2012 IEEE Seventh International Conference on Networking, Architecture, and Storage

P-Dedupe: Exploiting Parallelism in Data Deduplication System

Wen Xia[†‡], Hong Jiang[‡], Dan Feng[†,*], Lei Tian[‡], Min Fu[†], Zhongtao Wang[†]

[†] *School of Computer, Huazhong University of Science and Technology, Wuhan, China*
Wuhan National Lab for Optoelectronics, Wuhan, China
[*] *Corresponding author: dfeng@hust.edu.cn*
[‡] *Dept. of Computer Science and Engineering,University of Nebraska-Lincoln, Lincoln, NE, USA*

***Abstract*— Data deduplication, an efficient space reduction method, has gained increasing attention and popularity in data-intensive storage systems. Most existing state-of-the-art deduplication methods remove redundant data at either the file level or the chunk level, which incurs unavoidable and significant overheads in time (due to chunking and fingerprinting). These overheads can degrade the write performance to an unacceptable level in a data storage system. In this paper, we propose P-Dedupe, a fast and scalable deduplication system. The main idea behind P-Dedupe is to fully compose pipelined and parallel computations of data deduplication by effectively exploiting the idle resources of modern computer systems with multi-core and many-core processor architectures. Our experimental evaluation of the P-Dedupe prototype based on real-world datasets shows that P-Dedupe speeds up the deduplication write throughput by a factor of 2∼4 through pipelining deduplication and parallelizing hash calculation and achieves 80%∼250% of the performance of a conventional storage system without data deduplication.**

I. Introduction

As the amount of digital data grows explosively, data deduplication has gained increasing attention for its space-efficient functionality that not only reduces the storage space requirement by eliminating duplicate data but also minimizes the transmission of redundant data in storage systems [1]. Deduplication uses a hash function (such as a SHA-1 or MD5 hash) to generate a hash signature, also called a fingerprint [2], [1], to uniquely identify regions of data (e.g., files or pieces of files) in the storage system. Due to the significant space-efficiency, data deduplication is widely used in the commercial data backup and archive storage systems [3], [4], [5].

Most existing deduplication solutions aim to remove duplicate data in the data storage systems using the traditional chunk-level deduplication strategies. Despite of recent progresses in deduplication research [3], [6], [7], [8], [9], [10], many challenges remain, particularly in the high performance storage systems. One of the main challenges stems from the time overheads incurred by deduplication that can impose a severe write-performance bottleneck for deduplication-based storage systems [11], [12], [13], [14], [15], [16]. This is because the four stages of the traditional data deduplication process, chunking, hashing for fingerprinting, indexing and storing the fingerprints and data chunks, are time- and space-consuming, where the chunking and hashing processes take up significant CPU resources while fingerprints and their index consume a great deal of RAM space. More specifically, the Rabin-based chunking algorithm [17] and the SHA-1- or MD5-based fingerprinting algorithm all need to compute the hash digest, which may lengthen the write latency to an unacceptable level in deduplication-based storage systems [14], [18], [19], [15].

While hash calculations for deduplication are time consuming and CPU-intensive, modern computer systems based on multicore or manycore processor architectures are poised to provide increasingly more CPU resources [13], [16]. On the other hand, our study of the deduplication process, to be detailed in Section II(B), indicates that the deduplication process can be viewed and organized in terms of data units (such as chunks and files) and functional units (such as chunking, hashing and writing etc.) that are independent of one another. Thus we can effectively utilize the idle CPU resources in multicore-/manycore-based computer systems to fully pipeline and parallelize the compute-intensive deduplication tasks (i.e., functional units) that are then fed by the deduplication data units (e.g., data chunks and files).

To this end, we propose P-Dedupe, a deduplication system that pipelines and parallelizes the compute-intensive deduplication processes to remove the write bottleneck. P-Dedupe aims to remove the time overheads of hashing and shift the deduplication bottleneck from the CPU to the DISK, so as to easily embed data deduplication into a normal file system with little or no impact on the write performance. Our experimental evaluations of P-Dedupe, based on serevral real-world datasets, show that the combination of deduplication pipelining and parallelized hashing can accelerate the write throughput by a factor of 2∼3 and 3∼4 with the Fix-Size Chunking and Content-Defined Chunking algorithms respectively.

The rest of the paper is organized as follows. Section II presents background and motivation for this research. Section III describes the architecture and the algorithm of the P-Dedupe system. Section IV presents our experimental evaluation of P-Dedupe and discusses the results. Section V gives an overview of related work, and Section VI draws conclusions and outlines our future work.

978-0-7695-4722-0/12 $26.00 © 2012 IEEE
DOI 10.1109/NAS.2012.46

II. Background and Motivation

In this section, we first provide the necessary background for deduplication research by introducing the existing challenges facing deduplication based storage systems, and then motivate our research by analyzing our observations based on extensive experiments on deduplication process under real-world workloads.

A. Deduplication based Storage Systems

Although the deduplication technology has been researched and developed for about ten years now and widely used in data backup and archiving products [3], [4], [5], [20], it has only recently started to emerge in the primary storage systems that also have significant amounts of redundant data [21], [22]. In the last two years, the debut of two notable open-source deduplication file system projects, namely, ZFS deduplication [14] and Opendedup [18], show the increasing attention being paid to deduplication in the storage systems. More recently, the data deduplication technology has also been employed for a number of promising primary storage applications, such as SSD [11], [12], VM storage [23], VM migration [24], etc.

However,many of storage systems require a much higher write throughput and employ deduplication primarily as an additional storage-space optimization tool. As such, and more importantly, the deduplication process must not consume too much system resources, especially the execution time and RAM space. One of the main challenges for high throughput data deduplication is the significant time overhead that stems from frequent hash calculations for chunking and fingerprinting in the data deduplication process, which lies in the critical path of write operations [11], [13], [15]. Since hash computation is CPU-intensive and thus time-consuming, data deduplication can result in unacceptable write performance.

Currently, there are two general approaches to accelerating the time-consuming hash calculation and alleviating the computing bottleneck, namely, *hardware-based methods and software-based*. The former refers to employing a dedicated co-processor to minimize the time overheads of computing the hash function so that the deduplication-induced primary-storage performance degradation becomes negligible or tolerable. A good example of the hardware-based methods is Shredder [15] that makes full use of the computing power of the GPGPU device to meet the computational demand of the hash calculation in data-intensive storage systems. In addition to relying on a dedicated hardware device such as GPGPU. Deduplication approaches have recently been employed to minimize the write traffic to the flash memory in SSD-based primary storage [11], [12] where a dedicated co-processor is used in the high-performance SSDs to minimize the hashing latencies.

