POD: Performance Oriented I/O Deduplication for Primary Storage Systems in the Cloud

Bo Mao[†], Hong Jiang[‡], Suzhen Wu[*✉], Lei Tian[‡]
[†]Software School of Xiamen University, China
[‡]Department of Computer Science and Engineering, University of Nebraska-Lincoln, USA
[*✉]Corresponding author, Computer Science Department of Xiamen University, China
{maobo, suzhen}@xmu.edu.cn, {jiang, tian}@cse.unl.edu

***Abstract*—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 the relatively high temporal access locality associated with small I/O requests to redundant data. On the other hand, we also observe that 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 approach, called POD, rather than a capacity-oriented I/O deduplication approach, represented by iDedup, to improve the I/O performance of primary storage systems in the Cloud without sacrificing capacity savings of the latter. The salient feature of POD is its focus on not only the capacity-sensitive large writes and files, as in iDedup, but also the performance-sensitive while capacity-insensitive small writes and files. 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% with an average of 58.8%. Moreover, our evaluation results also show that POD achieves comparable or better capacity savings than iDedup.**

***Keywords*-I/O Deduplication; Data Redundancy; Primary Storage; I/O Performance; Storage Capacity**

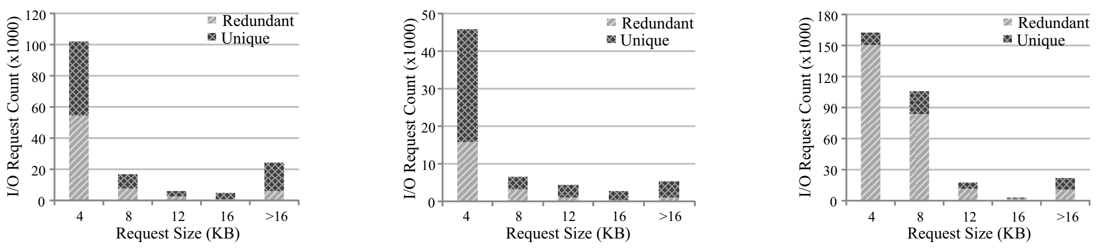
I. Introduction

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 [25], [36]. Recent studies reveal that moderate to high data redundancy clearly exists in VM (Virtual Machine) [5], [10], [22], enterprise [7], [13], [21], [28] and HPC [12], [20] storage systems. These studies have shown that by applying the data deduplication technology to large-scale data sets, an average space saving of 30%, with up to 90% in VM and 70% in HPC storage systems, can be achieved [5], [20], [28]. For example, the time for the live VM migration in the Cloud can be significantly reduced by adopting the data deduplication technology [35].

The existing data deduplication schemes for primary storage, such as iDedup [28] and post-processing deduplication [7], 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.*, 4KB, 8KB or less). The rationale is that the small write requests only account for a tiny fraction of the storage capacity requirement, making deduplication on them unprofitable and potentially counterproductive considering the substantial performance overhead involved in deduplication. Thus, the goal of iDedup is to save the storage capacity with minimal sacrifice on the performance of primary storage systems. However, previous workload studies have revealed that small files dominate in primary storage systems (more than 50%) and are at the root of the system performance bottleneck [3], [16], [20], [21], [27], [30], [34]. Furthermore, due to the buffer effect, primary storage workloads exhibit obvious I/O burstiness [3], [16], [26]. 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 [29], [37] in terms of both performance and capacity. Recent IDC studies indicate that in past five years the volume of data has increased by almost 9 times to 7ZB per year and a more than 44-fold growth to 35ZB is expected in the next ten years [31]. 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 [8], [22], [30].

Moreover, our workload analysis, detailed in Section II-A, 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.*



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

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

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 (read) times with deduplication are much higher than those without deduplication, by an average of 2.9× and up to 4.2× [18]. 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. The key distinction between POD and its capacity-oriented counterparts lies in the former's focus on not only the capacity-sensitive large writes and files, as in the latter, but also the performance-sensitive while capacity-insensitive small writes and files that the latter deliberately ignore. 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. The prototype of the POD scheme is implemented as an embedded module at the block-device level and a subfile deduplication approach is used. 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.

The rest of this paper is organized as follows. Background and Motivation are presented in Section II. We describe the POD architecture and design in Section III. The performance evaluation is presented in Section IV. We review the related work in Section V and conclude this paper in Section VI.

