A Comprehensive Study of the Past, Present, and Future of Data Deduplication

This paper provides a comprehensive survey of the state of the art in data deduplication technologies for storage systems, covering key technologies, main applications, open problems, and future research directions.

By Wen Xia, *Member IEEE*, Hong Jiang, *Fellow IEEE*, Dan Feng, *Member IEEE*, Fred Douglis, *Senior Member IEEE*, Philip Shilane, Yu Hua, *Senior Member IEEE*, Min Fu, Yucheng Zhang, and Yukun Zhou

ABSTRACT | Data deduplication, an efficient approach to data reduction, has gained increasing attention and popularity in large-scale storage systems due to the explosive growth of digital data. It eliminates redundant data at the file or subfile level and identifies duplicate content by its cryptographically secure hash signature (i.e., collision-resistant fingerprint), which is shown to be much more computationally efficient than the traditional compression approaches in large-scale storage systems. In this paper, we first review the background and key features of data deduplication, then summarize and classify the state-of-the-art research in data deduplication according to the key workflow of the data deduplication process. The summary and taxonomy of the state of the art on deduplication help identify and understand the most important design considerations for data deduplication systems. In addition, we discuss the main applications and industry trend of data deduplication, and provide a list of the publicly available sources for deduplication research and studies. Finally, we outline the open problems and future research directions facing deduplication-based storage systems.

KEYWORDS | Data compression; data deduplication; data reduction; delta compression; storage security; storage systems

I. INTRODUCTION

The amount of digital data in the world is growing explosively, as evidenced in part by the significant increase in the estimated amount of data generated in 2010 and 2011 from 1.2 zettabytes to 1.8 zettabytes, respectively [1], [2], and the predicted amount of data to be produced in 2020 is 44 zettabytes [3], [4]. As a result of this "data deluge," how to manage storage cost-effectively has become one of the most challenging and important tasks in mass storage systems in the big data era. The workload studies conducted by Microsoft [5], [6] and EMC [7], [8] suggest that about 50% and 85% of the data in their production primary and secondary storage systems, respectively, are redundant. According to a recent IDC study [9], almost 80% of corporations surveyed indicated that they were exploring data deduplication technologies in their storage systems to reduce redundant data and thus increase storage efficiency and reduce storage costs.

Data deduplication is an efficient data reduction approach that not only reduces storage space [5]–[7],

Manuscript received August 16, 2015; revised February 25, 2016 and May 12, 2016; accepted May 12, 2016. Date of publication August 2, 2016; date of current version August 18, 2016. This work was supported in part by the National Science Foundation of China (NSFC) under Grants 61502190 and 61232004; by the 863 Project 2013AA013203; by the State Key Laboratory of Computer Architecture under Grant CARCH201505; by the Fundamental Research Funds for the Central Universities, HUST, under Grant 2015MS073; by the U.S. National Science Foundation (NSF) under Grants CNS-1116606 and CNS-1016609; by the Key Laboratory of Information Storage System, Ministry of Education, China; and by the EMC Corporation. *(Corresponding author: Dan Feng.)*

W. Xia is with the School of Computer Science and Technology, Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: xia@hust.edu.cn).
H. Jiang is with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: hong.jiang@uta.edu).
D. Feng, **Y. Hua**, **M. Fu**, **Y. Zhang**, and **Y. Zhou** are with the Wuhan National Laboratory for Optoelectronics, the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: dfeng@hust.edu.cn; csyhua@hust.edu.cn; fumin@hust.edu.cn; cszyc@hust.edu.cn; ykzhou@hust.edu.cn).
F. Douglis and **P. Shilane** are with EMC Corporation, Princeton, NJ 08540 USA (e-mail: fred.douglis@emc.com; philip.shilane@emc.com).

Digital Object Identifier: 10.1109/JPROC.2016.2571298

[10]–[13] by eliminating duplicate data but also minimizes the transmission of redundant data in low-bandwidth network environments [8], [14], [15]. In general, a typical *chunk-level* data deduplication system splits the input data stream (e.g., backup files, database snapshots, virtual machine images, etc.) into multiple data "chunks" that are each uniquely identified and duplicate-detected by a *cryptographically secure hash signature* (e.g., SHA-1), also called a *fingerprint* [11], [14]. These chunks can be fixed in size [11], like file blocks, or variable-sized units determined by the content itself [14]. Deduplication systems then remove duplicate data chunks and store or transfer only one copy of them to achieve the goal of saving storage space or network bandwidth.

Traditionally, data reduction has been the result of data compression approaches that use a dictionary model to identify redundancy for short strings (e.g., 16 B), such as the classic LZ77/LZ88 algorithms [16], [17]. Most of these approaches first compute a weak hash of strings and then compare the *hash-matched* strings byte by byte. Because of their time and space complexity, dictionary-model-based compression approaches, such as LZO [18], LZW [19], DEFLATE [20], only compress data in a much smaller region, e.g., data within a file or a group of small files [21], which trades off processing speed against compression effectiveness.

For large-scale storage systems, data deduplication is shown to be more scalable and efficient than traditional compression approaches (e.g., Huffman coding or LZ compression). The main benefits of data deduplication are twofold. First, deduplication identifies and eliminates redundancy at the *chunk-* (e.g., 8KB) or *file-level* while the traditional compression approaches work at the string or byte level. Second, deduplication identifies the duplicate content (files or chunks) by calculating its *cryptographically secure hash-based fingerprints* (simply "*fingerprints*" for short in the remainder of the paper), which avoids the traditional method of byte-by-byte comparisons. These two features enable deduplication to be easily applicable for global data reduction in large-scale storage systems by computing and indexing the fingerprints of chunks or files. This is because the size of fingerprints for deduplication is orders of magnitude smaller than that of the original data.

Generally, the main workflow of most of the chunk-level data deduplication approaches consists of five key stages, namely, chunking, fingerprinting, indexing of fingerprints, further compression, and storage management. Further compression is optional, including traditional compression of the nonduplicate chunks (e.g., LZ compression) and delta compression of the nonduplicate but similar chunks. The storage management in data deduplication systems can be specified into several categories, such as data restore (fragment elimination), garbage collection, reliability, security, etc.

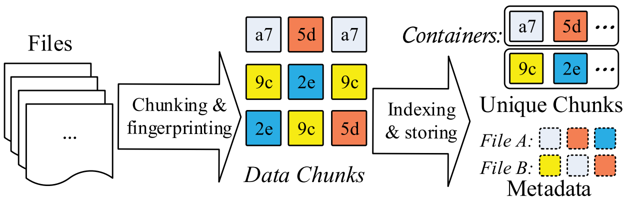


Fig. 1. ***Overview of deduplication processing.***

Fig. 1 shows the general workflow of data deduplication. The files are first divided into equal or similarly sized chunks and each chunk is uniquely represented by its fingerprint. Deduplication only stores the unique (i.e., nonduplicate) chunks on disk by quickly verifying their uniqueness using (indexing) their fingerprints. It also records the list of constituent chunks in *metadata* that will be used to reconstruct the original file. When the size of fingerprints overflows the RAM capacity in large-scale storage systems, several optimizing approaches are proposed for accelerating the on-disk index-lookup process. A typical example is the data domain file system (DDFS) [13] using Bloom filters [22] and a locality-preserved cache to accelerate fingerprints indexing for deduplication, as detailed in Section III-C. Usually, the unique chunks will be stored into several large fixed-size storage units called containers [13], and thus the restore of each file will cause many random I/Os to the containers due to chunk fragmentation [23]: the chunks of a file become scattered all over different containers after deduplication. The detailed storage management of chunks and metadata will be discussed in Section III. In addition there are many publicly available deduplication tutorials [24]–[27], which may also be helpful for understanding the deduplication concepts.

In this paper, we focus our study of the state-of-the-art research on each of the five stages of the data deduplication workflow to provide insight into the evolution of the technology over the years and pros and cons of the state-of-the-art approaches, and outline the open problems and research challenges facing data-deduplication-based storage systems. We recognize and appreciate the contributions of several recent survey studies of data deduplication [28]–[32], which introduce deduplication strategies and use cases, focus on a particular aspect of deduplication (e.g., indexing schemes [31]), or survey the application of the deduplication technology in a particular area (e.g., cloud storage [32]). However, our study is different from these prior surveys in that we study the evolution and key features of the technologies for redundant data reduction. More importantly, we provide an in-depth study of the state-of-the-art approaches to all stages of data deduplication, including the new and emerging areas for deduplication, such as delta compression, restore, garbage collection, security, reliability, etc.

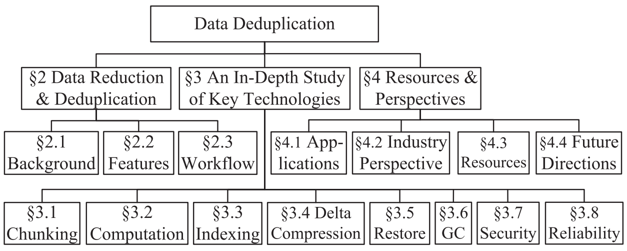


Fig. 2. ***Organization of this survey paper.***

The main contributions of this study are threefold.

- We study the background and methodologies of data deduplication. By examining the major differences between data deduplication and the traditional compression approaches, we present the key features and advantages of data deduplication.
- We study and classify the state of the art of each stage of the data deduplication workflow, including the chunking approaches, computation accelerations for chunking, indexing of fingerprints, delta compression, data restore, garbage collection, security, and reliability. Based on the in-depth stage-based study of existing approaches, we present a detailed taxonomy of the state-of-the-art data deduplication techniques, which provide useful insights into important design issues of deduplication-based storage systems.
- We discuss the main applications and industry trends of data deduplication; provide a collection of publicly available open-source projects, data sets, and traces for the data deduplication research community; and outline open problems and research challenges facing data deduplication research.

It is noteworthy that the data deduplication discussed in this study is different from the term "deduplication" in the database, data mining, and knowledge engineering fields. In these fields, duplicate detection is commonly used to identify records that represent the same entity [33], [34]. This study focuses on eliminating identical data, thus saving storage space in storage systems. The organization of the paper is illustrated in Fig. 2. Section II presents the background and methodologies of data deduplication. The state-of-the-art data deduplication approaches are described and classified into six categories in Section III following the key stages of the data deduplication workflow. Section IV presents the main applications, discusses industry trends and future research directions of data deduplication, and introduces a collection of publicly available resources for the research community to study data deduplication technologies. Section V summarizes the paper. Finally, in the Appendix, we list and explain some frequently discussed acronyms and terminologies.

II. REDUNDANT DATA REDUCTION AND DATA DEDUPLICATION

In this section, we first provide the necessary background for redundant data reduction by introducing a taxonomy that shows both the traditional data compression approaches and the data deduplication approaches, as well as their evolution over the decades. We then study the key features of data deduplication to show how and why it is different from the traditional compression approaches. Finally, we review the key workflow of a typical data deduplication system.

A. A Primer on Redundant Data Reduction

"The goal of data compression is to represent an information source (e.g., a data file) as accurately as possible using the fewest number of bits" [35], [36]. Generally speaking, data compression can be classified into two broad categories, *lossless* and *lossy*. Lossless compression reduces data by identifying and eliminating statistical redundancy in a reversible fashion, as represented by algorithms such as GZIP [37] and LZW [19]. Lossy compression reduces data by identifying unnecessary information and irretrievably removing it, as typified by the JPEG image compression [38]. This paper focuses on the lossless compression category, including data deduplication, since lossless compression is required for general-purpose storage systems.