The software-based approaches exploit the parallelism of data deduplication instead of employing faster computing devices. Liu et al. [25] and Guo et al. [10] have attempted to improve the deduplication performance by pipelining and parallelizing the deduplication process. Nevertheless, their approaches only improve the performance of Fixed Size Chunking (FSC) based deduplication approaches. Because of the internal content-dependency, it remains a challenge to fully exploit the parallelism in the chunking and fingerprinting tasks of the Content Defined Chunking (CDC) based deduplication approaches.

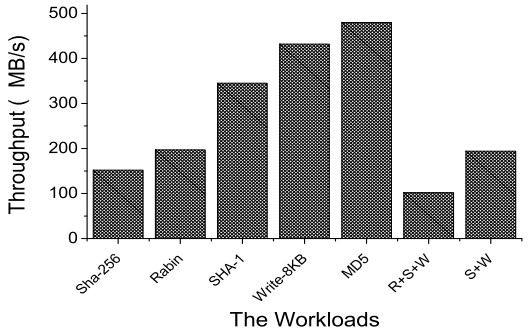


Fig. 1. The average throughputs of chunking (Rabin), fingerprinting (Sha-256, Sha-1, MD5), writing (Write-8KB), CDC-based deduplication (Rabin+Sha-1+Write) and FSC-based deduplication (Sha1+Write)

B. Independence and Dependence

Since the computing power of single-core stagnates while the throughput of storage devices continues to increase steadily [40], the chunking and fingerprinting stages of deduplication are becoming much slower than the writing stage in real-world deduplication systems. Our experimental observation (Linux environment, 7200rpm hard disk , 16 GB ram and 2.8GHz Intel quad-core processor) based on real-world datasets, presented in Figure 1, shows the throughputs of 198MB/s, 345MB/s and 432MB/s for the chunking, fingerprinting and writing stages respectively. Note that the write performance is evaluated with existing caching schemes of file system, which is much higher than existing HDD performance and can be also achieved by other faster storage devices (e.g., SSD or PCM).

Figure 1 also suggests that CDC-based deduplication throughput drops to 102 MB/s due to the serial execution of Rabin followed by Sha-1. To make things worse, with increasing attention being paid to much stronger collision resistant hash digest algorithms (e.g., Sha-256 and Sha-512) for deduplication fingerprints, the throughput for the fingerprinting stage is likely to suffer more significantly, as evidenced by the Sha-256 throughput in Figure 1. Thus, we argue, the time-consuming hash calculation will continue to be the main performance bottleneck of data deduplication in storage systems. Because the hash calculation is **dependent** on the content of data streams, it is challenging to accelerate the chunking and fingerprinting processes in a conventional manner.

Our experimental observations, as well as intuitions, suggest that the deduplication tasks are **independent** of one another. The chunk-level deduplication workflow typically consists of the three stages of chunking, fingerprinting and writing. The writing stage can be further divided into the three sub-stages of looking up the index table, writing the chunk data, and

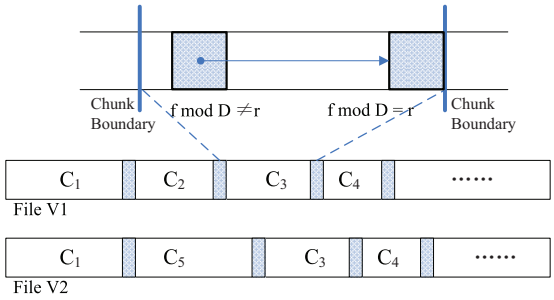


Fig. 2. The sliding window technique used in content defined chunking algorithm. The hash of the sliding window, f, is computed using the Rabin fingerprint algorithm. If the lowest log_2S bits of the hash match a predetermined value r (i.e., f mod S = r), the offset is marked as an anchor.

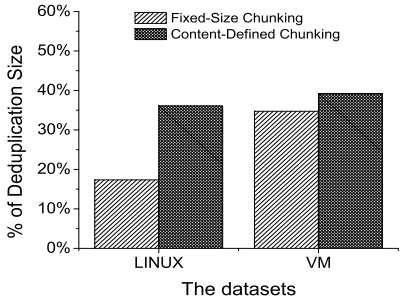


Fig. 3. The duplicate elimination of Content-Defined Chunking approach and Fixed-Size Chunking approach on two different datasets .

writing the metadata. Furthermore, all of these deduplication stages can be considered individual tasks that are serially operated on an independent unit of data chunk. In other words, the fingerprinting of Chunk B can be done concurrently with the writing of Chunk A. Therefore, the stages of chunking, fingerprinting and writing can be pipelined. In addition, the deduplication process also needs to update file metadata that may be transferred over the network, which can be further added to the deduplication pipeline.

C. Challenge of Parallel Chunking

Content-Defined Chunking algorithm is proposed to address the the well known boundary-shift problem [1]: if one data chunk in a file changes (e.g., insertion or deletion), its subsequent data chunks will be all impacted, which dramatically decreases the duplicate elimination by the Fixed-Size Chunking approaches. CDC algorithm uses a sliding window technique to move on the content of files and mark the chunk boundary if the content of this sliding window satisfies their pre-defined requirement as shown in Figure 2. Therefore, to chunk a file V2 that is modified on the chunk C_2 of file V1, the Content-Defined Chunking algorithm can still identify the correct chunking boundary of chunk C_3 whose content has not been modified. To study the duplicate elimination of CDC and FSC approach, we use two real word datasets, 77 virtual machine images (e.g. VM) and 560 versions of Linux (e.g. LINUX), which represent the workloads of large files (237 GB) and small files (74 GB) respectively. Figure 3 shows our observation of duplicate elimination with those two approaches, which well demonstrates that the CDC approach works much better than FSC approach. Note that the completed duplicate files have been removed in our evaluation in Figure 3 since both of CDC and FSC approaches can achieve the same effect on the case of the duplicate files.