II. Background and Motivation

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

A. Workload characteristics: domination of small files and I/Os in primary storage

Knowing the workload characteristics is important for the design of storage systems, which explains the many recent studies conducted by researchers to analyze primary workloads [1], [3], [8], [16], [20], [21], [26]. These studies reveal that more than 50% of files are smaller than 4KB [1], [21] and 30% to 62% of I/O requests seen at the block level are 4KB [26]. In primary storage data sets, small files are the most common and up to 62% files are smaller than 4KB [20]. 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%, up to 80% for the primary data sets [20]. Our own analysis on primary workloads also finds that small write redundancy (*i.e.*, 4KB or 8KB) dominates, as illustrated in Figure 1 that shows the distribution of the I/O redundancy among requests of different sizes on the 15th day of the three FIU traces. We can see that small writes 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 of primary storage systems in the Cloud. Capacity-oriented deduplication systems, such as iDedup [28], do not deduplicate the small write requests because their deduplication 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

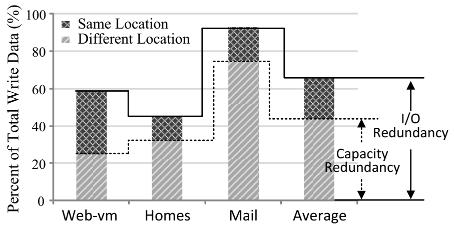


Figure 2. I/O redundancy vs. capacity redundancy for the three applications.

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 [14], [18], [28]. 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.

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 *capacity redundancy* in that the latter is identified from the static data stored on the storage devices when data blocks with different LBAs (logical block addresses) contain the same content [7], [20], [21]. For the redundant write data, only the write data addressed to different locations (LBAs) are considered capacity redundancy and thus may contribute to capacity savings. Figure 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 capacity 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% among the four 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 likely much more effective in reducing I/O traffic than post-processing deduplication [7] for primary storage workloads.

B. Index cache vs. 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 1TB data with an average chunk size of 4KB, the space required by the hash index table is about 8GB. Typical servers cannot keep such a huge hash index table in

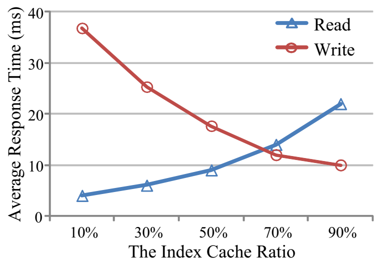


Figure 3. Read and write performances as a function of the percentage of the memory space allocated to the index cache in a deduplication-based storage system, where the memory space is assumed to be divided between the index cache and the read cache.

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 [33], [36].

Figure 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 IV. We can see that with a larger index cache, the write performance is improved because most hash index entries 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, a larger index cache is beneficial to the write performance and a larger 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 primary storage systems, suggesting that read-intensive periods are interleaved with write-intensive periods [2], [26]. 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 is imperative.

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 I 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 the performance-sensitive but capacity-insensitive small writes and files while retaining the capacity savings of capacity-sensitive large writes and files. Moreover, it manages the storage cache intelligently and dynamically according to the workload to improve the overall system performance.

Table I
COMPARISON BETWEEN POD AND THE STATE-OF-THE-ART SCHEMES.

Features	I/O Dedup [13]	iDedup [28]	Post-processing Dedup [7]	POD
Capacity Saving		✓	✓	✓
Performance Enhancement	✓			✓
Small writes Elimination				✓
Large writes Elimination		✓	✓	✓
Cache Partitioning Strategy	Static	Static	Static	Dynamic / Adaptive

III. THE DESIGN OF POD

In this section, we first present an architecture overview of POD, followed by detailed descriptions of two main POD components, Select-Dedupe and iCache.

A. *POD architecture overview*

Figure 4 shows a system architecture overview of our proposed POD in the context of the system I/O in the Cloud. As shown in Figure 4, 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 [21] and iDedup [28]. 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 [10].

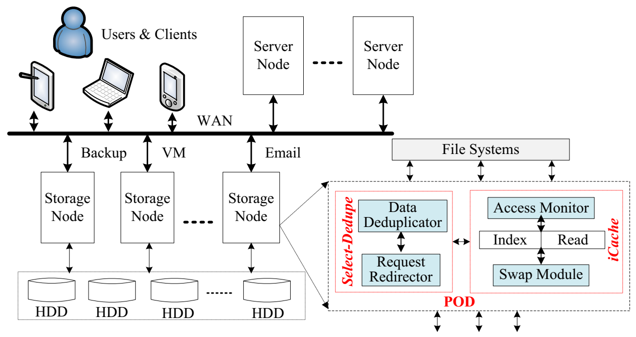


Figure 4. System architecture of POD.

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.