The theory of data compression was formulated by Claude E. Shannon who introduced the theory of information entropy (simply "entropy" for short henceforth) to define the relationship between the occurrence probability of information (or the uncertainty) and data redundancy in his seminal 1948 paper "A mathematical theory of communication" [39]. This entropy theory established the existence of a fundamental limit to lossless data compression. Specifically, the entropy H of a discrete random variable X with possible values $\{x_1, \ldots, x_n\}$ and probability mass function $P(X)$ was defined by Shannon as

$$\begin{aligned} H(X) = E(I(X)) &= \sum_{i=1}^{n} P(x_i) I(x_i) \\ &= -\sum_{i=1}^{n} P(x_i) \log_b P(x_i). \end{aligned} \quad (1)$$

Here E is the expected value operator, I is the information content of X, $I(X)$ is itself a random variable, and b is the base of the logarithm used (e.g., $b = 2$ refers to binary representation). For example, given a string "abaaacabba," the occurrence counts of characters "a," "*b*," and "*c*" are 6, 3, and 1, respectively. So the entropies of each character and the string are calculated as follows:

$$\begin{aligned} H(a) &= -\log_2(0.6) = 0.737 \text{ bits} \\ H(b) &= -\log_2(0.3) = 1.737 \text{ bits} \\ H(c) &= -\log_2(0.1) = 3.322 \text{ bits} \\ H(\{a,b,c\}) &= H(a) \times 0.6 + H(b) \times 0.3 + H(c) \times 0.1 \\ &= 1.295 \text{ bits.} \end{aligned} \tag{2}$$

ASCII would require 80 bits to represent these ten characters. Equation (2) means that each character can be maximally compressed into 1.295 bits and thus the whole string into 12.95 bits in theory, though more are needed since partial bits are not possible. For example, we can encode "*a*," "*b*," and "*c*" with "0," "10," and "11," respectively; this would require 14 bits ("01000011010100"). Therefore, Shannon's entropy theory discloses that an information message can be expressed by fewer bits, which is the essence of redundant data reduction.

The early data compression approaches use the statistical-model-based coding, also called entropy coding, that identifies redundancy at the byte level. The most widely used entropy coding algorithm is Huffman coding [40], which uses a frequency-sorted binary tree to generate the optimal prefix codes for entropy coding. In order to achieve the fundamental limit of data compression ratio (e.g., the aforementioned entropy value), arithmetic coding [41], first proposed by Elias in the 1960s [42], encodes the entire message into several decimals for a higher compression ratio. Huffman coding separates the input into component symbols and replaces each with a shorter code.

With the growing size of digital data in the world, a new approach called dictionary-model-based coding was proposed by Lempel and Ziv in the 1970s, represented by the LZ77 [16] and LZ78 [17] algorithms. These simplify and speedup data compression by identifying redundancy at the string level: it identifies the repeated strings using a sliding window and replaces these repeated strings by the positions and lengths of the matched ones. Later in the 1980s, variants of the LZ compression approach were proposed to either improve compression ratio (e.g., DEFLATE and LZMA [20], [43], [44]) or speed up the compression process (e.g., LZO [18] and LZW [19]).

In general, entropy-coding approaches need to count all the information before coding the frequently occurring bytes into shorter bits, which is not scalable. Meanwhile, dictionary-coding approaches need to search all the strings to support the matching and elimination of duplicate strings. As a result, both entropy- and dictionary-coding approaches often limit the compression window to trade off between the compression ratio and speed. For example, DEFLATE [43] (which combines LZ77 and Huffman coding) uses the compression window size of 64 kB, while the maximum windows for bzip2 [21], [45] and 7z [44] are 900 kB and 1 GB, respectively. The wide range of compression window sizes can result in large variations in overall compressibility, though there are some techniques to increase compression by preprocessing data [21], [46].

Delta compression was proposed in the 1990s to target the compression of very similar files (i.e., the different versions of a file) or similar chunks. It has been widely used in many applications, such as source code version control [47]–[49], remote synchronization [8], [50]–[52], and backup storage systems [53]–[55]. Taking Xdelta [49] as an example, for similar files (or chunks) A and B, Xdelta uses the COPY/INSERT instructions to record the matched/unmatched strings into a delta file $\Delta_{A,B}$. Xdelta can quickly recover the file A by decoding the delta file $\Delta_{A,B}$ using file B. It is noteworthy that delta compression predates deduplication by a decade or more but adding delta compression to deduplication is gaining attention recently, as discussed in Section III-D.

Data deduplication was proposed in the 2000s to support global compression in large-scale storage systems at a much coarser granularity (e.g., a file [10] or a data chunk of 8K kB [11]) than the traditional compression approaches. It computes the *cryptographically secure hash-based fingerprints* of the files or chunks and then identifies the duplicates by matching their fingerprints.

Fig. 3 introduces a taxonomy of these redundant data reduction techniques and their evolution over the past several decades. Dates for each approach are approximate and indicate when research began. In many cases, research for each approach continues currently. These redundant data reduction techniques have been developed to cope with the increasing size of users' digital data, evolving from the entropy coding to dictionary coding, to delta compression, and now to data deduplication. These approaches are increasingly scalable by identifying redundancy at an increasingly coarse granularity, from the byte level to the string level, to the chunk level, and to the file level.

B. Key Features of Data Deduplication

In this section, we will study the key features of data deduplication. At present, the most widely used data deduplication approach eliminates redundancy at the chunk level in large scale storage systems by computing and then matching fingerprints of data chunks [5], [7]. It is worth noting that file-level deduplication was proposed earlier [10], [56], but that technique was subsequently overshadowed by chunk-level deduplication due to the latter's better compression performance [5], [12], [13], [57]. In some contexts, file-level deduplication gains most of the benefits of finer-grained comparisons [5].

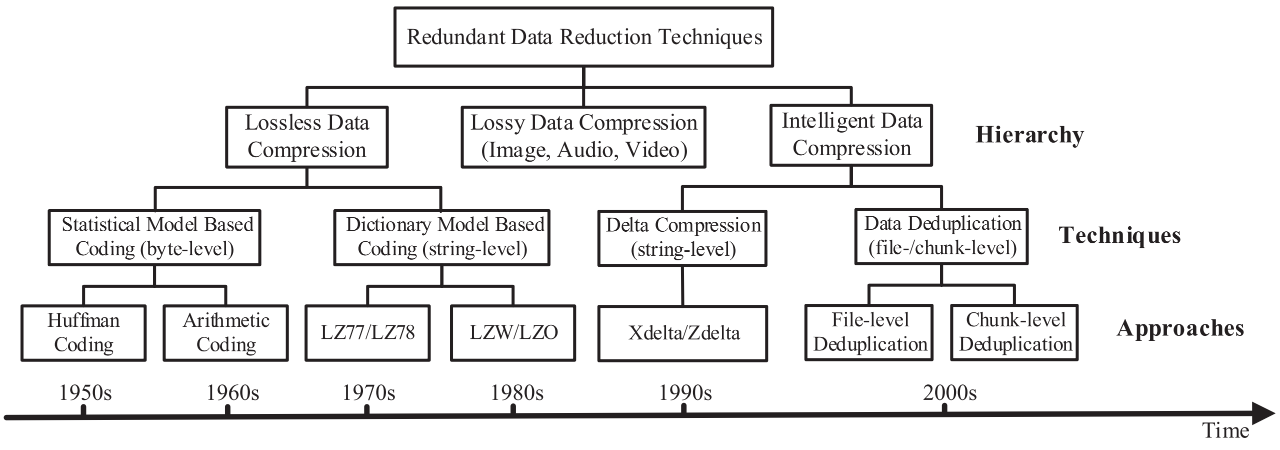

Redundant Data Reduction Techniques
Lossless Data Compression
Lossy Data Compression (Image, Audio, Video)
Intelligent Data Compression
Hierarchy
Statistical Model Based Coding (byte-level)
Dictionary Model Based Coding (string-level)
Delta Compression (string-level)
Data Deduplication (file-/chunk-level)
Techniques
Huffman Coding
Arithmetic Coding
LZ77/LZ78
LZW/LZO
Xdelta/Zdelta
File-level Deduplication
Chunk-level Deduplication
Approaches
1950s
1960s
1970s
1980s
1990s
2000s
Time

Fig. 3. ***Taxonomy of redundant data reduction techniques and approximate dates of initial research for each approach.***

The low-bandwidth network file system (LBFS) [14] and Venti [11] are two pioneering studies of data deduplication. LBFS was proposed to detect and eliminate duplicate chunks of about 8 kB size (using a variable-sized chunking approach as detailed in Section III-A) and thus avoid transmitting them over a low-bandwidth network environment, whereas Venti was proposed to remove duplicate chunks and save storage space. Specifically, LBFS employs content-defined chunking to identify redundancy at the chunk level while Venti uses fixed-size chunking. On the other hand, both Venti and LBFS adopt the SHA1 hash function for computing fingerprints of chunks and consider the probability of two different chunks having the same SHA1 value to be far lower than the probability of hardware bit errors [11], [13], [14]. The schemes of chunk-level duplicate elimination and SHA1-based fingerprinting have been widely accepted and adopted by academia and industry alike in the last ten years [13], [58]–[64]. Issues with the possibility of collisions have been raised [65] and refuted [66], as discussed further below.

Chunk-Level Duplicate Identification: While the size of digital data is consistently growing in recent years, the information entropy is not increasing proportionally. For example, a large portion of high-volume data can be a result of some data being repeatedly copied and stored in backup/archive storage systems [7]. In addition, traditional compression approaches use a byte-level sliding window to find matching strings for duplicate identification within a relatively small window; by comparison, data deduplication divides the input into nonoverlapped and independent chunks across a storage system.

Generally speaking, the chunking approaches for data deduplication include fixed-size chunking and variable-size chunking. Fixed-size chunking simply divides the input into fixed-size chunks according to the offset (i.e., position) of the content. This approach is simple but may fail to identify substantial redundancy because of the known *boundary-shift problem*, in which a minor change (e.g., insertion/deletion of some bytes) in the data stream can lead to the shift of boundaries for all chunks. Since the chunks contain slightly different content, they fail to deduplicate [14], [60], [67], [68]. Variable-size chunking, also called content-defined chunking, divides the input into variable-size chunks according to the content itself; this largely solves the boundary-shift problem and is the most widely used chunking approach today. Chunking will be studied in Section III-A.

Cryptographically Secure Hash-Based Fingerprinting (Also Called Compare by Hash) [65], [66]: The fingerprinting technique simplifies the process of duplicate identification. In some early data reduction approaches (such as LZ compression [16], [17] and Xdelta [49]), the duplicates are first matched by their calculated weak hash digest and then further confirmed by a byte-by-byte comparison. In data deduplication systems, the duplicates are completely represented by their cryptographic hash-based fingerprints (e.g., SHA1, SHA256) and the matched fingerprints mean that their represented contents are, with high probability, identical to each other. Here the fingerprint refers to a family of cryptographic hash functions [69] that has the key property of being practically infeasible to 1) find two different messages with the same hash and 2) generate a message from a given hash.

According to the "birthday paradox" [11], [70], the collision probability of a given SHA1 pair can be calculated as follows:

$$\text{hash calculation: } CA = \text{hash(content)}\ (CA \text{ length of } m \text{ bits})$$
$$\text{hash collision: } p \le \frac{n(n-1)}{2} \times \frac{1}{2^m}\ (n \text{ is a number of chunks}) \tag{3}$$

Table 1 Hash Collision Probability Analysis of SHA-1, SHA-256, and SHA-512 in Storage Systems with Difference Sizes of Unique Data and with an Average Chunk Size of 8 KB

Size of unique data	SHA-1 160 bits	SHA-256 256 bits	SHA-512 512 bits
1GB (2^{30}B)	10^{-38}	10^{-67}	10^{-144}
1TB (2^{40}B)	10^{-32}	10^{-61}	10^{-138}
1PB (2^{50}B)	10^{-26}	10^{-55}	10^{-132}
1EB (2^{60}B)	10^{-20}	10^{-49}	10^{-126}
1ZB (2^{70}B)	10^{-14}	10^{-43}	10^{-120}
1YB (2^{80}B)	10^{-8}	10^{-37}	10^{-114}

where CA is the *m*-bit fingerprint of the content using the "hash" function. Table 1 shows the hash collision probability according to (3) with varying amounts of unique data. The probability of a hash collision when data deduplication is carried out in an EB-scale storage system, based on the average chunk size of 8 kB and fingerprints of SHA1, is smaller than 10^{-20} (about 2^{-67}). In contrast, in computer systems, the probability of a hard disk drive error is about $10^{-12} \sim 10^{-15}$ [71], [72], which is much higher than the aforementioned probability of SHA1-fingerprint collisions in data deduplication. Consequently, SHA1 has become the most wildly used fingerprinting algorithm for data deduplication because most existing approaches, such as LBFS [14], Venti [11], and DDFS [13], consider the hash collision probability to be sufficiently small to be ignored when applying deduplication in a PB-scale storage system. Nevertheless, Henson [65] points out, however, that the hash-based comparison approach is not risk-free. Black *et al.* [66] offer several arguments in support of compare-by-hash by addressing concerns raised by Henson. More recently, stronger hash algorithms, such as SHA256, have been considered for fingerprinting in some data deduplication systems, such as ZFS [73] and Dropbox [74], to further reduce the risk of hash collision.