Since the Content Defined Chunking (e.g., Rabin chunking) based deduplication approaches can find 10∼20% more duplicate data than the fixed-size chunking approaches especially when file modifications are stored [26], [21], [27], most commercial deduplication products employ the CDC-based approaches to efficiently eliminate duplicate data [3], [4]. But CDC is time-consuming and thus undoubtedly further degrades the write performance in deduplication based storage systems (as shown in Figure 1). Exploiting parallelism of chunking and minimizing the chunking overhead are becoming increasingly urgent for deduplication in both primary and secondary storage systems.

Parallelizing the chunking task, however, is non-trivial for the CDC-based approaches. In fact, it is much more difficult than parallelizing fingerprinting and may not be parallelized in a conventional manner similar to the case of parallel fingerprinting. For example, the Rabin-based chunking algorithm follows a sliding window that is heavily **dependent** on the content of the data stream, as illustrated in Figure 2. More specifically, Rabin's signature generation algorithm for a sliding window (sequence B_1,B_2,...,B_α), is defined as:

$$Rabin(B_1, B_2, ..., B_\alpha) = \{\sum_{x=1}^{\alpha} B_x p^{\alpha-x}\} mod S$$

where S is the average chunk size and α is the number of bytes in the sliding window. The Rabin-based algorithm is very efficient since it is able to compute the signature in an incremental fashion. For the substring in the sliding window, the signature can be incrementally computed from the previous value as follows:

$$Rabin(B_i, B_{i+1}, ..., B_{i+\alpha-1}) =$$
$$\{[Rabin(B_{i-1}, ..., B_{i+\alpha-2}) - B_{i-1}P^{\alpha-1}]p + B_{i+\alpha-1}\} mod S$$

Thus, a chunk is not an independent data unit in the CDC-based chunking stage, because the boundary (i.e., "cut" point) dividing two adjacent chunks remains uncertain until the current chunking for one of the chunks is complete. This makes it very difficult to parallelize the chunking task in the CDC-based approaches.

But, given the independence among the deduplication data units (e.g., chunks and files) and among the deduplication functional units (e.g., the fingerprinting and chunking tasks), we believe that there is a great opportunity to first pipeline the deduplication process while further parallelizing the time-consuming tasks of fingerprinting and chunking, and effectively exploit compute resources in modern computer systems

with multicore/mancore processor architectures, as analysed below.

Given the time overheads due to the chunking, fingerprinting and writing stages, denoted by T_c, T_f and T_w respectively. Let D denote the deduplication factor (i.e., the compression ratio), then T_w/D represents the amount of unique data (i.e., the actual amount of data) that needs to be written. The write throughput of data storage systems without data deduplication can be computed as follows:

$$X_{put} = 1/T_w$$

The traditional serial deduplication's write throughput can be computed as follows:

$$X_{put} = 1/(T_c + T_f + T_w/D)$$

The write throughput of a pipelined deduplication process can be computed as:

$$X_{put} = 1/Max(T_c, T_f, T_w/D)$$

When we further parallelize the hash calculation by "N" parallel threads for deduplication, the dedupliation write throughput can be estimated as:

$$X_{put} = 1/Max(\frac{T_c}{N}, \frac{T_f}{N}, T_w/D)$$

Thus, with sufficient hash-computation parallelism to make "T_c/N" and "T_f/N" less than or equal to "T_w/D", the deduplication write throughput will be approximately D times the normal write throughput without the data deduplication. Thus the combination of pipelined deduplication and parallelized hashing can significantly improve the deduplication write performance and potentially completely remove the write bottleneck in the deduplication based storage systems. Given deduplication's potential to accelerate the write performance by reducing the write traffic, we believe that P-Dedupe will be capable of speeding up the write performance in storage system by a factor up to the deduplication factor D if it completely eliminates the compute bottleneck of deduplication by means of pipelining and parallelizing the chunking and fingerprinting stages.

III. Design and Implementation

P-Dedupe is designed to be a high-write-throughput deduplication system for data-intensive storage systems . In this section, we will first describe the architecture of P-Dedupe, followed by detailed discussions of its design and implementation issues.

A. System Architecture Overview

To provide an appropriate context for presenting the P-Dedupe deduplication scheme that aims to remove the write bottlenecks, this section describes the system architecture of P-Dedupe. Note that the methods presented are general and can be easily applied to other deduplication based storage systems.

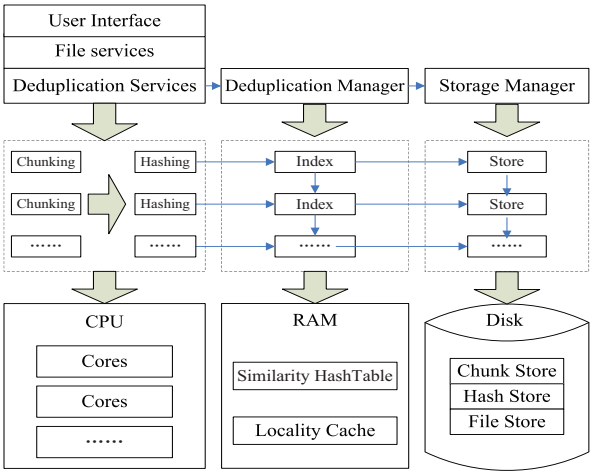


Fig. 4. P-Dedupe system architecture.

Figure 4 shows an overview of the P-Dedupe architecture, which is made up of a stack of software components that include:

- User Interface that provides the common file access interface in the data storage systems.
- File Services that manage the name space and file metadata, where the file metadata need further update when deduplication is finished.
- Deduplication Services that provide the pre-processes of data deduplication, such as parallel chunking, parallel fingerprinting, etc., for the input data stream. Deduplication Services exploit parallelism for the time-consuming chunking and fingerprinting tasks by utilizing the idle CPU resources of the multicore or manycore processors.
- Deduplication Manager that is responsible for managing the fingerprint indexing of the input data stream. It exploits the file semantics of file type, file size, file similarity and locality in datasets by indexing the chunks' fingerprints in P-Dedupe's Similarity Hash Table and Locality based Cache [8], which are stored in the memory.
- Storage Manager that consists of three components, namely, File Store, Hash Store and Chunk Store. Chunk Store and File Store are responsible for storing the chunk data and file metadata respectively, and Hash Store builds the mapping relations between File Store and Chunk Store after deduplication.