B. *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 Figure 5: (1) the fully redundant write requests whose write data are already sequentially stored on disks, (2) 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 (3) the partially redundant write requests of which the number of redundant data chunks per request exceeds the threshold. Select-Dedupe deduplicates the write requests belonging to category (1) and category (3), and ignores any write requests belonging to category (2). For the write requests in category (2), 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 IV. However, for category (3) 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 IV.

Figure 6 illustrates the process flow of a write request in Select-Dedupe. There are two key data structures sup-

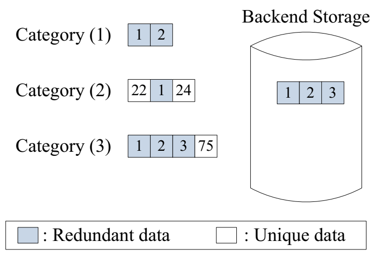


Figure 5. Categorization of write requests in the Select-Dedupe.

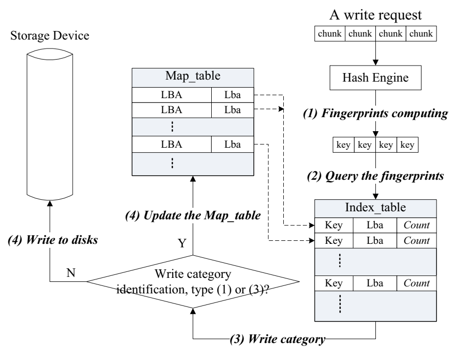


Figure 6. The write process flow in Select-Dedupe.

porting 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 Figure 6. 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 (Local 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 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 Figure 6. 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. 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 Figure 5. If the write request belongs to category (1) or category (3), 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. Otherwise, the data is directly written to the disks without deduplicating the redundant data. The data consistency is also checked to make sure that the referenced data is not overwritten.

C. 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 II-B, 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. Figure 7 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.

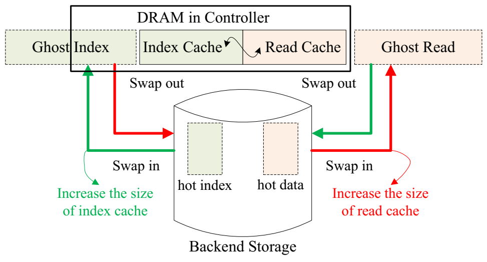


Figure 7. The iCache structure.

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. 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. 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 II-A, 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 two ways. First, on the write path, Select-Dedupe does not deduplicate the write requests with scattered redundant data, as shown in Figure 5. Thus, the operations of the subsequent read requests to these data will be identical to those without data deduplication. Second, 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.

IV. 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.

A. 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 [28]. All the experiments were conducted on a storage node with an Intel Xeon X3440 CPU and 4GB memory. In this system, a SAMSUNG HE253GJ SATA HDD (250GB) 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).

We used three traces, a virtual machine running two web-servers (web-vm), a file server (homes) and an email server (mail), covering a duration of three weeks. The three traces were collected beneath the memory buffer cache so that the caching/buffering effect of the storage stack is already fully captured by the traces. Because the original request data have been split into several small data chunks with a fixed size (*e.g.*, 4KB or 512B), 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 4KB data chunk, which is an overestimation for the processors in modern controllers [9]. 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 II. Due to the different footprints of the three traces, the total memory size is set to be 100MB, 500MB, and 500MB for the web-vm, homes, and mail traces, respectively [13].

Table II
The characteristics of the three traces

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

We replayed the three traces at the block level and evaluated the user response times [32]. 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.

B. Select-Dedupe Performance

We first conducted experiments on a 4-disk RAID5 system with a stripe unit size of 64KB. 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. Figure 8 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% 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 Figure 9.

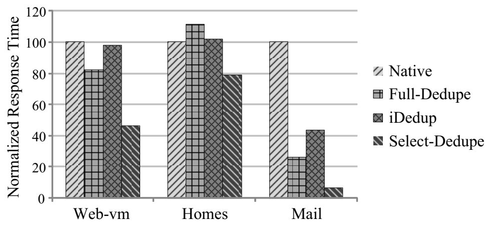


Figure 8. The response-time performance of the different deduplication schemes normalized to the Native system, driven by the three traces for a 4-disk RAID5.

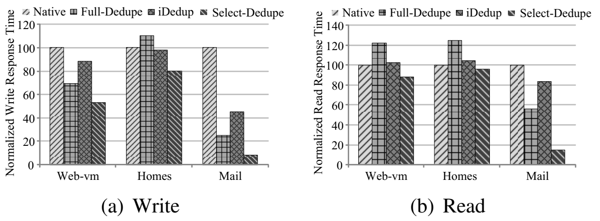


Figure 9. The average response times of write requests and read requests of the different deduplication schemes normalized to those of the Native system.