Content-defined chunking and secure-hash based fingerprinting are widely used in commercial storage systems [7], [13], [58], [59], [62], [64] and known as the key features of data deduplication. Table 2 shows a summary of analyses on data reduction ratio of large-scale real-world workloads (i.e., TB-PB-scale data sets) conducted by Microsoft [5], [6], EMC [7], [8], and Mainz University [70]. Here chunk-level deduplication is based on content-defined chunking [14] and delta compression is based on Xdelta encoding [49]. Table 2 indicates that data deduplication at the chunk level detects much more redundancy than that at the file level and duplicate chunks are abundant in computer systems, with data reduction ratios of about 69% ~ 97%, 42% ~ 68%, and 20% ~ 30% in secondary storage, primary storage, and HPC data centers, respectively.

C. Basic Workflow of Data Deduplication

A typical data deduplication system follows the workflow of chunking, fingerprinting, indexing, further compression, and storage management, as illustrated in Fig. 4. The storage management in data deduplication systems can be further specified into several categories, including data restore (fragment elimination), garbage collection, reliability, security, etc. Since data deduplication is designed to implement global redundant data reduction in large-scale storage systems, as discussed in Section II-A, there are a host of interesting research problems and solutions related to each stage of the data deduplication workflow, which can be classified as follows and elaborated on in the remainder of this paper.

- Chunking of data stream: The problem of how to design an efficient chunking algorithm to maximally detect redundancy in the data stream for data deduplication is an important research and practical problem, which, along with its solutions, will be studied in depth in Section III-A.

Table 2 Summary of Recently Published Analyses on Redundant Data Reduction Ratios of Large-Scale Real-World Datasets by Industry and Academia

Institutes	Workloads	Size	Redundant Data Reduction Schemes	Reduction Ratio
Microsoft	857 desktop computers [5]	162 TB	File-level deduplication internal users	about 21%
			Chunk-level deduplication internal users (8KB)	about 42%
			File-level deduplication across users	about 50%
			Chunk-level deduplication across users (8KB)	about 68%
	15 MS file servers [6]	6.8 TB	File-level deduplication	0~16%
			Chunk-level deduplication (64KB)	15~90%
EMC	Over 10,000 production backup systems [7]	700 TB	Chunk-level deduplication (8KB)	69~93%
	6 large backup datasets [8]	33 TB	Chunk-level deduplication (8KB)	85~97%
			Deduplication + delta compression	66~82%
			Deduplication + delta compression + GZIP compression	74~87%
Mainz Univ.	Four HPC data centers [70]	1212 TB	File-level deduplication	5~10%
			Chunk-level deduplication (8KB)	20~30%

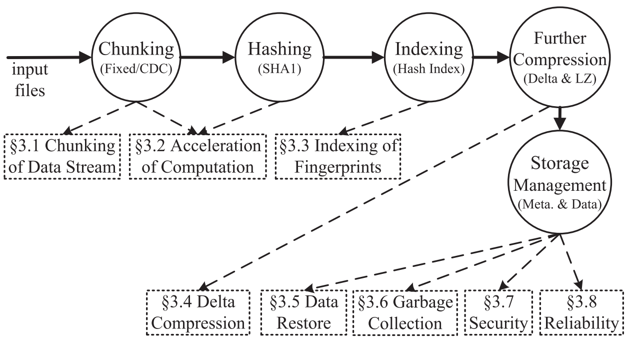


Fig. 4. ***Basic workflow of data deduplication.***

- Acceleration of computation tasks: Chunking and fingerprinting in data deduplication systems consumes significant computational resources. Reducing this overhead is an emerging and challenging problem, which will be examined in Section III-B.
- Indexing of fingerprints: As the size of the user data continues to grow from TB to PB and even EB scale, the total size of the fingerprints representing the user data will quickly overwhelm the main memory. This gives rise to the problem of how to efficiently store and index these fingerprints of the user data chunks stored on the disks, an increasingly important and challenging research and practical problem. We will comprehensively study the existing solutions to this problem in Section III-C.
- Further compression: How to eliminate the redundancy among the nonduplicate but very similar chunks in data deduplication systems, namely, delta compression, is another interesting and important research problem. As shown in Fig. 4, the delta and traditional compression (e.g., LZ compression) approaches belong to the further compression category. LZ compression is intuitive and easy to implement [7], [8], [13] while delta compression is an optional approach since it brings new challenges to data deduplication systems, which will be studied in depth in Section III-D.
- Data restore: In data-deduplication-based storage systems, the chunks of files or data streams may be physically scattered and stored after deduplication, which can significantly decrease the performance of restoration (read). The state of the art works on improving reading performance of deduplication-based storage systems will be comprehensively studied in Section III-E.
- Garbage collection: Some data chunks will be shared by many files in deduplicated systems, thus it is a new challenge to determine which chunks can be reclaimed when files are deleted. The garbage collection issue and existing solutions will be studied in Section III-F.
- Security: The deduplication among data of different users can result in potential security risk in which one user's private and sensitive information is leaked to another and *vice versa*. The secure deduplication issues and preliminary solutions will be studied in depth in Section III-G.
- Reliability: Data deduplication reduces the reliability of the storage system because the loss of a few critical data chunks can lead to many referenced files/backups being lost. The reliability-enhanced solutions for deduplication systems will be comprehensively studied in Section III-H.

III. DATA DEDUPLICATION: AN IN-DEPTH EXAMINATION OF KEY TECHNOLOGIES

In this section, we examine the state-of-the-art works on data deduplication in sufficient depth to understand their key and distinguishing features. These features divide the existing data deduplication technologies into eight categories (Section III-A–H) based on the workflow of the data deduplication process, as shown in Fig. 4. Existing approaches in each category will be comprehensively studied and discussed in each section.

A. Chunking of Data Stream

Chunking is the first step in the data deduplication process, in which a file or data stream is divided into small chunks of data so that each can be fingerprinted (see Fig. 4). The simplest chunking approach is to cut the file/data stream into equal, fixed-sized chunks, an approached referred to as fixed-size chunking (FSC). In FSC, if a part of a file or data stream, no matter how small, is modified by the operation of insertion or deletion, not only is the data chunk containing the modified part changed but also all subsequent data chunks will change, because the boundaries of all these chunks are shifted. This can cause otherwise identical chunks (before modification) to be completely different, resulting in a significantly reduced duplicate identification ratio of FSC-based data deduplication.

In order to address this boundary-shift problem [12], [14], [75], [76], the content-defined chunking (CDC) algorithm, also called content-based breakpoint chunking, was proposed in LBFS [14], to chunk files or data streams for duplicate identification. Specifically, CDC uses a sliding-window technique on the content of files and computes a hash value (e.g., Rabin fingerprint [77]) of the window, as shown in Fig. 5. A chunk breakpoint is determined if the hash value of this sliding window

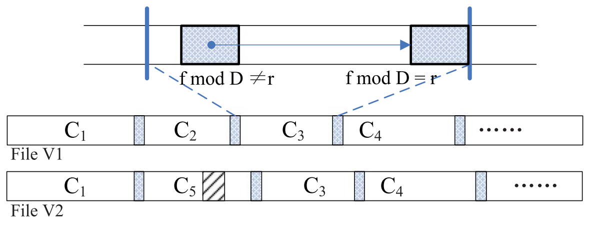


Fig. 5. ***Sliding window technique for the CDC algorithm. The hash value of the sliding window f is computed via the Rabin algorithm. If the lowest $\log_2 D$ bits of the hash value match a predefined value r, i.e., f $\bmod D = r$, the offset is marked as a chunk breakpoint (also called a cut point). Here the shaded region inside chunk C_5 of file V2 indicates the newly inserted content in file V2 relative to file V1.***

satisfies some predefined condition. Therefore, to chunk a file $V2$ that is modified on the chunk C_2 of file $V1$, the CDC algorithm can still identify the correct boundary of chunks C_3 and C_4 whose content has not been modified.

Currently, the Rabin algorithm [70], [77], [87] is the most widely used algorithm for computing the hash value of the sliding window in CDC for data deduplication. Specifically, the Rabin signature (fingerprint) for a sliding window (byte sequence $\{B_1, B_2, \ldots, B_\alpha\}$) of the data stream, is defined as

$$\mathrm{Rabin}(B_1, B_2, \ldots, B_\alpha) = \left\{ \sum_{x=1}^{\alpha} B_x p^{\alpha - x} \right\} \bmod D \tag{4}$$

where D is the average chunk size and α is the number of bytes in the sliding window. The Rabin signature is obtained by a rolling hash algorithm 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:

$$\begin{aligned}
&\mathrm{Rabin}(B_{i+1}, B_{i+2}, \ldots, B_{i+\alpha}) \\
&= \left\{ \sum_{x=i+1}^{i+\alpha} B_x p^{\alpha - x + i} \right\} \bmod D \\
&= \left\{ \left[\sum_{x=i}^{i+\alpha-1} B_x p^{\alpha-x+i-1} - B_i p^{\alpha-1} \right] p + B_{i+\alpha} \right\} \bmod D \\
&= \left\{ \left[\mathrm{Rabin}(B_i, B_{i+1}, \ldots, B_{i+\alpha-1}) - B_i p^{\alpha-1} \right] p \right. \\
&\quad \left. + B_{i+\alpha} \right\} \bmod D.
\end{aligned} \tag{5}$$

Therefore, building the new Rabin hash can be quickly computed from the old one, only requiring operations of two XOR, one OR, two left shifts, and two array lookups per byte [68]. In data deduplication systems, although Rabin meets the needs of content-defined chunking in terms of the rolling hash property, its efficiency can be improved. For example, a recent study, called QuickSync [88], suggests that Rabin-based CDC is quite computationally expensive for deduplication-based synchronization in mobile cloud storage. Generally speaking, Rabin-based CDC has the three main deficiencies of high chunk size variance, computational overhead, and inaccuracy of duplicate detection.

Table 3 summarizes the sate-of-the-art approaches to CDC-based data deduplication that address these three deficiencies of Rabin-based CDC, which are elaborated on below.

Reducing Chunk Size Variance by Imposing Limits on MAX/MIN Chunk Sizes for CDC: The cumulative distribution of chunk size X in a Rabin-based chunking approach

Table 3 State-of-the-Art Approaches to Content-Defined Chunking for Data Deduplication

Name	Source	Unique Features	Taxonomy
LBFS	SOSP'01 [14]	With max/min/average chunk size of 64KB/2KB/8KB	Restrictions on the max/min chunk size
TTTD	HP Tech. Report [78]	A threshold each for the maximum and minimum chunk sizes	
Regression Chunking	USENIX'12 [6]	Multiple thresholds for the maximum chunk size	
Fingerdiff	MSST'06 [79]	Merge consecutive duplicate or unique chunks into bigger ones	
MAXP	MS Tech. Report [80]	Treat the symmetric extremum in a fixed-size region as cut-points	
SampleByte	NSDI'10 [81]	Skip a half of the expected chunk size to accelerate chunking	Improving speed of CDC
Gear	Performance'14 [68]	Reducing hash calculation by a random integer table	
AE	Infocom'15 [82]	Use an asymmetric sliding window to identify cut-points	
Leap-based CDC	MSST'15 [83]	Make the sliding window jump forward to reduce calculation	
Bimodal Chunking	FAST'09 [75]	Re-chunk the unique but duplicate-adjacent chunks	Further re-chunking the non-duplicate chunks
Subchunk	SYSTOR'11 [84]	Re-chunk all the unique chunks into smaller ones	
FBC	MASCOTS'10 [85]	Re-chunk the non-duplicate chunks with high frequency	
MHD	ICPP'13 [86]	Dynamically merge & re-chunk the non-duplicate chunks	

with an average chunk size of 8 kB follows the following exponential distribution [68]:

$$P(X \leq x) = F(x) = 1 - e^{-\frac{x}{8192}}, \qquad x \geq 0. \tag{6}$$

Equation (6) suggests that there will be many chunks of extremely small or large size (e.g., < 1 kB or > 64 kB). Chunks of a smaller size cause higher metadata overhead for deduplication because this overhead is proportional to the number of chunks, whereas, chunks of a larger size decrease deduplication ratio [75] as large chunks tend to hide duplicates from being detected. Therefore, LBFS imposes the restrictions on the minimum and maximum chunk sizes (e.g., 2 and 64 kB) for content-defined chunking [14]. This solution, however, may result in forced cut points during chunking that are no longer content defined.