When a data stream enters the system, it goes through the User Interface to the File Services layer and then stores the file metadata. Deduplication Services break the data stream into chunks and compute the fingerprints of these chunks. Deduplication Manager detects a chunk's fingerprint in the Hash Table to determine if it is a new chunk. At the end, the chunk information will be updated in File Store, Hash Store and Chunk Store appropriately.

To read a data stream from the system, a client initiates the read operation through the User Interface and the File Services layer. Deduplication Manager obtains the chunk-to-

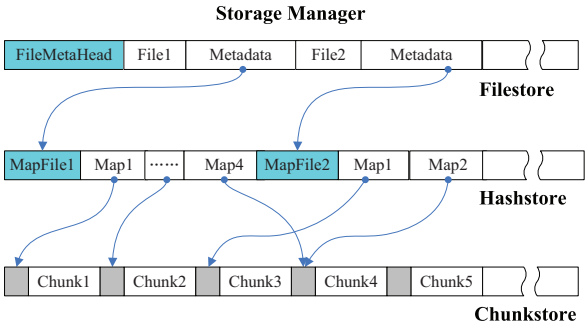


Fig. 5. Index relationship among File Store, Chunk Store and Hash Store in the P-Dedupe system.

file mapping information from Hash Store according to the file metadata in File Store. Then the chunks will be read from Chunk Store according to the mapping information and reconstructed into a file with file metadata. Figure 5 depicts the index structures and the relationship among File Store, Chunk Store and Hash Store. These three components efficiently build an index mapping relationship between files and deduplicated chunks in our P-Dedupe system. For example, when a chunk, say, C_2 of $File_2$ is detected to be a duplicate of chunk C_1 that is already stored in Chunk Store, instead of storing chunk C_2 in Chunk Store, the system only records the C_1 mapping information in $MapFile_2$ in Hash Store. The read operation will reconstruct $File_2$ by reading $MapFile_2$ and then reading the deduplicated chunk C_2.

B. Pipelining Deduplication

The conventional and typical deduplication process follows sequential and serial tasks of chunking, fingerprinting, and writing the chunk data and metadata, chunk-by-chunk and file-by-file. Given the emerging multicore or manycore processor architectures, which offer an abundance of compute resources and parallelism, it is now possible to accelerate the deduplication process by pipelining and parallelizing its tasks.

The pipelining technique in P-Dedupe divides the deduplication process into 4 stages: (S1) chunking, (S2) fingerprinting, (S3) indexing the fingerprints, and (S4) writing chunk data and file metadata (including the chunk-to-file mappings). Because deduplication runs on independent and contiguous chunks, the pipeline can run on the data units of chunks in the deduplication process. The chunk-based deduplication pipeline separates the CPU-intensive tasks, stages S1 and S2, from the I/O-intensive tasks, stages S3 and S4, as shown in Figure 4. This pipelined approach avoids the time waiting on the serial and time-consuming chunking and fingerprinting operations and significantly improves the deduplication throughput in storage systems.

As another type of data units in deduplication, the files can also be processed in the deduplication pipeline, especially when a data stream consists of a large number of small files. The chunk- and file-based pipeline further reduces the unnecessary time waiting on updating file metadata, thus maximizing the efficiency of the deduplication pipeline.

Although our deduplication pipeline is designed for the typical deduplication approaches based on Content Defined Chunking (CDC) (e.g., Rabin-based chunking algorithm [17]), it can also be easily adopted to the deduplication approaches based on Fix-Size Chunking (FSC) or the whole-file granularity, for which the first stage of "chunking"(S1) in the above pipeline would be removed.

C. Parallelizing Fingerprinting

Although the deduplication pipeline separates the CPU-intensive tasks from the I/O-intensive tasks to allow multi-threading, the chunking and fingerprinting threads are still the bottlenecks of the P-Dedupe system as indicated in Figure 4 and Section 2.2. As mentioned earlier, the bottleneck of hash calculation can also be removed by parallelizing on the multicore or manycore processors since the data chunks and files, the subjects of hashing, are independent of one another. The main challenge for parallel fingerprinting lies in the fact that the output order of chunks' computed hash values must be the same as the order in which chunks arrive (i.e., the order of chunks in a file) so as to feed into the next stage of the pipeline (i.e., indexing the hash table and writing data chunks) correctly. For example, the pipeline input of chunks at the parallel fingerprinting stage follows the order of A-B-C, the parallel hash computations of these three chunks by three independent threads may render an out-of-order completion (such as A-C-B or B-A-C) due to the dynamic multicore/manycore runtime conditions and the variable chunk size. Thus, a synchronization of the fingerprinting threads must be introduced in the pipeline to make sure that the chunks' hash values are fed to the Stage 3 (S3) of the pipeline in the correct order in the P-Dedupe system.

D. Parallelizing Chunking

The most challenging issue in parallelizing deduplication task lies in the Rabin-based Chunking (CDC) algorithm, which uses a sliding window on the data stream as depicted in Figure 2. Since the chunking task cannot be parallelized between two adjacent chunks because of content dependency between them, we propose to parallelize the chunking of different substreams of the same data stream, which we call data "sections", of appropriate length (e.g., 128KB) that may contain a large number of chunks. P-Dedupe divides a data stream into multiple data sections, where each section is used to chunk a corresponding portion of the data stream and the size of each section must be larger than the maximum chunk size defined in the CDC algorithm. It then applies the CDC algorithm on the data sections in parallel. When the data sections are chunked individually, the boundaries of the sections need some subtle modifications. More specifically, the ends of the parallel sections are automatically considered as the chunk boundaries by the parallel CDC algorithm, which is not the case in the sequential CDC algorithm.

For example, Sections A and B in Figure 6 are chunked to produce chunks $C_1 \sim C_n$ and $D_1 \sim D_n$ respectively, then the

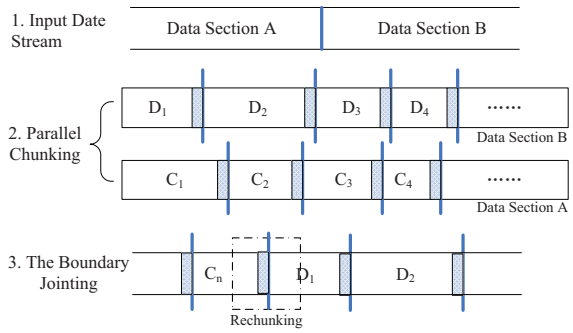


Fig. 6. The parallel chunking algorithm runs with two threads. The data stream is divided to Section A and Section B. The boundaries of Sections A and B are re-chunked at the end of the chunking process.