Figure 9(a) shows that Select-Dedupe reduces the write response times of the Native system by 47.2%, 20.2%, and 91.6%, and those of the iDedup system by 40.2%, 18.8%, and 81.5%, 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 Figure 11. For the mail trace, Select-Dedupe removes 70.7% 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% 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 Figure 11, resulting in much smaller performance improvement. Although Full-Dedupe reduces much more write requests than Select-Dedupe, as shown in Figure 11, its write-performance improvement is less than that of Select-Dedupe. In particular, for the homes trace, Full-Dedupe actually increases the write response time of the Native system by 10.1%. 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 Figure 9(b) and its explanation next), thus indirectly affecting the write performance. Furthermore, the on-disk index-lookup operations also degrade the write performance.

Figure 9(b) indicates that Full-Dedupe underperforms the Native system in read performance by 22.1% and 24.7% for the web-vm and homes traces respectively, but outperforms the Native system by 44.2% 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. iDudup 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% 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.

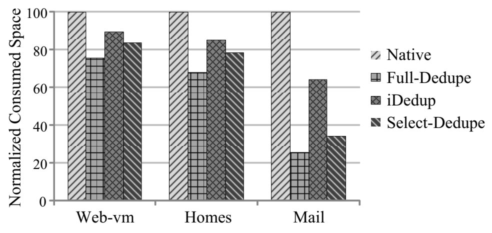


Figure 10. The normalized storage capacity used by the different deduplication schemes.

We also plot the normalized storage capacity used by the different schemes, as shown in Figure 10. 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 write requests. When the small write requests become a major part of the total requests, the capacity saving is also increased accordantly.

C. POD: Select-Dedupe with iCache

Figure 11 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

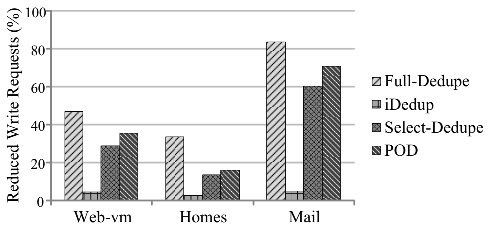


Figure 11. The percentage of removed write requests by the different deduplication schemes under the three traces in a 4-disk RAID5 system.

other three deduplication schemes. The reason is that Full-Dedupe 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 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 Figure 1 and described in Section II-A. 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.

D. 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.

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 [9], [13] 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.

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 20 bytes. In the experimental setup of our evaluations, the maximum amounts of non-volatile memory overhead are 0.8MB, 0.3MB and 1.5MB 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.

V. Related Work

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. 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 [6], [8], [15], [25], [33], [36]. Recent studies have shown that moderate to high data redundancy also exists in primary storage systems [7], [13], [20], [21], [28].

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 [13] 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 [13]. 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 [11] and iDedup [28] 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.*, 4KB, 8KB 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 (Figure 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 [4], [9], [28], leaving the memory contention problem unsolved [7]. 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.* [24] proposed a dynamic cache partition between prefetched data and cached data to maximum the cache efficiency. ARC [19] 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 [17], [23], Yongseok *et al.* 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. 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.

VI. 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.

VII. Acknowledgments

We thank the SyLab in FIU for providing us the I/O traces. We are also grateful to all the members of ADSL group for their continuous support and discussion. Part of the work was performed while Bo Mao was at the University of Nebraska-Lincoln, USA. This work is supported by the US NSF under Grant No. NSF-CNS-1116606, NSF-CNS-1016609, NSF-IIS-0916859, NSF-CCF-0937993, the China National Natural Science Foundation No. 61100033, the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry. This work is also sponsored by Huawei Innovation Research Program.

References

[1] N. Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A Five-Year Study of File-System metadata. In *FAST'07*, Feb. 2007.

[2] A. Batsakis, R. Burns, A. Kanevsky, J. Lentini, and T. Talpey. AWOL: An Adaptive Write Optimizations Layer. In *FAST'08*, Feb. 2008.

[3] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage*, 7(3):1–26, 2011.

[4] 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 *FAST'11*, Feb. 2011.

[5] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized Deduplication in SAN Cluster File Systems. In *USENIX ATC'09*, Jun. 2009.

[6] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in Scalable Data Routing for Deduplication Clusters. In *FAST'11*, Feb. 2011.

[7] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li, and S. Sengupta. Primary Data Deduplication - Large Scale Study and System Design. In *USENIX ATC'12*, Jun. 2012.