Two-thresholds–two-divisors (TTTD) [78] introduces an additional threshold for chunking, which has a higher probability of finding cut points and decreases the chunk size variance. Regression chunking [6] is similar to TTTD, but it uses multiple thresholds to reduce forced cut-point declarations at the maximum chunk size. MAXP [80], [89], [90] treats the extreme values in a fixed-size region as cut points, which also results in smaller chunk size variance. FingerDiff [79], [91] uses a smaller expected chunk size to detect more duplicates and then merges the consecutively duplicate or unique chunks to amortize the additional metadata overhead stemming from the smaller expected chunk size.

Reducing Computation to Accelerate the Chunking Process: Since the frequent computations of Rabin fingerprints on the sliding window are time consuming, many alternatives to Rabin have been proposed to accelerate the CDC process for deduplication [68], [81], [82]. SampleByte [81] is designed for providing fast chunking for fine-grained network redundancy elimination. It uses one byte to declare a fingerprint for chunking (while Rabin uses a sliding window) and increases the minimum size from one quarter of the expected average to one half, and compared to LBFS [14] skips the minimum size before searching for a cut point for each chunk. Gear [68] uses fewer operations to generate rolling hashes by using a random integer table to map the ASCII values of the content, so as to achieve higher chunking throughput while obtaining comparable chunking accuracy to Rabin. Asymmetric extremum (AE) [82] employs an asymmetric sliding window, rather than a symmetric sliding window as in MAXP [80], to identify extrema as cut points, which further reduces the computational overhead for chunking and thus achieves high chunking throughput while maintaining low chunk-size variance. Yu *et al.* [83] adjusted the function for selecting chunk boundaries such that if weak conditions are not met, the sliding window can jump forward, saving unnecessary calculation steps.

Improving Duplicate-Detection Accuracy by Rechunking Nonduplicate Chunks: As shown in Fig. 5, the CDC approach simply determines the chunk boundary if the hash (e.g., Rabin [77]) of the CDC sliding window matches a predefined value, which helps identify the duplicate chunks C_3 and C_4 among files $V1$ and $V2$ but fails to identify the redundancy between chunks C_2 and C_5. More specifically, the CDC approach cannot accurately find the boundary between the changed regions (e.g., the shaded region in chunk C_5 of file $V2$) and the duplicate regions (e.g., the blank unshaded region in chunk C_5) between two similar files (e.g., files $V1$ and $V2$). Thus, a number of rechunking approaches have been proposed to further divide the nonduplicate chunks (e.g., chunks C_2 and C_5 in Fig. 5) into smaller regions to detect more redundancy [75], [84]–[86].

Bimodal chunking [75] first divides the data stream into large chunks, and then rechunks the nonduplicate but duplicate-adjacent chunks into smaller chunks to detect more redundancy. Subchunk [84] is similar to bimodal chunking, but it rechunks all of the nonduplicate chunks for higher deduplication ratio. Frequency-based chunking (FBC) [85] uses a statistical chunk frequency estimation algorithm to identify the frequent chunks and then rechunks those chunks into smaller ones to detect more duplicates. Metadata harnessing deduplication (MHD) [86] is also similar to bimodal chunking, but it further reduces metadata overhead of deduplication by dynamically merging multiple nonduplicate chunks into one big chunk and meanwhile dividing nonduplicate but duplicate-adjacent chunks into smaller ones while bimodal only does the latter.

MAXP [80], [89], [90] and SampleByte [81] are two nonrolling-hash-based chunking algorithms that are designed for eliminating redundant network traffic at fine granularity (e.g., 64 or 128 B). MAXP was proposed to deal with the problem of the reduced deduplication ratio in Rabin due to the forced minimum chunk size [80]. It is used in network deduplication due to its smaller variance in chunk sizes [89]. SampleByte was proposed to accelerate content-defined chunking for improved energy efficiency of redundancy elimination in resource-constrained devices, such as mobile smartphones [81].

In summary, the CDC approaches imposing minimum and maximum chunk sizes are widely accepted and thus used in data-deduplication-based storage systems [5], [7], [13], [92]. Imposing more restrictions on Max/Min chunk sizes, while simple to implement, only marginally improves the deduplication ratio. Rechunking schemes significantly improve the deduplication ratio, but at the cost of the time-consuming rechunking process and the additional storage management of the nonduplicate

Table 4 State-of-the-Art Approaches to Accelerating Computational Tasks for Data Deduplication

Name	Source	Unique Features	Taxonomy
THCAS	ICPADS'09 [93]	Pipeline CPU-bound, I/O-bound, network communication tasks of deduplication	Multicore based
HPDS	USENIX'11 [62]	An event-driven client pipeline of FSC based deduplication tasks	
Ma et al.	MSST'12 [94]	Adaptive pipeline of deduplication, compression, encryption, etc.	
P-Dedupe	NAS'12 [76]	Pipeline deduplication tasks and further parallelize CDC and fingerprinting	
StoreGPU	HPDC'08 [95]	Develop a GPU library for deduplication with shared memory management	GPGPU based
Shredder	FAST'12 [96]	Develop a GPU acceleration framework of streaming pipeline & memory coalescing	
GHOST	PMAM'12 [97]	Offload the deduplication processing to GPGPU in primary storage	

chunks and their rechunked subchunks. Other fast chunking approaches, such as Gear [68], AE [82], and SampleByte [81], may be recommended if higher deduplication throughput is required by users.

Impact of Interspersed Metadata: Chunking of unchanged data can result in different chunks when metadata is interspersed with the data [98]. The metadata can take different forms. One type occurs when there are headers of fixed-sized blocks (e.g., for virtual tapes), which cause the metadata to move relative to content-defined chunks; these can be detected by the deduplication system and stripped prior to deduplication. Another type is per-file metadata preceding a file contained in a *tar* aggregate file or a backup file. When the per-file metadata contains timestamps, sequence numbers, or other variable content, the surrounding chunk will not deduplicate. Such metadata can be addressed by preprocessing the input (termed *migratory tar* by Lin, *et al.* [98]) or by creating deduplication-friendly file formats that segregate metadata from data.

B. Acceleration of Computational Tasks

As shown in Fig. 5 and Section III-A, data deduplication is a compute-intensive process that contains two time-consuming computational tasks, i.e., content-defined chunking and secure-hash-based fingerprinting. The former divides data streams into several chunks by CDC and the latter computes a cryptographic digest (i.e., fingerprint) for each chunk to uniquely represent it for duplicate detection. Thus, the chunking and fingerprinting stages of the deduplication process need to compute hashes (e.g., Rabin and SHA1), which may lengthen the write latency in a deduplication-based storage system [95], [96], [99], especially in a high-performance primary storage system using flash-based devices to the increase in-memory processing capacity [97].

While hash calculations for deduplication are time consuming and central processing unit (CPU) intensive, modern computer systems based on multicore/manycore processors or general-purpose computing on graphics processing units (GPGPU) processors are providing increasingly more computing resources [95]. Meanwhile, the data deduplication process can be divided into several independent subtasks as illustrated in Fig. 5. Exploiting parallelism among these subtasks [96], [100] can speed up deduplication by making full use of the abundant computing resources in modern computer systems. Note that the approaches discussed in Section III-A increase the CDC speed by improving the internal chunking process (e.g., using new hashing algorithms) while this section mainly discusses approaches to accelerating Rabin-based CDC externally by leveraging the multicore processors or GPGPU devices to exploit the parallelism of the CDC process.

Our studies show that the state-of-the-art approaches to accelerating the compute-intensive deduplication tasks are broadly based on either exploiting the parallelism of data deduplication with multiple threads [62], [100] or integrating the data deduplication process into the GPGPU hardware architecture [95], [96], [101]. These are summarized in Table 4 and elaborated on below.

- Multithread-based methods. THCAS [93] proposes a storage pipeline of CPU bound (i.e., chunking and fingerprinting), I/O bound (i.e., writing), and network communication tasks in its deduplication system. P-Dedupe [76] is similar to THCAS, but it further parallelizes the subtasks of chunking and fingerprinting and thus achieves higher throughput. Guo *et al.* [62] propose an event-driven, multithreaded client–server interaction model, to pipeline FSC-based deduplication. Ma *et al.* [94] propose an adaptive pipelining model for the computational subtasks of fingerprinting, compressing, and encrypting in FSC-based deduplication systems.
- GPGPU-based methods. GPGPU devices have been shown to offer stronger computing power than CPU for many compute-intensive applications, especially for the applications of hash and cryptographic calculation in high performance storage systems. StoreGPU [95], [101] and Shredder [96] make full use of GPGPU's computational power to accelerate popular compute-intensive primitives (i.e., chunking and fingerprinting) in data deduplication. Similarly, GHOST [97] offloads the deduplication tasks of chunking, fingerprint, and indexing to GPGPU to remove computing bottlenecks in high-performance primary storage systems.

Table 5 State-of-the-Art Approaches to Fingerprints Indexing for Data Deduplication

Name	Source	Scheme	Unique Features	Taxonomy
DDFS	FAST'08 [13]	Exact	Bloom Filter + locality preserved caching	Locality-based
Sparse Indexing	FAST'09 [58]	Approx.	Sparse sampling + locality exploited	
MAD2	MSST'10 [104]	Exact	Bloom Filter Arrary + locality exploited	
SAM	ICPP'10 [105]	Exact	Exploit file semantics of size, type, locality, etc.	
HPDS	USENIX'11 [62]	Approx.	Progressive sampled indexing + locality exploited	
BLC	SYSOR'13 [106]	Exact	Cache the locality of the last backup stream	
Extreme Binning	MASCOTS'09 [60]	Approx.	Use file representative fingerprints for primary index	Similarity-based
Aronovich et al.	SYSTOR'09 [107]	Approx.	Construct Karp-Rabin similarity signatures	
SiLo	USENIX'11 [63, 102]	Approx.	Combined exploitation of similarity and locality	
DedupeV1	MSST'10 [108]	Exact	Chunk fingerprints stored on SSDs	Flash-assisted
ChunkStash	USENIX'10 [61]	Exact	Cuckoo hash + fingerprints stored on flash	
BloomStore	MSST'12 [109]	Exact	Bloom-Filter based Key-Value Store + flash assisted	
HYDRAstor	FAST'09 [59]	Exact	Fingerprint prefix based stateless routing	Cluster deduplication
DEBAR	IPDPS'10 [110]	Exact	Post-processing deduplication by a shared global index	
Dong et al.	FAST'11 [111]	Approx.	Super-chunk based routing by exploiting similarity	
$\sum$-Dedupe	Middleware'12 [112]	Approx.	Similarity and locality based data routing	
Kaiser et al.	MSST'12 [113]	Exact	Design a joint distribute chunk index	
Produck	SOCC'12 [114]	Approx.	Probabilistic based data routing	

In summary, multithread-based solutions can be easily implemented in computer systems with multicore/manycore processors by pipelining deduplication tasks and parallelizing chunking and fingerprinting [93], [100]. GPGPU-based solutions can provide higher throughput but require additional hardware costs [95], [96], [101]. Another approach, which does not affect the speed of the computational tasks but allows a backup system to scale to additional clients, is to offload chunking to the clients; this is discussed in the context of industry trends in Section IV-B.