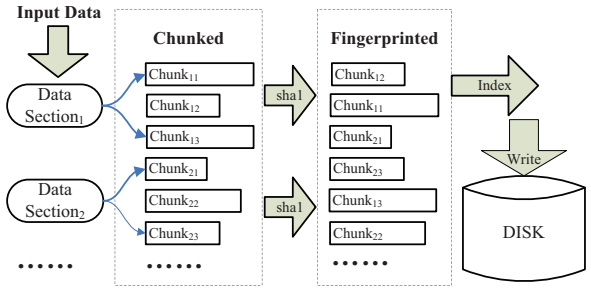


Fig. 7. An example showing the asynchronous executions of the parallel chunking and fingerprinting algorithms respectively with multiple threads in the deduplication pipeline.

last portion of the data in Sections A from the last "cut" point, C_n, may be considered a "tentative chunk" because the hash digest of the sliding window does not meet the condition (see Figure 2). The real boundary of chunk C_n may be located inside D_1, and the system needs to re-chunk the portion of data consisting of the two serial chunks, C_n and D_1.

In fact, only the last α-1 bytes of chunk C_n (sequences of bytes $\{X_1,...,X_{\alpha-1}\}$) and the first α-1 bytes of chunk D_1 (sequences of bytes $\{ Y_1,...,Y_{\alpha-1}\}$) need to be recomputed to generate the Rabin signature if we assume the sliding window size of α. More specifically, P-Dedupe will mark a new "cut" point among the string $\{ C_n , D_1\}$ if any Rabin signatures f_i satisfies the conditions f_i mod S = r in substring $(X_1,...,X_{\alpha-1},Y_1,...,Y_{\alpha-1})$.. Note that this "divide and conquer and connect" approach can be easily applied to more data sections than the two in the example, resulting in any number of parallel threads. As a result, the parallel chunking approach achieves the same effect of the sequential Content-Defined Chunking while linearly reducing the time overheads with multi-threading.

E. Synchronous vs. Asynchronous Schemes in deduplication parallelization

The P-Dedupe scheme combines parallel chunking and parallel fingerprinting to remove the bottleneck of the deduplication pipeline, as depicted in Figure 7. A dilemma ensues, however, with the execution in Stages 1 and 2 of parallel tasks on chunks of variable lengths (due to CDC) and on processing cores with dynamic load conditions. That is, as shown in Figure 7, a purely asynchronous execution of the parallel chunking and fingerprinting tasks can cause serious file fragmentation, where chunks of a file or files enter the pipeline in the order that makes them contiguous in storage but exit the Fingerprinting stage of the pipeline completely out of order. This causes the chunks of a file to be scattered randomly in storage, destroying the sequentiality that is crucial for the write performance of the next stages (S3 and S4) and resulting in unacceptable write and read performances in high performance storage. On the other hand, a purely synchronous execution of these tasks can lead to significant idle waiting for the slowest thread in each statge, again seriously degrading the pipeline performance.

We resolve this dilemma by striking a good balance between synchronization and asynchronization, and making the executions of parallel sections synchronous and the executions of parallel chunks asynchronous. More specifically, this hybrid scheme will only make the chunks out of order within a section of the data stream after S1 and S2 but maintains the sequential order among the data sections of the data stream, which can achieve a performance comparable to the sequential order but without the idle waiting of the purely synchronous execution. This is because the out-of-order impact of chunks within each section can be almost fully absorbed by the existing caching schemes of file system caches. Our evaluation in Section 4 validates the effectiveness of this hybrid scheme.

It must be noted that, while the FSC-based approaches and whole-file deduplication are spared of the chunking bottleneck (i.e., they do not need chunking), they stand to benefit from parallel fingerprinting in the same way as the CDC-based approaches.

IV. Performance Evaluation

In order to evaluate P-Dedupe, we have implemented a prototype of the P-Dedupe system that allows us to examine the performance impact and sensitivity of several important design parameters to provide useful insights into the design of deduplication-based storage system. The evaluation is driven by several real-world datasets that represent different workload characteristics in deduplication systems..

A. Experimental Setup

We use a standard server configuration to evaluate the deduplication performance of the P-Dedupe system running on a Linux environment. The hardware configuration includes an Intel quad-core and eight-thread CPU running at 2.8GHz with a 1MB L2 cache, a 16 GB RAM, and a 1TB 7200rpm hard disk.

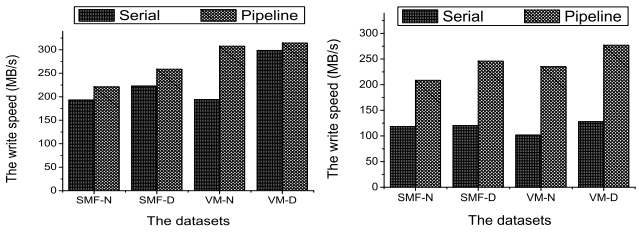
Note that our evaluation of the system throughput is measured by the write speed at which the input data is transferred, deduplicated and stored. As our experimental datasets are directly collected from the local server, we run on the datasets multiple times to obtain stable results of the write throughput. We also compare the write throughputs of systems with

deduplication and without deduplication to evaluate the effectiveness of P-Dedupe's pipelined deduplication and parallel hashing. The RAM usage in our evaluation is obtained by recording the space overhead of index-lookup. The duplicate elimination performance metric is defined as the percentage of duplicate data eliminated by the system.

