Jun. 2008, pp. 613–624.

[8] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A study on data deduplication in HPC storage systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2012, pp. 1–11.

[9] X. Zhang, Z. Huo, J. Ma, and D. Meng, "Exploiting data deduplication to accelerate live virtual machine migration," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2010, pp. 88–96.

[10] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu, "Six degrees of scientific data: Reading patterns for extreme scale science IO," in *Proc. 20th Int. Symp. High Perform. Distrib. Comput.*, Jun. 2011, pp. 49–60.

[11] V. Vasudevan, M. Kaminsky, and D. G. Andersen, "Using vector interfaces to deliver millions of IOPS from a networked key-value storage server," in *Proc. 3rd ACM Symp. Cloud Comput.*, Oct. 2012, pp. 1–12.

[12] B. Mao, H. Jiang, S. Wu, and L. Tian, "POD: Performance oriented i/o deduplication for primary storage systems in the cloud," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 767–776.

[13] J. Tucci, "Cloud + Big data = massive change, massive opportunity, keynote address at UW CSE 2011-12 annual industrial affiliates meeting," Oct. 2010, https://www.cs.washington.edu/industrial_affiliates/meetings/2011/agenda/

[14] R. Villars, C. Olofson, and M. Eastwood, "Big Data: What it is and why you should care, white paper," IDC, 2011, http://www.admin-magazine.com/HPC/content/download/5604/49345/file/IDC_Big%20Data_whitepaper_final.pdf

[15] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian, "SAR: SSD assisted restore optimization for deduplication-based storage systems in the cloud," in *Proc. IEEE 7th Int. Conf. Netw., Archit. Storage*, Jun. 2012, pp. 328–337.

[16] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," in *Proc. 5th USENIX Conf. File Storage Technol.*, Feb. 2007, pp. 31–45.

[17] FIU traces [Online]. Available: http://iotta.snia.org/traces/390. 2010.

[18] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 183–198.

[19] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Similarity and locality based indexing for high performance data deduplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1162–1176, Apr. 2015.

[20] A. Batsakis, R. Burns, A. Kanevsky, J. Lentini, and T. Talpey, "AWOL: An adaptive write optimizations layer," in *Proc. 6th USENIX Conf. File Storage Technol.*, Feb. 2008, pp. 67–80.

[21] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian,"Read performance optimization for deduplication-based storage systems in the cloud," *ACM Trans. Storage*, vol. 10, no. 2, pp. 1–22, 2014.

[22] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND flash-based SSDs," in *Proc. 9th USENIX Conf. File Stroage Technol.*, Feb. 2011, p. 7.