C. Indexing of Fingerprints

After chunking and fingerprinting the data streams, chunk fingerprints are indexed to help determine the duplicate and nonduplicate data chunks, which is a critical stage for the deduplication process. Early deduplication systems store the entire chunk fingerprint index in memory for fast duplicate identification [14].

With the explosive growth of data volume, the total number of fingerprints and thus the size of their index increase exponentially, quickly overflowing the RAM capacity of deduplication systems. This can result in frequent accesses to the low-speed disks for fingerprint-index lookups, thus severely limiting the throughput of deduplication systems. For example, to backup a unique data set of 1 PB and assuming an average chunk size of 8 kB, about 2.5 TB worth of SHA-1 fingerprints (160 bits each chunk) will be generated. The 2.5-TB fingerprints plus extra location and index messages for each chunk (e.g., 8 B), will be too large to be fully stored in the main memory of a typical deduplication system. Since the random accesses to on-disk index are much slower than that to RAM, frequent accesses to on-disk fingerprints will cause the system throughput to become unacceptably low. For example, some data deduplication systems [11], [13], [58] report that the accessing throughput to the on-disk fingerprint-index is about 1–6 MB/s, which becomes a severe performance bottleneck in these systems. Therefore, an efficient fingerprint-indexing scheme is necessary for large-scale data deduplication systems.

Depending on the specific approach used, a fingerprint-indexing scheme can lead to either exact deduplication or approximate deduplication [102], [103]. While the former means that all duplicate chunks are eliminated, the latter trades a slightly reduced accuracy of duplicate detection (i.e., a small number of duplicate chunks are not detected) for a higher index-lookup performance and lower memory footprint. Currently, there are four general categories of approaches to accelerating the index-lookup process of deduplication and alleviating the disk bottleneck, namely, locality-based, similarity-based, flash-assisted, and cluster deduplication approaches. Table 5 presents these four categories of the state-of-the-art approaches to fingerprint indexing for data deduplication systems, which are elaborated on upon next.

Locality-Based Approaches: Locality in the context of data deduplication refers to the observation that similar or identical files, say, *A*, *B*, and *C* (thus their data chunks), in a backup stream appear in approximately the same order throughout multiple full backups with a very high probability [13], [75]. Mining this locality for deduplication indexing increases the RAM utilization and reduces the accesses to on-disk index, thus alleviating the disk bottleneck.

Fig. 6 shows an example of the locality-based approach, DDFS [13]. This well-known deduplication system makes full use of this locality property by storing the chunks in the order of the backup stream (e.g., chunks' fingerprints {3b, a7, 2f, 5c} of file *A*) on the disk. Upon the lookup of fingerprint "3b" of file *C*, DDFS will prefetch the fingerprints {3b, a7, 2f, 5c} and preserve this locality in the RAM, which helps reduce the

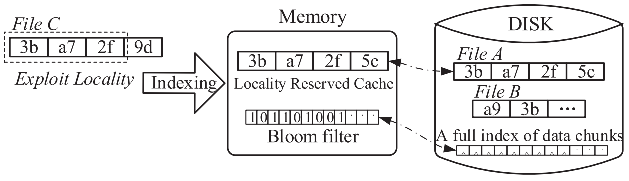


Fig. 6. ***Typical locality-based fingerprints indexing approach: DDFS [13].***

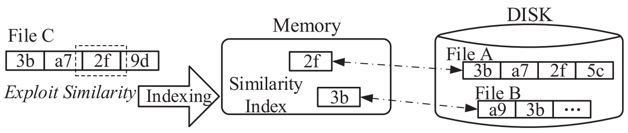


Fig. 7. ***Typical similarity-based fingerprints indexing approach: Extreme Binning [60].***

accesses to the on-disk index when looking up the fingerprints of “a7” and “2f” at a later time. Usually, DDFS stores the nonduplicate chunks in several large fixed-size storage units called containers that preserve backup stream locality. DDFS also uses Bloom filters [22] to quickly identify new (i.e., nonduplicate) chunks, avoiding an index lookup for chunks that are known not to already exist; this helps compensate for the cases where there is no or little locality. A Bloom filter [22] is a space-efficient data structure that uses a bit array with several independent hash functions to represent membership of a set of items (e.g., fingerprints).

Sparse indexing [58] improves DDFS memory utilization by sampling the index of chunk fingerprints in memory, instead of using Bloom filters [22] as in DDFS, which reduces the RAM usage to less than half of that in DDFS. SAM [105] first combines the global file-level deduplication and local chunk-level deduplication, and then exploits file semantics of size, type, locality, etc., to optimize fingerprint indexing. MAD2 [104] employs a Bloom filter array as a quick index for deduplication while also preserving locality of fingerprints in cache. HPDS [62] exploits the inherent locality of the backup stream with a progressive sampled indexing approach to further reduce memory overhead for fingerprint indexing. DDFS [13] captures locality by storing and prefetching in the order of the stored unique chunks in containers. Unlike DDFS, block locality caching (BLC) [106] improves indexing performance by exploiting the locality of the most recent backup in a long-term backup system.

Similarity-Based Approaches: The similarity in the deduplication context refers to the similarity characteristics of a file or a data stream to previous, similar files or data streams. A common similarity detection technique is to represent a file with the maximal or minimal value of the sets of chunk fingerprints [60]. As a result, the selected fingerprints can be used to build a primary index and minimize RAM overhead for deduplication indexing, especially for the data sets with little or no locality. Extreme binning [60] is a similarity-based approach that improves deduplication scalability by exploiting the file similarity to achieve a single on-disk index access per file for chunk lookup.

Extreme binning demonstrates that the probability of files S_1 and S_2 sharing the same representative fingerprint is closely correlated to their similarity degree according to Broder’s theorem [115]. That is, consider two sets S_1 and S_2, with $H(S_1)$ and $H(S_2)$ being the corresponding sets of the hashes of the elements of S_1 and S_2, respectively, where H is chosen uniformly and randomly from a min-wise independent family of permutations. Let $\min(S)$ denote the smallest element of the set of integers S. Then

$$\Pr[\min(H(S_1)) = \min(H(S_2))] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}. \qquad (7)$$

Fig. 7 shows an example of how extreme binning exploits file similarity by, where two sets of chunk fingerprints, {3b, a7, 2f, 5c} and {3b, a7, 2f, 9d}, belong to files *A* and *C*, respectively. Here the file similarity is represented by the minimal fingerprint in the hash set of a file whose prefix bits represent the smallest value among the same prefix bits of all the fingerprints there. Thus, in the event of the minimal fingerprint “2f” of file *C* being detected to be identical to that of file *A*, we can consider the two files similar and then detect duplicate chunks between files *A* and *C*, which avoids global indexing for the chunk fingerprints of file *C*.

Aronovich *et al.* [107] exploit the similarity of backup streams in large-scale deduplication systems. They divide the data stream into multiple large 16-MB blocks, where a signature is constructed for each of these blocks to detect similar blocks. For each detected pair of similar blocks, a byte-by-byte comparison is performed to identify and eliminate duplicate data. SiLo [102] jointly and complementarily exploits similarity and locality by first exploiting similarity of data segments (a group of chunks) to reduce the space of primary index in RAM and then mining locality to enhance duplicate detection through probabilistic similarity detection [see (7)].

A recent paper [103] presents a general-purpose and open-source framework, for comprehensively studying and evaluating many of the above discussed locality- and similarity-based approaches. It further clarifies the locality concept by dividing it into two categories, logical and physical locality, which means the chunk (fingerprint) sequence of a backup stream before and after deduplication, respectively. DDFS [13], HPDS [62], and ChunkStash [61] exploit physical-locality while sparse indexing [58], SiLo [102], and BLC [106] exploit logical locality. DeFrame also discusses the design tradeoffs among deduplication ratio, indexing memory footprint, and restore performance, which provides a detailed guide to make design decisions for deduplication systems.

Flash-Assisted Approaches: Because the lookup throughput of on-disk fingerprints is limited by the expensive disk seek operations (only about 100 input/output operations per second (IOPS)), the random-access flash memory has been proposed as an alternative to disks to provide high-throughput I/Os (about 100 000 IOPS) for fingerprint indexing [116]. The memory-efficient and high-performance primary-index design for on-flash fingerprints, called key-value store [109], [116], [117], is further exploited for these flash-assisted deduplication indexing approaches.

DedupeV1 [108] and ChunkStash [61] store the chunk fingerprints on flash memory instead of hard disk to accelerate the index-lookup process. ChunkStash preserves the backup-stream locality in the memory to increase the RAM utilization and reduce the accesses to on-flash index. In addition, Cuckoo hash [118] is used by ChunkStash to organize the fingerprint index as key-value store in RAM, which is shown to be more efficient than Bloom filters in DDFS. More importantly, ChunkStash uses an extra "stash" to avoid a possible loop from indexing fingerprints in the cuckoo hash table, hence the partial name "stash." FlashStore [116] and SkimpyStash [117] integrate data structures of Bloom filters [22] and hash tables for better key-value store performance. BloomStore [109] further amortizes the memory overhead and improves the lookup/insertion performance by employing a Bloom-filter-based index structure for efficient key-value store of on-flash fingerprints.

Cluster Deduplication: Most of the aforementioned approaches eliminate duplicates on a single node, which limits the throughput and scalability of data deduplication. This limitation has led to the development of cluster deduplication systems consisting of multiple nodes [59], [111], [112]. The basic idea of cluster deduplication is to assign data streams from backup clients to multiple deduplication nodes by a data routing scheme that supports internode load balance and enables independent intranode duplicate elimination in individual nodes [112].

HYDRAstor [59] performs deduplication at a coarse chunk granularity (64 kB) and distributes data at the chunk level to storage nodes based on the prefixes of fingerprints using distributed hash tables (DHT). DEBAR [110] first deduplicates files partially as they are written to server disks and then completes postprocessing deduplication at the chunk level by a global fingerprint index among nodes. Note that the extreme binning [60] and MAD2 [104] approaches, mentioned above, also support cluster deduplication by using a stateless routing approach based on distributed hash tables (DHT); this is similar to HYDRAstor.

Dong *et al.* [111] propose two superchunk-based data routing algorithms, where a superchunk is a group of contiguous chunks. One is a "stateless" routing algorithm, which uses the content within the superchunk (such as the first 64 B of the first chunk in the superchunk) to select a "bin," which is then mapped to a node. For load balancing, the mapping of bins to nodes can change over time. The other algorithm is a "stateful" routing approach, which exploits data similarity to direct a superchunk where it deduplicates the best. The benefit of deduplication is offset by any load imbalance, such that a node that currently stores more than the average total content needs a corresponding improvement in its deduplication to be selected as the target for the superchunk.

Fu *et al.* [112] propose a similarity and locality-based routing algorithm, called $\sum$-Dedupe, which further exploits the locality to alleviate the chunk index-lookup bottleneck in each node after distributing superchunks based on their similarity. Kaiser *et al.* [113] design a joint distributed chunk index for exact cluster deduplication, but it is partially limited by the internode communication, which has stringent bandwidth requirements. Produck [114] is a stateful routing approach similar to the superchunk approach [111], which further balances the load among cluster nodes with less overhead by a probabilistic similarity metric for data routing.

Summary: In general, locality-based approaches are widely used to improve deduplication indexing performance, whether fingerprints are stored on a hard disk drive [13] or flash drive [61]. Similarity-based approaches are shown to be effective in reducing the RAM overhead for deduplication indexing [60], [102], [103]. Flash-assisted approaches incur additional hardware costs for deduplication systems but their key-value store schemes can also be used for indexing on-disk fingerprints [108], [109]. Cluster deduplication approaches are scalable for massive storage systems but may decrease the deduplication ratio [111], [112], [114] or require more system resources to maintain a high deduplication ratio [59], [113].