Since there are still lots of deduplication systems that employ the Fix-Size Chunking (FSC) approaches[10], [18], [14] we implement both a Fix-Size Chunking algorithm and a Content-Defined Chunking (CDC) algorithm in our P-Dedupe system to more comprehensively and fairly evaluate our approach. The datasets of SMF and VM which has been described in section II(C) represent workloads of different object and file sizes in deduplication based storage systems. Both datasets have two versions in our evaluation, the N and D versions, where the former (i.e., SMF-N and VM-N) represent the datasets that are almost completely new as they are written to storage system for the first time, while the latter (i.e., SMF-D and VM-D) represent the datasets that are almost completely redundant as they are duplicated in the storage system. We mainly use 8 Linux versions and 1 VM image to evaluate the sensitivity of our pipeline and parallel schemes to various design parameters, such as hash threads, chunk sizes, FSC or CDC, etc. In fact, the evaluation of a few other large datasets on the P-Dedupe system shows consistent results with the above datasets. The VM dataset is downloaded from the website [28]. The SMF dataset is also downloaded from the website [29] and consists of 560 Linux versions from versions 2.0.1 to 3.2.9, which represents the characteristics of typical source code files.

B. Impact of Pipelining Deduplication

We first compare the pipelined deduplication techniques with serial deduplication across a range of datasets. Figure 8 shows the deduplication write speed of our pipelined approaches under the FSC and CDC chunking algorithms respectively.



(a) Pipelined vs. serial with Fix-Size Chunking (b) Pipelined vs. serial with Content Defined Chunking

Fig. 8. Comparison between serial deduplication and pipelined deduplication with FSC and CDC on four datasets in the P-Dedupe system.

Figure 8 (a) shows that the pipelined deduplication with the FSC algorithm works the best on the VM-N dataset whose average file size is the largest, since the pipeline separates the CPU-intensive tasks from the I/O-intensive tasks. But the pipelined deduplication only obtains very small improvements in the write-throughput performance on other three datasets, since the hash calculations for fingerprints remain the main bottleneck of the pipeline. Figure 8 (b) shows that the pipelined deduplication with the CDC algorithm nearly doubles the write-throughput performance under all four datasets with pipelining the hash calculation of chunking and fingerprinting in the deduplication process. Because the pipelined deduplication alleviates the deduplication bottlenecks from the time-consuming hash calculation of the CDC algorithm, the pipelined CDC-based deduplication achieves a write-throughput performance that is similar to that of the pipelined FSC-based deduplication shown in Figure 8 (a) and (b).

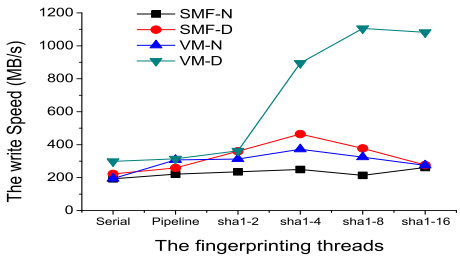


Fig. 9. The performance of the parallel fingerprinting approach, with the FSC algorithm, on the four datasets in the P-Dedupe system.

C. Impact of Parallel Fingerprinting

This subsection presents evaluation results on the impact of parallelizing the compute-intensive tasks of fingerprinting in FSC-based deduplication. Parallelizing chunking and fingerprinting can further improve the performance of the pipelined deduplication by removing the pipeline bottlenecks. The parallel unit of deduplication, namely, the average chunk size is also evaluated for P-Dedupe system.

Figure 9 shows that the write throughput of P-Dedupe achieves a near linear speedup for the four datasets as the number of parallel-fingerprinting threads increases from 1 to 16. But the speedup flattens out beyond 4 threads since the bottleneck of the P-Dedupe system shifts from the fingerprinting task to others (e.g., including, possibly writing to the disk, synchronizing the threads, etc.). From Figure 9, we also find that the deduplication write speedup is around 2∼3.66 on datasets SMF-D, VM-N and VM-D. The deduplication write speedup exceeds 3.5 on the VM-D dataset while fingerprinting with 8 threads. This is because deduplication's ability to reduce the write traffic for datasets with redundant data [11], [12].

Figure 10 shows the performance of parallel fingperprinting approach of FSC based deduplication as a function of different chunk size on four datasets. The larger the chunk size, the faster the write speed our P-Dedupe has, But P-Dedupe get limited improvement of write performance with more than 16KB chunk size on the small file datasets. Note that the duplicate elimination is inversely proportional to the chunk size [21]. Thus the 8KB or 16KB could be a good choice for deduplication based storage system.

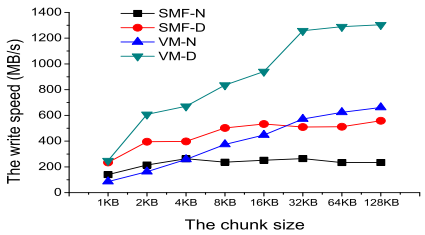
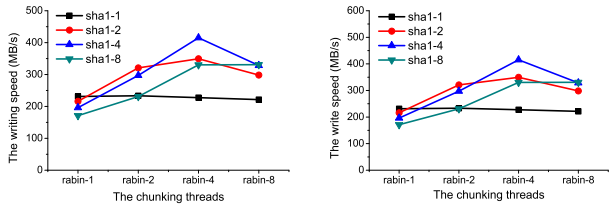


Fig. 10. The performance of the 4 threads parallel fingerprinting as a function of different chunk size on the four datasets in the P-Dedupe system.



(a) Parallel chunking and fingerprinting on SMF-D.

(b) Parallel chunking and fingerprinting on VM-D.

Fig. 11. The deduplication write throughput performance of P-Dedupe as a function of the number of parallel chunking threads and the number of parallel fingerprinting threads, with the CDC algorithm, on two datasets.

D. Impact of Parallel Chunking

Figure 11 shows that the deduplication write throughput increases with the number of parallel chunking and fingerprinting threads in the P-Dedupe system. More importantly, with an equal number of chunking and fingerprinting threads, the P-Dedupe pipeline achieves a good load balance among all the pipeline stages and thus achieves the highest write throughput. On the other hand, when the number of chunking threads exceeds the number of fingerprinting threads, the maintenance and synchronization overheads of the former threads will cause the P-Dedupe pipeline to stall, thus decreasing the write throughput.