[23] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. USENIX Conf. File Stroage Technol.*, Feb. 2011, pp. 77–90.
[24] B. Schroeder and G. Gibson, "Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you?" in *Proc. 5th USENIX Conf. File Stroage Technol.*, Feb. 2007, pp. 1–16.
[25] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "Improving availability of raid-structured storage systems by workload outsourcing," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 64–79, Jan. 2011.
[26] S. Wu, H. Jiang, and B. Mao, "Proactive data migration for improved storage availability in large-scale data centers," *IEEE Trans. Comput.*, 2014, DOI: 10.1109/TC.2014.2366734.
[27] G. Gibson, "Storage at exascale: Some thoughts from Panasas CTO. Interview," May 2011, http://www.hpcwire.com/2011/05/25/storage_at_exascale_some_thoughts_from_panasas_cto_garth_gibson/
[28] Y. Zhang and V. Prabhakaran, "DADA: Duplication aware disk array," in *Proc. USENIX Conf. File Stroage Technol.*, Feb. 2011, pp. 1–2.
[29] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "WorkOut: I/O workload outsourcing for boosting the raid reconstruction performance," in *Proc. 7th Conf. File Storage Technol.*, Feb. 2009, pp. 239–252.
[30] S. Wu, H. Jiang, and B. Mao, "IDO: Intelligent data outsourcing with improved RAID reconstruction performance in large-scale data centers," in *Proc. 26th Int. Conf. Large Installation Syst. Admin.*, Dec. 2012, pp. 17–32.
[31] Y. Hua and X. Liu, "Scheduling heterogeneous flows with delay-aware deduplication for avionics applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 9, pp. 1790–1802, Sep. 2012.
[32] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, and Q. Liu, "Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2014, pp. 181–192.
[33] S. Jones, "Online de-duplication in a log-structured file system for primary storage," Univ. of California, Santa Cruz, CA, USA, Tech. Rep. UCSC-SSRC-11-03, May 2011.
[34] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in *Proc. 15th ACM Symp. Operating Syst. Principles*, Dec. 1995, pp. 79–95.
[35] N. Megiddo and D. Modha, "Arc: A self-tuning, low overhead replacement cache," in *Proc. Conf. File Storage Technol.*, Mar. 2003, pp. 115–130.
[36] Y. Oh, J. Choi, D. Lee, and S. H. Noh, "Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems," in *Proc. 10th Conf. File Storage Technol.*, Feb. 2012, pp. 313–326.
[37] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: A capacity-optimized ssd cache for primary storage," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2014, pp. 501–512.
[38] Y. Hu and Q. Yang, "DCD-Disk caching disk: A new approach for boosting I/O performance," in *Proc. 23rd Annu. Int. Symp. Comput. Archit.*, May 1996, pp. 169–178.
[39] J. Menon, "A performance comparison of RAID-5 and log-structured arrays," in *Proc. 4th IEEE Int. Symp. High Perform. Distrib. Comput.*, Aug. 1995, pp. 167–178.
[40] D. Stodolsky, G. Gibson, and M. Holland, "Parity logging overcoming the small write problem in redundant disk arrays," in *Proc. 20th Annu. Int. Symp. Comput. Archit.*, May 1993, pp. 64–75.
[41] C. Zhang, X. Yu, A. Krishnamurthy, and R. Y. Wang, "Configuring and scheduling an eager-writing disk array for a transaction processing workload," in *Proc. 1st USENIX Conf. File Storage Technol.*, Jan. 2002, pp. 289–304.
[42] B. S. Gill, M. Ko, B. Debnath, and W. Belluomini, "STOW: A spatially and temporally optimized write cache algorithm," in *Proc. USENIX ATC*, Jun. 2009, pp. 327–340.
[43] J. Schindler, S. Shete, and K. A. Smith, "Improving throughput for small disk requests with proximal I/O," in *Proc. 9th USENIX Conf. File Stroage Technol.*, Feb. 2011, pp. 133–148.
[44] S. Savage and J. Wilkes, "AFRAID: A frequently redundant array of independent disks," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, Jan. 1996, pp. 27–39.
[45] A. Anand, S. Sen, A. Krioukov, F. Popovici, A. Akella, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and S. Banerjee, "Avoiding file system micromanagement with range writes," in *Proc. 8th USENIX Conf. Operating Syst. Des. Implementation*, Dec. 2008, pp. 161–176.
[46] L. Costa, S. Al-Kiswany, R. Lopes, and M. Ripeanu, "Assessing data deduplication trade-offs from an energy perspective," in *Proc. Workshop Energy Consumption Rel. Storage Syst.*, Jul. 2011, pp. 1–6.
[47] E. Rozier and W. Sanders, "A framework for efficient evaluation of the fault tolerance of deduplicated storage systems," in *Proc. 42nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2012, pp. 1–12.

Bo Mao received the BE degree in computer science and technology in 2005 from Northeast University; and the PhD degree in computer architecture in 2010 from the Huazhong University of Science and Technology. His research interests include storage system, cloud computing, and big data. He is an assistant professor at the Software School of Xiamen University. He has more than 20 publications in international journals and conferences including *IEEE Transactions on Computers*, *ACM Transactions on Storage*, USENIX FAST, IPDPS, Cluster, USENIX LISA, MASCOTS, and ICPADS. He is a member of the IEEE, ACM, and USENIX.

Hong Jiang received the BSc degree in computer engineering in 1982 from the Huazhong University of Science and Technology; the MASc degree in computer engineering in 1987 from the University of Toronto; and the PhD degree in computer science in 1991 from the Texas A&M University. He is Willa Cather professor of computer science and engineering at the University of Nebraska-Lincoln. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, and performance evaluation. He has more than 200 publications in major journals and international conferences in these areas, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *ACM Transactions on Architecture and Code Optimization*, ISCA, MICRO, USENIX ATC, FAST, LISA, ICDCS, IPDPS, Middleware, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc. He is a fellow of the IEEE and a member of ACM.

Suzhen Wu received the BE and PhD degrees in computer science and technology and computer architecture from the Huazhong University of Science and Technology, in 2005 and 2010, respectively. She is an associate professor at the Computer Science Department of Xiamen University since August 2014. Her research interests include computer architecture and storage system. She has more than 20 publications in journal and international conferences including *IEEE Transactions on Computers*, *ACM Transactions on Storage*, USENIX FAST, USENIX LISA, IPDPS, MASCOTS, and ICPADS. She is a member of the IEEE and a member of ACM.

Lei Tian received the PhD degree in computer engineering from the Huazhong University of Science and Technology in 2010. He is currently a senior member of technical staff at Tintri. Prior that, he was a research assistant professor in the Department of Computer Science and Engineering at the University of Nebraska-Lincoln. His research interests mainly lie in storage systems, distributed systems, cloud computing, and big data. He has more than 40 publications in major journals and conferences including *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Storage*, FAST, HotStorage, ICS, SC, HPDC, ICDCS, MSST, MASCOTS, ICPP, IPDPS, CLUSTER, etc. He is a senior member of the IEEE and a member of ACM.