D. Post-Deduplication Compression

Data deduplication has been widely deployed in storage systems for space savings. In practice, the fingerprint-based deduplication approaches fail to identify a significant fraction of redundancy. Each chunk typically has *internal* redundancy that can be removed with a traditional compressor such as LZ. If unique chunks that are added to the system together are compressed together in a larger "compression region," the overall compression rate will be higher than compressing

Table 6 State-of-the-Art Approaches to Resemblance Detection and Delta Encoding for Post-Deduplication Delta Compression

Name	Source	Unique Features	Taxonomy
Super-feature	CMP'00 [119]	Coalesce multiple deterministically sampled hashes into super-feature	Resemblance detection
REBL	USENIX'04 [120]	Use super-feature & delta encoding to eliminate redundancy at block level	
TAPER	FAST'05 [51]	Measure file similarity with a Bloom filter storing chunk fingerprints	
DE	OSDI'08 [121]	Identify similar pages based upon hashes of sub-pages	
SIDC	FAST'12 [8]	Use a memory-efficient and locality-based Super-feature cache	
DARE	DCC'14 [55]	Detect unique but duplicate-adjacent chunks as potential candidates	
Xdelta	UCB Tech. Rep. [49]	Copy/Insert instructions based and KMP otimized delta encoding	Delta encoding
Zdelta	NYU Tech. Rep. [122]	Incorporate Huffman coding from Zlib into delta encoding	
Ddelta	Performance'14 [68]	Delta encoding based on Gear-based content-defined chunking	

every chunk individually. DDFS, for instance, reports a typical 2× compression [7].

Beyond the simple compressibility of post-deduplication chunks, there can be high overlap between similar chunks that only contain a small number of different bytes, e.g., chunks C_2 and C_5 in Fig. 5. Even if there exist only several different bytes, secure-hash-based fingerprints of these chunks will be completely different [8], [11], [68]. As discussed in Section III-A, rechunking approaches were proposed to increase deduplication ratio by further dividing nonduplicate chunks into smaller ones [75], [84]–[86], which helps identify more redundancy. In contrast, post-deduplication delta compression eliminates redundancy among the nonduplicate but similar chunks without the need for rechunking operations to achieve a higher redundancy elimination ratio [8], [54], [55], [68]. As a result, it is considered an effective post-deduplication process to further remove data redundancy but adding extra computation, indexing, and I/O overheads, and thus an optional post-deduplication storage management stage (see Fig. 4).

The main challenges facing post-deduplication delta compression stem from the three time-consuming stages of resemblance detection, reading base chunks, and delta encoding [54]. Table 6 summarizes the state-of-the-art approaches to resemblance detection and delta encoding in delta compression; the read-back issue will be discussed later.

Resemblance Detection: In delta compression, the key research issue is how to accurately detect a fairly similar candidate for delta compression with low overheads. Manber [123] proposes a basic approach to find similar files in a large collection of files by computing a set of polynomial-based fingerprints (i.e., Rabin [77]); the similarity between two files is proportional to the fraction of fingerprints common between them. This approach has been used in DERD [124] to detect similar files and then delta encode them. The superfeature approach [119] is based on Broder's theorem [115] [see (7) in Section III-C]: deterministically sampling several Rabin fingerprints (e.g., using the minimum hash as in extreme binning [60]) of files or chunks as *features* and coalescing them into a superfeature, also referred to as *superfingerprint*, and then indexing the superfeatures to detect similar files or chunks. The superfeature approach is widely used for delta-compression-based redundancy elimination [8], [54], [120], [125].

TAPER [51] proposes an alternative to the superfeature method by representing each file with a Bloom filter that records chunk fingerprints, measuring file similarity based on the number of matching bits between Bloom filters, and then delta compressing the detected similar files. Difference engine (DE) [121] and I-CASH [126] make full use of the delta compression techniques to eliminate redundancy in memory pages and SSD caches, respectively, where they detect similar 4-kB pages using a parameterized scheme based upon computing the hashes of several 64-B subpages, which is similar to the superfeatures approach.

Stream-informed delta compression (SIDC) [8] shows that post-deduplication delta compression can further improve the data reduction ratio by a factor of $3 \sim 5$ when replicating between EMC's deduplicated backup storage systems (see Table 2 in Section II-B). Moreover, SIDC shows that the superfeature index for delta compression can also overflow the RAM capacity (similar to the problem of fingerprint indexing in Section III-C). SIDC leverages the locality of similar chunks, which have repeated patterns as discussed for duplicate chunks, so storing superfeatures in a stream-informed manner and prefetching into a cache captures most potential similarity detection, while reducing RAM requirements. SIDC is also applied to a storage system [54], which demonstrates new complexities in managing delta-encoded storage. Deduplication-aware resemblance detection (DARE) [55], [127] extends this work by detecting potential similar chunks for delta compression based on the existing duplicate-adjacent information after deduplication, i.e., considering two chunks similar if their respective adjacent chunks are determined as duplicate in a deduplication system.

Delta Encoding the Similar Chunks: Another challenge facing the delta compression is the time-consuming process of calculating the differences among similar data chunks. The efficiency of delta encoding becomes increasingly more important with the rapid growth of storage capacity and network bandwidth [54], [126]. Like traditional lossless compression approaches, Xdelta [122] uses a byte-wise sliding window to identify the matched (i.e., duplicate) strings between the base and the input chunks for the delta calculation. This is an open-source project widely used in many redundancy elimination systems [8], [55], [121]. Zdelta [68] incorporates the Huffman coding [40] for delta compression to further eliminate redundancy. Most recently, Ddelta [68] proposes a deduplication-inspired delta-encoding approach that divides similar chunks into several independent and nonoverlapped strings by a fast Gear-based content-defined chunking algorithm. This algorithm simplifies and thus accelerates the delta encoding process, while achieving a comparable data reduction ratio to Xdelta and Zdelta.

Additional Delta Compression Challenges: In addition to the above two challenges, other post-deduplication delta compression and storage management stages, e.g., reading base chunks, data restore, and garbage collection, are also important aspects of post-deduplication delta compression. Specifically, compressing relatively similar chunks requires the reading of the already-stored nonduplicate but resemblance-matched chunks for delta encoding with the input chunks. Shilane *et al.* [54] propose to store the unique chunks on SSDs to address the throughput bottleneck of delta compression caused by random accesses to on-disk base chunks. Additionally, the post-delta-compression data restore and garbage collection are still open problems and will be further discussed in Section IV-D.

E. Data Restore

After identifying duplicate data and storing the nonduplicate data, it is desirable to efficiently restore data and effectively manage the fragmented storage space (after users' deletion operations). The latter process is known as *garbage collection*. Thus, data restore and garbage collection have become two important problems in the storage management stage of data deduplication systems. This section mainly reviews the state-of-the-art schemes on data restore, and the garbage collection issue will be detailed in Section III-F.

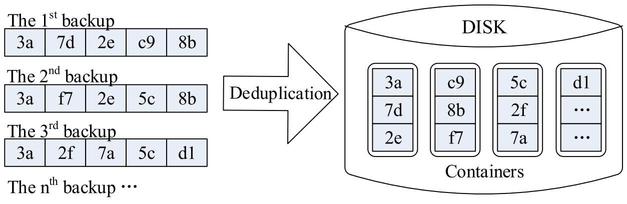


Fig. 8. ***Example of data fragmentation in deduplication-based backup storage systems. Note that a container is a fixed-size storage unit that stores sequential and nonduplicate chunks for better backup and restore performance by using large I/Os [13], [23], [103], [129].***

Fig. 8 shows an example of data fragmentation in a deduplication-based backup storage systems. The logically contiguous chunks in each backup (e.g., the third backup) are physically scattered in several data containers (a fixed-size storage unit) after deduplication, also referred to as chunk fragmentation [23], [128], [129], rather than being arranged in a compact continuous sequence in the traditional way. Due to the poor random I/O performance of HDDs, chunk fragmentation (like disk fragmentation) significantly decreases restore performance. In addition, the chunk fragmentation also hurts the performance of garbage collection [62], [129]. For example, if users delete the first backup in Fig. 8, it will be difficult to reclaim the storage space of the data chunks in the first backup since some of them are referenced by the second and third backups.

Several post-deduplication data restore schemes rewrite the duplicate, but fragmented, chunks to alleviate the degradation of the read (restore) performance, thus trading off the deduplication ratio (capacity savings) and read (restore) performance. Table 7 comprehensively studies the state of the art on improving restore performance in deduplication systems, and classifies them into three categories according to the storage environment in which deduplication is deployed, i.e., primary storage, backup storage, and cloud storage.

Primary storage systems are I/O-latency sensitive [6], [64] which makes the deduplication-induced,

Table 7 State-of-the-Art Approaches to Post-Deduplication Data Restore

Name	Source	Unique Features	Taxonomy
iDedup	FAST'12 [64]	Selectively deduplicate only sequences of disk blocks (large writes)	Primary storage systems
POD	IPDPS'14 [130]	Focus on performance-sensitive but capacity-insensitive small writes & files	
CFL-SD	MASCOTS'11 [128]	Defragment based on the metric of chunk fragment level (CFL)	Backup storage systems
CBR	SYSTOR'12 [131]	Selectively deduplicate based on stream context and disk context	
Capping	FAST'13 [23]	Defragment based on the write buffer (about 20MB) analysis	
RevDedup	APSYS'13 [132]	Optimizes reads to the latest backups using reverse deduplication	
HAR	USENIX'14 [129]	Defragmentation by exploiting historical information of backup systems	
CABdedupe	IPDPS'11 [133]	Identify unmodified data among chronological versions of datasets	Cloud storage systems
SAR	NAS'12 [134]	Store unique chunks with high reference count on SSDs	
NED	ICA3PP'14 [135]	Defragmentation t based on accurate segment reference analysis before backup	

read-latency-lengthening fragmentation issue extremely important. iDedup [64] exploits spatial locality of primary storage workloads to selectively deduplicate sequential duplicate disk blocks to reduce fragmentation and amortizes the read latency caused by random I/O. Performance-oriented I/O deduplication (POD) [130] further improves the read performance of deduplication-based primary storage systems by identifying capacity-insensitive but performance-sensitive small duplicate writes and files in the critical I/O path.

For deduplication-based backup storage, the restore speed for the most recent backup can drop by orders of magnitude over the lifetime of a backup system [23], [129] due to the chunk fragmentation problem. Nam *et al.* [128], [136] propose to selectively eliminate sequential and duplicate chunks with a quantitative metric called chunk fragmentation level (CFL), which is similar to iDedup [64]. Context-based rewriting (CBR) [131] and capping [23] algorithms determine the fragmented chunks in the write buffer (e.g., $10 \sim 20$ MB) using their specific fragmentation metrics, and then selectively write the fragmented chunks to improve restore speed. In addition, capping uses a forward assembly technique to efficiently cache chunks by exploiting the perfect knowledge of future chunk accesses available when restoring the already known backups.

Reverse deduplication (RevDedup) [132], [137] eliminates duplicates from the previous backups while conventional deduplication eliminates duplicates from the new backups. Thus, RevDedup shifts fragmentation to old backups, which keeps the layout of the latest backup as sequential as possible but at the cost of postprocess deduplication. History-aware rewriting (HAR) [129], [138] tries to rewrite less data to achieve better restoration performance by accurately classifying fragmentation into two categories: sparse containers and out-of-order containers, where a container is a 4-MB-size storage unit that stores chunks after deduplication. Sparse containers are identified from the last backup by exploiting historical information of backup versions. New chunks that are duplicates of chunks in sparse containers may be rewritten to improve data locality. Because the recipe containing the list of chunks for the entire file is known in advance, the out-of-order containers can be efficiently cached during the restore according to Belady's optimal caching scheme [139].