Figure 12 shows that the write throughput of CDC-based P-Dedupe achieves a near-linear speedup for the four datasets as the number of parallel-chunking threads increases from 1 to 16 in our 4-core evaluation platform. This performance improvement stems from our parallel chunking algorithm that

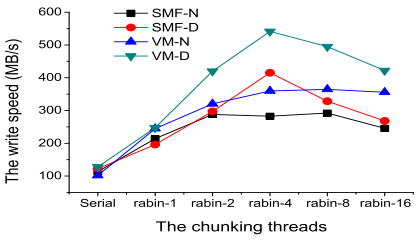
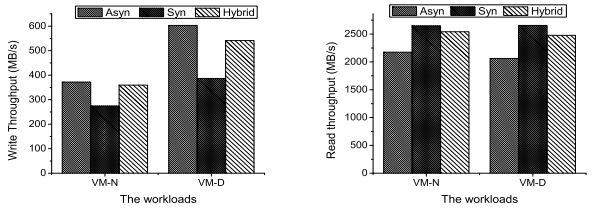


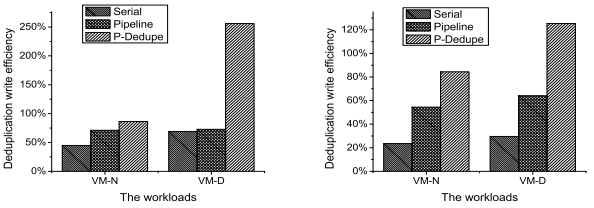
Fig. 12. The parallel chunking performance, with the CDC algorithm and 16 parallel fingerprinting threads, on the four datasets in the P-Dedupe system while chunking is synchronous and fingperprinting is asynchronous.



(a) Write throughput Performance.

(b) Read throughput Performance.

Fig. 13. The parallel chunking and fingerprinting performance of P-Dedupe under the four datasets as a function of asynchronous, synchronous and the hybrid synchronization scheme (i.e., synchronous section chunking and asynchronous chunk fingerprinting)



(a) Fix-Size Chunking.

(b) Content Defined Chunking.

Fig. 14. Comparison of write efficiency between serial, pipeline and P-Dedupe with FSC and CDC approaches.

divides the input data stream into large data sections, chunking these data sections individually and then re-chunking the boundaries of data sections. In addition, the hybrid synchronization scheme for chunking and fingerprinting is able to keep deduplication pipeline fully fed, thus achieving a near-linear speedup of the deduplication throughput. By effectively exploiting thread-level parallelism through pipelining and parallel hashing, the CDC-based P-Dedupe achieves a write-throughput speedup of 2.8 $\sim$ 4.2 over the serial deduplication.

Figure 13 also shows that P-Dedupe with the hybrid synchronization scheme, i.e., synchronous chunking combined with asynchronous fingerprinting, achieves a comparable performance on both write and read throughput to the other two schemes (i.e. asynchronous and synchronous). Although the synchronous chunking and asynchronous fingerprinting will result in out-of-order completion and storing of chunks within a section, the intra-section impact can be well absorbed by existing caching approaches in file systems, achieving a data read performance comparable to that of the in-order case. Figure 13 (a) shows that the hybrid synchronization scheme achieves the write performance near to the scheme of completed synchronization scheme of chunking and fingerprinting. Figure 13 (b) shows that read performance on the data deduplicated by the hybrid synchronization scheme is also near to the read performance on the data deduplicated by the absolute asynchronization scheme, namely, the in order case,

Figure 14 shows that the FSC and CDC based serial deduplications only achieve respectively 40% and 20% of the write efficiency of a system without deduplication, while P-Dedupe achieves 80% $\sim$ 250% of the deduplication write efficiency of

a non-deduplication system by fully and effectively exploiting parallelism in chunking and fingerprinting for deduplication system. Therefore, P-Dedupe is able to minimize the write performance degradation induced by the compute overhead of deduplication in storage system. In addition, we observe that deduplication can potentially improve the write performance if there is enough duplicate data as shown in the workload of VM-D of Figure 14.

V. Related Work

Chunk-based deduplication is the most widely used deduplication method for secondary storage. Such a system breaks a file into contiguous chunks and eliminates duplicate chunks by identifying hash digests (e,g. SHA-1 and MD5) of chunks [2], [1]. Numerous studies have investigated deduplication approaches using fix-size chunking [2], [23], [21], content-defined chunking [26], [1] and bimodal chunking [30]. Generally, the content-defined chunking approaches can improve duplicate elimination when small file modifications are stored. A recent study [21] shows that the CDC approach removes 10% more duplicate data than the FSC approach in their primary storage datasets.

The scalability of deduplication approaches has become an increasingly important issue in mass storage systems, which face potentially severe fingerprint-lookup disk bottleneck with increasing data volumes. DDFS [3] is one of the earliest studies on the idea of exploiting the inherent backup stream locality to reduce accesses to on-disk index and avoid disk bottleneck of inline deduplication. Sparse Indexing [4] is an approximate deduplication solution that uses a sampling technique to reduce the size of fingerprint index in the memory and only requires about half of the RAM usage of DDFS. But its duplicate elimination and throughput is heavily dependent on the sampling data and chunk locality of the backup streams. ChunkStash [7] stores the chunk fingerprints on an SSD instead of an HDD to accelerate the index-lookup. It also exploits backup-stream locality and uses Cuckoo Hash to organize the fingerprint index in RAM, which is shown to be more efficient than the Bloom Filters used in DDFS [3].

Recently, data deduplication in primary storage has been gaining increasing attention due to its ability to save storage space and reduce write. Both the CA-SSD [12] and CAFTL [11] schemes are novel deduplication-based SSD primary storage systems in that they increase the effective SSD space and improve write performance by removing unnecessary duplicate writes and eliminating duplicate data. VMFlock [24] is a deduplication-based virtual machine (VM) co-migration solution, which not only achieves a high compression ratio but also enables faster VM migration by deduplicating VM images. In addition, data deduplication has also emerged in open-source software projects, namely, ZFS dedupe [14] and Opendedup [18]. We also believe that the parallelism schemes of P-Dedupe can be used to boost the deduplication write throughput and scalability of those two approaches since both of them face the hashing challenges.

Increasing attentions are being paid to the hash calculation bottleneck in deduplication systems. Guo et al.[10] propose an event-driven, multi-threaded client-server interaction model to improve deduplication throughput. It also introduces schemes of progressive sampled indexing and grouped mark-and-sweep to improve the single-node deduplication scalability. The GPU technology has been also shown to offer stronger compute power than CPU for many data-intensive (e.g., data-parallel) applications, especially for the application of hash and cryptographic calculation in storage systems. StoreGPU [13], [19] and Shredder [15] make full use of GPU's computational power to accelerate popular compute-intensive primitives in distributed storage systems.