[8] D. Frey, A. Kermarrec, and K. Kloudas. Probabilistic Deduplication for Cluster-Based Storage Systems. In *SOCC'12*, Nov. 2012.

[9] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam. Leveraging Value Locality in Optimizing NAND Flash-based SSDs. In *FAST'11*, Feb. 2011.

[10] K. Jinand and E. L. Miller. The Effectiveness of Deduplication on Virtual Machine Disk Images. In *SYSTOR'09*, May 2009.

[11] Stephanie Jones. Online De-duplication in a Log-Structured File System for Primary Storage. Technical Report UCSC-SSRC-11-03, University of California Santa Cruz. May 2011.

[12] S. Kiswany, M. Ripeanu, S. S. Vazhkudai, and A. Gharaibeh. STDCHK: A Checkpoint Storage System for Desktop Grid Computing. In *ICDCS'08*, Jun. 2008.

[13] R. Koller and R. Rangaswami. I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance. In *FAST'10*, Feb. 2010.

[14] M. Lillibridge, K. Eshghi, and D. Bhagwat. Improving Restore Speed for Backup Systems that Use Inline Chunk-Based Deduplication. In *FAST'13*, Feb. 2013.

[15] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. In *FAST'09*, Feb. 2009.

[16] 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 *HPDC'11*, Jun. 2011.

[17] B. Mao, H. Jiang, D. Feng, S. Wu, J. Chen, L. Zeng, and L. Tian. HPDA: A Hybrid Parity-based Disk Array for Enhanced Performance and Reliability. In *IPDPS'10*, Apr. 2010.

[18] 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 *NAS'12*, Jun. 2012.

[19] N. Megiddo and D. Modha. Arc: A self-tuning, low overhead replacement cache. In *FAST'03*, Mar. 2003.

[20] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel. A Study on Data Deduplication in HPC Storage Systems. In *SC'12*, Nov. 2012.

[21] D. T. Meyer and W. J. Bolosky. A Study of Practical Deduplication. In *FAST'11*, Feb. 2011.

[22] C. Ng, M. Ma, T. Wong, Patrick P. C. Lee, and John C. S. Lui. Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud. In *Middleware'11*, Dec. 2011.

[23] Y. Oh, J. Choi, D. Lee, and Sam H. Noh. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In *FAST'12*, Feb. 2012.

[24] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *SOSP'95*, Dec. 1995.

[25] S. Quinlan and D. Sean. Venti: A New Approach to Archival Data Storage. In *FAST'02*, Jan. 2002.

[26] A. Riska and E. Riedel. Disk Drive Level Workload Characterization. In *USENIX'06*, Jun. 2006.

[27] P. C. Roth. Characterizing the I/O Behavior of Scientific Applications on the Cray XT. In *PDSW'07*, Nov. 2007.

[28] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. iDedup: Latency-aware, Inline Data Deduplication for Primary Storage. In *FAST'12*, Feb. 2012.

[29] J. Tucci. Cloud + Big Data = Massive Change, Massive Opportunity, Keynote Address at UW CSE 2011-12 Annual Industrial Affiliates Meeting. Oct 2010.

[30] V. Vasudevan, M. Kaminsky, and David G. Andersen. Using Vector Interfaces to Deliver Millions of IOPS from a Networked Key-value Storage Server. In *SOCC'12*, Oct. 2012.

[31] R. Villars, C. Olofson, and M. Eastwood. Big Data: What It Is and Why You Should Care, White Paper, IDC. 2011.

[32] S. Wu, H. Jiang, and B. Mao. IDO: Intelligent Data Outsourcing with Improved RAID Reconstruction Performance in Large-Scale Data Centers. In *LISA'12*, Dec. 2012.

[33] W. Xia, H. Jiang, D. Feng, and Y. Hua. SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput. In *USENIX ATC'11*, Jun. 2011.

[34] Y. Yue, H. Jiang, L. Tian, F. Wang, D. Fang, and Q. Zhang. RoLo: A Rotated Logging Storage Architecture for Enterprise Data Centers. In *ICDCS'10*, Jun. 2010.

[35] X. Zhang, Z. Huo, J. Ma, and D. Meng. Exploiting Data Deduplication to Accelerate Live Virtual Machine Migration. In *Cluster'10*, Sep. 2010.

[36] B. Zhu, K. Li, and H. Patterson. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In *FAST'08*, Feb. 2008.

[37] P. Zikopoulos, C. Eaton, D. Deroos, T. Deutsch, and G. Lapis. Understanding Big Data. 2010.