For deduplication-based cloud storage, the restore speed may be severely limited by the relatively low bandwidth of WAN or the frequent accesses to the fragmented chunks in the cloud servers. CAusality-Based Deduplication (CABdedupe) [133] identifies unmodified data among chronological versions of backup data sets between the client and cloud, thus improving the restore performance. SSD-assisted restore (SAR) [134], [140] stores unique chunks with high reference counts on SSDs in the cloud servers to improve restore performance, which utilizes the good random-read performance property of SSDs. Near-exact defragmentation (NED) [135] groups chunks into segments for cloud backup and identifies the fragmented segments before uploading them to the cloud according to a metric called segment reference ratio. This scheme achieves a comparable restore performance to that of a cloud backup system without deduplication.

In summary, the chunk fragmentation problem in deduplication systems results in degraded performance in data restore. State-of-the-art schemes strike appropriate tradeoffs between the capacity savings and performance penalty in deduplication. We believe that efficient and accurate defragmentation can significantly alleviate the orders-of-magnitude performance drop of restoration in deduplication-based storage systems [129].

F. Garbage Collection

This section discusses the state-of-the-art approaches to garbage collection (GC) in deduplication-based storage systems. Since unique chunks may be shared by multiple files, reference management is critical for tracking of chunk usage and to reclaim freed space (i.e., GC). In general, GC consists of two key steps, finding the invalid chunks and then reclaiming their storage space. The GC approaches can be generally divided into two categories according to the first step, namely, reference count and mark-and-sweep.

- Reference count approach and its variants. The chunk reference count of a particular chunk in a deduplication system refers to the number of times this chunk is used/referenced. For example, while a reference count of N means that the chunk is referenced N times (in $1 - N$ files), a 0 reference count value means that the chunk is no longer shared due to users' deletion operations and can be reclaimed for garbage collection. Wei *et al.* [104] employ this naive reference count approach for GC. However, accurately building a reference counter for each chunk is space inefficient [129]. In addition, Guo *et al.* [62] point out that reference counting may suffer from low reliability: When errors occur, some chunks may be updated and some may not. Simha *et al.* [141] propose a hybrid approach of maintaining both a reference count and an expiration time for each physical block for incremental backup. Strzelczak *et al.* [142] also maintain a reference counter and propose a concurrent deletion algorithm based on an epoch mechanism and undelete markers to separate old and newly written data, which allows for deletion to proceed concurrently with users' ongoing reads and writes in their commercial HYDRAstor system [59]. Dmdedup [143] also uses the reference count solution for block device deduplication, but it does

Table 8 State-of-the-Art Schemes to Secure Data Deduplication

Name	Source	Unique Features	Taxonomy
Douglas et al.	Patent(1995) [149]	First propose convergent encryption (CE) in a backup system patent	Convergent encryption and its variants
Farsite	ICDCS'02 [150]	Develop convergent encryption (CE) for file-level deduplication	
Pastiche	OSDI'02 [151]	Propose hash based CE (HCE) for chunk-level deduplication	
MLE	Eurocrypt'13 [155]	Propose randomized convergent encryption (RCE)	
DupLESS	USENIX SS'13 [156]	Introduce server-aided CE resisting brute-force attacks	
Dekey	IEEE TPDS'14 [154]	Use the Ramp secret sharing scheme for CE key management	
SecDep	MSST'15 [162]	Propose User-Aware CE and Multi-Level Key management	
Harnik et al.	IEEE S&P'10 [147]	Sometimes artificially turn off cross-user deduplication	Side channel attack
Gateway-based	IEEE NTMS'12 [157]	Dedupe happens between server and gateway rather than client	
PoWs	CCS'11 [148]	Introduce POW and resolved by Merkle trees and specific encodings	Proofs of ownership (POW)
s-POW	ASIA CCS'12 [159]	Use knowledge of randomly selected bit-string as POW of a file	
Xu et al.	ASIA CCS'13 [161]	Implement POW in the weak leakage-resilient setting	

not reclaim blocks immediately, nor does it remove the corresponding hashes from the index, which decreases the latency of the critical write path. Fu *et al.* [129] propose a container-marker algorithm (CMA) to simplify the chunk reference management by only recording the referenced containers while reducing the fragmented chunk in each container by their history-aware rewriting (HAR) algorithm.

- Mark-and-sweep approach and its variants. Mark-and-sweep is another GC solution that consists of two stages. During the mark stage, all files are traversed so as to mark the used chunks. In the sweep stage all chunks are swept and unmarked chunks are reclaimed. Grouped mark-and-sweep (GMS) [62] generates a bitmap to mark the valid chunks in each container (i.e., a group of chunks as shown in Fig. 8) referenced by each backup, and reclaims the storage space by merging the bitmaps of all containers to locate and reclaim the storage space occupied by all invalid chunks. Botelho *et al.* [144] build a perfect hashing vector [145] as a compact representation of all chunks and then traverse all the file recipes to reclaim the storage space of invalid chunks.

In summary, the GC problem in deduplication systems is caused by sharing chunks among files/backups after deduplication. Efficient chunk-reference management are critical for GC to identify the invalid chunks and reclaim their storage space in deduplication systems. The reference count solutions [104] and some of its variants [129] support GC inline while mark-and-sweep and its variants [62], [144] are inherently offline. Note that in deduplication-based backup systems, GC is usually performed as a background process after one or more complete backups have been deleted, but in primary storage systems, GC is usually performed inline or partly inline. Generally speaking, there is a tradeoff among the existing GC approaches. Specifically, the immediate GC increases the latency of the critical I/O path while the delayed GC decreases space utilization in deduplication-based storage systems.

G. Security

Deduplication-based storage systems face the inherently serious issues of security and reliability, especially in cloud storage systems, such as Dropbox, Wuala, and Mozy [146]. The security issue arises from users who share data chunks or files after deduplication, which can expose security vulnerabilities and privacy concerns in cloud storage systems.

Currently, cross-user deduplication faces the following three main security challenges. 1) Contradiction with encryption: different users may encrypt their data with their own keys. In these cases, identical data of different users will result in different ciphertexts, making deduplication impossible across different users. 2) Side-channel attacks: the occurrence of cross-user deduplication can be used as a side channel to reveal a user's private information by three types of attacks: identifying files, learning the contents of files, and establishing a covert channel [147]. 3) Proofs of ownership: there is an attack, using a small hash value (i.e., a fingerprint) as a proxy for the entire file, the client can prove to the server that it indeed has uploaded the file. Specifically, an attacker who knows the proxy fingerprints of a file can convince the storage server that it owns that file so that the server would allow the attacker to download the entire file [148].

Table 8 examines and classifies state-of-the-art approaches to secure data deduplication according to the three challenges above, which are elaborated upon next.

- Convergent encryption (CE) uses a hash obtained from the data content as a key to encrypt data. Hence, identical data from different users will generate the same ciphertext, which allows deduplication to work across different users after encryption. The method of combining content-based keying and deduplication was first described in a backup system patent in 1995 [149], then was developed and called convergent encryption in Microsoft's Farsite distributed file system [56], [150]. CE is widely used in many deduplication systems [151]–[154]. Message-locked encryption (MLE) [155] studies and compares CE and its variants, hash-based CE [151], randomized CE in practice and theory. To further ensure CE security, DupLESS [156] adds a message on the CE key from a key server via an oblivious RSA-based oblivious pseudorandom function (RSA-OPRF) protocol, which well resists possible brute-force attacks from the traditional deterministic CE. To ensure security and reliability of CE keys, Dekey distributes the CE keys to be shared across multiple servers for efficient CE key management via the ramp secret sharing scheme (RSSS) [154]. SecDep combines cross-user file-level and inside-user chunk-level deduplication, and employs difference CE variants among and inside users to minimize the computation overheads. In addition, SecDep uses file-level key to encrypt chunk-level keys so that the key space will not increase with the number of sharing users.

- Side channel attacks refer to the multiclient deduplication performed by a cloud storage server where an adversary at one client site uses deduplication as side channel attacks to reveal the private information about the contents of files/chunks of other clients (i.e., having identical and sensitive contents). Harnik *et al.* [147] propose a hybrid approach that sometimes artificially turns off cross-user deduplication to reduce the risk of data leakage. Heen *et al.* [157] propose a gateway-based deduplication approach that allows deduplication from gateways to servers rather than from clients, which resists side channel attacks from an adversary at the client side.
- Proofs of ownership (POW) lets a client efficiently prove to a server that he/she holds a file, which is used to prevent an attacker from gaining access to file contents of other users based on files' fingerprints. Halevi *et al.* [148] first introduce the notion of POW to allow a client to efficiently prove to a server that the client owns a file, by constructing error correction codes and Merkle trees (used for data authentication) [158] of the file. The variants of POW have been proposed subsequently to either reduce the overhead for generating POW [159] or further ensure security on POW [160], [161].

In addition, Li *et al.* build a multicloud storage system called CDStore [163] recently, to ensure both data security and reliability for deduplication. CDStore is different from the traditional CE approach: it proposes an augmented secret sharing scheme called convergent dispersal called CAONT-RS [164], which replaces original random information with deterministic cryptographic hash information and splits data into n secret shares. Thus, the identical data will generate the same secret shares, which enables deduplication on secret shares. CDstore also ensure the reliability by distributing the secret shares into multicloud and recovering the data by getting k secret shares ($k < n$ in CAONT-RS), but there is a tradeoff between the saved storage space by deduplication and increased storage costs by using CAONT-RS.

Existing approaches to secure data deduplication focus on addressing the three known security challenges in cross-user deduplication, especially in cloud storage. Shutting down cross-user deduplication would provide the highest security level to avoid privacy leakage among users [147], [161]. However, trading cross-user deduplication against data security remains a challenge facing cloud storage, which will be discussed as one of the future directions in Section IV-D.

H. Reliability

Data deduplication reduces the reliability of the storage systems because the loss of a few critical data chunks can lead to many referenced files being lost [165], [166]. In addition, reliability issues may further arise because deduplication reduces the redundancy upon which all existing reliability and fault-tolerance mechanisms rely. For example, data recovery in faulty storage devices is achieved by exploiting data redundancy introduced by adding copies (e.g., replication or parity in RAID), which fail to work if redundant data are entirely eliminated by deduplication.

In general, the state-of-the-art approaches can be broadly classified into two categories, deduplication then "RAID" and reference-count-based replication, as detailed below.

- Deduplication then "RAID" refers to directly coding unique chunks after deduplication using certain erasure codes. DDFS [13] uses software RAID-6 to store unique chunks to ensure a high-level data integrity and reliability. Hydrastor [59] places unique chunks into different resilience classes, each of which has a different level of reliability provided by users. In R-ADMAD [167], the variable-size chunks are first packed into bigger fixed-size objects (e.g., 8 MB), which are then erasure coded and distributed on multiple storage nodes in a redundancy group. In addition, replicating the unique chunks to another server is also a generally simple but efficient approach for ensuring storage reliability because the data has been deduplicated first [168].
- Reference-count-based replication provides different levels of reliability according to the reference counts of unique chunks. Bhagwat *et al.* [165] observe that deduplication alters the reliability of stored data due to the sharing of common chunks. They address this problem by choosing each chunk from a replication level that should

Table 9 Typical Application Scenarios of Data Deduplication and Their Known Examples

Scenarios	Benefits from Data Deduplication	Examples
Secondary storage	Save storage space, devices cost, time for backup/archive	Venti [11], DDFS [13], HydraStor [59]
Primary storage	Save storage space and eliminate duplicate disk I/Os	I/O Deduplication [179], iDedup [64], ZFS [73]
Cloud storage	Save storage space and reduce time of uploading data in WAN	Cumulus [180], SkyDrive [74], DropBox [74]
Virtual machine	Save main/primary storage space, save time for VM migration	VMware ESX [181], VMflock [15], DEDE [182]
Network	Reduce time of transmitting redundancy for WAN optimization	Spring et al. [183], SmartRE [184], EndRE [81]
SSD, multimedia, etc.	Extend lifetime of SSDs, eliminate visual redundancy, etc.	CAFTL [185], Nitro [186], ViDedup [175]

be logarithmic to the popularity of that chunk. Rozier *et al.* [169] design and implement a modeling framework to evaluate the reliability of a deduplication system with different hardware and software configurations. The reference-count-based replication approach is preferable for ensuring reliability in data deduplication systems.