P-Dedupe is in part inspired by THCAS [25], an optimization method to improve performance in deduplication storage system. THCAS adopts a storage pipeline and parallelizes calculation of fingerprints. THCAS divides a chunk into 4 sub-chunks and parallelizes hash calculations of sub-chunks, which may increase the hash collisions of hash algorithm. P-Dedupe is significantly different from THCAS in a number of notable ways. First, THCAS pipelines the deduplication process between files while P-Dedupe pipelines both inter-files and intra in the data stream. Second, THCAS parallelizes hash calculation of sub-chunks while P-Dedupe parallelizes the fingerprinting of chunks. Third, THCAS is unable to parallelize CDC-based chunking while P-Dedupe parallelizes CDC-based chunking and combines parallel fingerprinting and chunking to accelerate write performance nearly linearly.

VI. Conclusions and Future Work

In this paper, we propose P-Dedupe, an efficient and scalable deduplication system that exploits parallelism in the deduplication process in storage systems. P-Dedupe divides the deduplication tasks into four stages and pipelines the four stages with the data units of chunks and files. P-Dedupe also proposes parallel chunking and fingerprinting algorithms to further remove the hash calculation bottleneck of deduplication in high performance storage systems. By effectively exploiting parallelism of deduplication tasks and hash calculation, P-Dedupe is able to make the full use of the idle compute resources in multicore- or manycore-based computer systems to effectively remove the deduplication write bottleneck in deduplication based storage systems.

P-Dedupe is shown to effectively improve the write throughput performance and scalability of deduplication-based storage systems. Its general philosophy of pipelining deduplication and parallelizing hashing is well poised to fully embrace the industry trend of building multicore and manycore processors. With the quad-core and eight-core CPU becoming the mainstream in modern computer systems, the exploitation of CPU resource can easily remove the computing bottlenecks of deduplication based storage systems. With the removal of the deduplication bottlenecks in storage system, it may not be long before data deduplication becomes mainstream in high performance storage systems.

As the number of cores of chip multiprocessors increases with the new "Moore's law", the storage systems may require higher deduplication throughput because of the incorporation of faster storage devices (e.g., SSD or PCM). Thus P-Dedupe must consider exploiting more thread-level parallelism on processors with more cores and address more potential challenges due to much higher parallelism, which are considered as our future work.

VII. Acknowledgments

This work was supported by Chinese 973 Program of under Grant No.2011CB302301, Changjiang innovative group of Education of China No. IRT0725, National Natural Science Foundation of China (NSFC) under Grant 61025008 and 61173043, and the US NSF under Grants NSF-CNS-1116606, NSF-CNS-1016609, NSF-IIS-0916859 and NSF-CCF-0937993. The authors are also grateful to anonymous reviewers for their feedback and guidance.

References

[1] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*. ACM, 2001, pp. 174–187.

[2] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," in *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, vol. 4, 2002.

[3] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*. USENIX Association, 2008, pp. 1–14.

[4] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: large scale, inline deduplication using sampling and locality," in *Proccedings of the 7th conference on File and storage technologies*. USENIX Association, 2009, pp. 111–123.

[5] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. Klein, "The design of a similarity based deduplication system," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, pp. 1–14.

[6] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*. IEEE, 2009, pp. 1–9.

[7] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: speeding up inline storage deduplication using flash memory," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*. USENIX Association, 2010, p. 16.

[8] 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 *Proceedings of the 2011 conference on USENIX Annual technical conference*. USENIX Association, 2011.

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

[10] F. Guo and P. Efstathopoulos, "Building a High-performance Deduplication System," in *Proceedings of the 2011 conference on USENIX Annual technical conference*. USENIX Association, 2011.

[11] 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 *FAST11: Proceedings of the 9th Conference on File and Storage Technologies*. USENIX Association, 2011.

[12] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging Value Locality in Optimizing NAND Flash-based SSDs," in *FAST11: Proceedings of the 9th Conference on File and Storage Technologies*. USENIX Association, 2011.

[13] A. Gharaibeh, S. Al-Kiswany, S. Gopalakrishnan, and M. Ripeanu, "A gpu accelerated storage system," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 167–178.

[14] ZFS-Deduplication, "http://bit.ly/bsT60L."

[15] P. Bhatotia, R. Rodrigues, and A. Verma, "Shredder: Gpu-accelerated incremental storage and computation," in *USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[16] W. Xia, H. Jiang, D. Feng, and L. Tian, "Accelerating data deduplication by exploiting pipelining and parallelism with multicore or manycore processors," in *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST poster session)*, 2012.

[17] M. Rabin, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[18] opendedup, "http://www.opendedup.org/."

[19] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, and M. Ripeanu, "Storegpu: exploiting graphics processing units to accelerate distributed storage systems," in *Proceedings of the 17th international symposium on High performance distributed computing*. ACM, 2008, pp. 165–174.

[20] L. You, K. Pollack, and D. Long, "Deep store: An archival storage system architecture," in *Proceedings of the 21st International Conference on Data Engineering*. IEEE Computer Society, 2005.

[21] D. Meyer and W. Bolosky, "A study of practical deduplication," in *FAST'11: Proceedings of the 9th Conference on File and Storage Technologies*, 2011.

[22] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "idedup: Latency-aware, inline data deduplication for primary storage."

[23] K. Jin and E. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, pp. 1–12.

[24] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: Virtual machine co-migration for the cloud," in *Proceedings of the 20th ACM International Symposium on High Performance Distributed Computing*. ACM, 2011.

[25] C. Liu, Y. Xue, D. Ju, and D. Wang, "A novel optimization method to improve de-duplication storage system performance," in *2009 15th International Conference on Parallel and Distributed Systems*. IEEE, 2009, pp. 228–235.

[26] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data," in *Proceedings of the annual conference on USENIX Annual Technical Conference*. USENIX Association, 2004, p. 6.

[27] J. Min, D. Yoon, and Y. Won, "Efficient deduplication techniques for modern backup operation," *Computers, IEEE Transactions on*, no. 99, pp. 1–1, 2011.

[28] VM-dataset, "http://bit.ly/TKjna."

[29] Linux-dataset, "http://bit.ly/9HmdXP."

[30] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *Proceedings of the 8th USENIX conference on File and storage technologies*. USENIX Association, 2010, p. 18.