In designing reliable deduplication, leveraging different levels of reliability according to the reference count of unique chunks helps to improve the entire system reliability in deduplication-based storage systems. However, there are always tradeoffs between ensuring reliability and space savings by deduplication. In addition, Li *et al.* [166] also suggest deduplication may improve reliability of storage systems if the traditional MTTDL metric is used to evaluate reliability. This is because deduplication reduces the number of required disk drives and hence the probability of seeing disk errors.

I. Other Interesting Issues

There are other interesting issues in deduplication, such as deduplication ratio estimation, file recipe compression, and video/image deduplication, etc., that are worthy of future research and development attention.

- The research on deduplication ratio estimation is motivated by the observation that the deduplication ratio may vary significantly according to the workload as well as the underlying deduplication techniques [170]–[173]. The accurate estimation of deduplication ratio in concrete data sets will allow the users to decide how many disks to buy, what techniques to use, etc. Harnik *et al.* [173] use a general framework of sampling and scanning for estimating deduplication ratio with low time and RAM overheads. Xie *et al.* [172] propose an adaptive technique to incrementally maintain an up-to-date estimate of deduplication ratio that takes into account any changes to the file system.
- File recipe compression is proposed by Meister *et al.* [174] to further reduce the space overhead for file recipes by recording the sequence of chunk fingerprints. They report that deduplicating a 1-PB backup data set with average chunk size of 8 kB, SHA-1 fingerprints, and deduplication factor of 30, will generate file recipes of about 2.4 TB, which can be efficiently reduced by up to 90% using techniques such as zero-chunk suppression and entropy coding.
- Video and image deduplication detects and eliminates redundancy among the very similar videos/images based on their visual contents, which is very different from the traditional deduplication approaches but is gaining increasing tractions recently in academia and industry. ViDedup [175] proposes a general framework for detecting very similar videos at an application-level point of view to eliminate redundancy. Perra *et al.* [176] leverage state-of-the-art video compression techniques to efficiently reuse image content among several very similar images, a technique referred to as image content deduplication, to support large-scale compression of similar images. By using semantic hashing and flat addressing, FAST [177] explores and exploits the semantic correlation of images to fast identify similar contents and support near-real-time image retrieval. Dewakar *et al.* [178] point out that the similar video files can significantly benefit from their proposed content-aware deduplication techniques, which improve storage efficiency by up to 45% in some use cases.

IV. RESOURCES AND PERSPECTIVES

In this section, we will first study the typical application scenarios of data deduplication, and how they incorporate and benefit from deduplication. Then we provide industry insights into how deduplication works in commercial storage systems. We also introduce the publicly available resources for deduplication research, including open-source projects, data sets, traces, etc. Finally, we conclude this section by identifying the open problems and research challenges facing the research community as possible future directions for data deduplication.

A. Application Scenarios of Data Deduplication

Data deduplication schemes have been widely applied in many scenarios of computer systems. Table 9 introduces some typical application scenarios of data deduplication and their beneficial features.

- Secondary storage. There is an abundance of duplicates in secondary storage systems, such as backup and archive storage [7], [11], [13], [58], [59], [187]. Based on this observation, data domain [13], a storage company and now a part of EMC, argues that "disk-based deduplication storage has emerged as the new-generation storage system for enterprise data protection to replace tape libraries." In fact, the use of data deduplication in such systems has been shown to achieve a data reduction factor of about 5 ~ 40 [7], [8], leading to significant savings in storage space and corresponding hardware costs. More recently, post-deduplication delta compression has been used to compress the nonduplicate but similar data chunks as a complementary approach to data deduplication. Such post-deduplication schemes are shown to achieve an extra data reduction factor of 2 ~ 5 on top of data deduplication but adding additional computation and I/O overheads [54], [55], [68].

- Primary storage. Recent studies reveal that data deduplication can achieve a data reduction factor of up to 40% ~ 60% in primary storage [5], especially for server file systems [5], [6], [179] (refer to Table 2). Primary storage data deduplication not only reduces the storage space requirement but also eliminates duplicate I/Os on the critical I/O path [64], [130], [179], which helps improve the disk I/O performance for primary storage. More recently, several open-source file systems for primary storage, such as ZFS [73], OpenDedupe [188] and Lessfs [189], have incorporated deduplication for better storage performance. Since deduplication incurs computing and indexing latency in the write path and disk fragmentation in the read path, ZFS and Lessfs provide deduplication as an optional function to users. Note that in comparison with secondary storage deduplication, primary storage deduplication is much different for three main reasons: I/O latency is much more sensitive to deduplication operations, read and delete operations occur much more frequently, and fewer duplicates exist in primary storage systems.
- Cloud storage. Cloud storage has emerged as an important storage platform for computer systems in recent years [74], [180]. Since the limited network bandwidth in the underlying wide area network (WAN) imposes the main performance bottleneck of cloud storage, data deduplication can help accelerate the data synchronization between client and cloud by identifying the unmodified data. In the meantime, data deduplication also helps reduce the storage overhead in the cloud side. Currently, DropBox, SkyDrive (now called OneDrive), Google Drive, etc., incorporate data deduplication to provide better cloud storage service [74], [190]. As discussed in Microsoft's study [5], while there exist a large number of duplicates across users, cross-user data deduplication can cause security problems [161] (see Section III-G). This remains an open problem, to be further discussed in Section IV-D.
- Virtual machines. A great deal of redundant data exists in virtual machines, either in the main memory [121], [191] or the external memory [192]. This is because the operating systems and application programs on homogeneous or heterogeneous virtual machines tend to generate duplicate data. Moreover, deduplication meets the design goal of virtualization in computer systems by saving storage space of both memory [121], [181], [191], [193] and disks [182], [192], [194], thus relieving the burden on storage devices. In addition, deduplication is employed to accelerate live migration of virtual machines in many state-of-the-art approaches by significantly reducing the amount of data migrated [15], [195], [196].
- Network environments. One of the initial purposes for using data deduplication is to save network bandwidth by avoiding transmitting redundant data, especially in wide area network (WAN) environments [89], [197]–[199] and avionics network environments [200]. Network deduplication, also called redundancy elimination, is slightly different from data deduplication in storage systems. Specifically, the granularity for network deduplication, i.e., the size of a data chunk, is often tens or hundreds of bytes, which is much finer than the kilobyte-scale (or even megabyte-scale) granularity in backup storage deduplication. Moreover, the objects for data deduplication in network environments are data streams or data packets, for which network deduplication often uses a weaker but faster hash for the fingerprinting algorithm to identify the data redundancy in a byte-by-byte manner. A study from Microsoft Research [89] shows that packet-level redundancy elimination techniques could achieve bandwidth savings of about 15% ~ 60% on 12 large network nodes. Deduplication for network transfer, unlike within a storage system, does not have the overheads of fragmented locality and garbage collection, since transferred data are typically reconstructed at the destination.
- Endurance-limited nonvolatile storage. Recently, deduplication has been used in emerging nonvolatile storage devices that have endurance constraints (i.e., write limit), such as SSD devices, to reduce the amount of write traffic to the devices and increase their effective logical capacity. CAFTL [185] and CA-SSD [201] employ deduplication to eliminate duplicate I/Os to extend SSD lifetime. I-CASH [126] uses delta compression to

eliminate both duplicate and similar data to enlarge the logical space of SSD caches. Nitro [186] implements deduplication and LZ compression for SSD-cache-based primary storage to decrease the cost of SSD caches and increase logical capacity.

- Other applications. Recently, deduplication among the very similar multimedia files is gaining increasing attention as discussed in Section III-I. Katiyar *et al.* [175] and Perra *et al.* [176] identify and reduce redundancy among the very similar videos/images based on their visual contents instead of binary contents. Dewakar *et al.* [178] suggest that deduplication of similar videos could improve storage efficiency by up to 45% in some use cases. Tang *et al.* [202] present UNIC, a system that securely deduplicates general computations, which significantly speeds up some applications by memorizing and reusing computation results.

In general, data deduplication has been most widely used in backup/archive storage systems and is starting to be adopted in primary storage. By means of eliminating duplicate data, deduplication provides many benefits not only for backup/archive storage systems but also other application scenarios, such as reducing duplicate I/Os for primary storage, avoiding transmitting duplicate data for network environments, extending lifetime of emerging nonvolatile storage devices, etc. With the explosive growth in the volume of digital data in the big data era, we believe that there will be more applications that stand to benefit from data deduplication that effectively and efficiently identifies and eliminates duplicate copies of redundant data to improve system performance.

B. Industry Perspective of Deduplication

Deduplication has been commercially available since 2004 in a dedicated backup appliance [168], and multiple vendors have created competing products [59], [203]–[205]. Since that time, deduplication has become a standard feature on numerous storage systems from static archival to high performance, primary storage. While deduplication has become widely implemented, we focus our discussion on backup storage since it has the longest history, and its evolution since 2004 provides insights into how the industry adapts to technology changes. Since the introduction of deduplicated backup storage in 2004, that market sector has grown to $3.1 billion in 2013 [206].

1) Shift From Tape to Disk: Early backups were written to tape systems that had high sequential read/write performance, but they also had other properties that complicated restores. First, a tape would have to be explicitly mounted before use; even though these mounts evolved from human-operator actions to automatic "jukebox" operations, each mount still suffered notable delays. Second, magnetic tape has very low random IOPS, due to its hardware characteristics. For these reasons, backups to tape have traditionally followed a model in which the entire contents of a system would be backed up at once (a "full" backup), and then the changes to the system since the last full backup would be saved on a regular basis. The latter are called "incremental" backups, and include each file that has been modified anywhere in the file.

Switching from tapes to HDDs offered numerous advantages to customers because of their performance differences, but hard drive storage alone was too expensive to be cost-competitive on a byte-for-byte basis. However, each time a full backup would be written to the system, it would contain numerous files that were unmodified in whole or in part. Deduplication, in combination with compression, became a promising approach to increasing effective capacity and lowering system cost. Also, the duplicate chunks tended to be written in consecutive order largely unchanged from backup to backup. This combination of high rates of duplicate content and consistent patterns were leveraged to create high performance, deduplicated backup storage.

Once the data was deduplicated on HDD, it could be processed in ways differently than when it is stored on tape. Data could be quickly replicated locally or offsite since deduplicated data are a small fraction of its original size. The accuracy of the data can be verified quickly and on a regular schedule as compared to tapes sent offsite, and potentially more versions can be kept on site since each new version requires little storage space relative to its original size.

The I/O characteristics of HDD are different from those of tape, so companies and customers started to use the systems in new ways. Instead of sending full backups that consist mostly of duplicate content, it became efficient for primary storage to send the changed data, and the deduplicated system could reconstruct a full backup. This process transfers incremental backups (perhaps daily, hourly, or more frequently) to the backup server where a full backup is synthesized. Only modified files or changed blocks need to be transferred, and then a full backup is synthesized by creating a recipe referencing the new data and previously written duplicate content. The on-disk representation on the deduplicated system is largely the same as writing full backups since only unique chunks are stored in either backup style, along with a recipe to represent the file. The I/O and CPU loads are decreased on the primary storage systems since less data are read, and since less data are transferred, network traffic is also less.

There is an important side-effect of the shift from 1) writing and then deduplicating full backups to

2) writing incremental backups forever and synthesizing a full backup simply by writing a new file recipe referring to all the chunks contained in the backup. When a new full backup is written, a deduplicating system can choose to write duplicates from the new backup to disk, then garbage-collect the older copy of a duplicate chunk. By writing sequential chunks of the new backup together on disk, a later restore of the backup will experience fewer random I/Os and have better read performance. As discussed in Section III-E, there are various techniques to decide when to write new duplicates. If the workload consists of only incremental updates that result in more random I/Os during restore, however, the system must be more proactive in reorganizing the file system to provide acceptable read performance.

2) Bridging the Primary Storage and Backup Storage Divide: Primary storage and backup storage used to be entirely distinct systems, with a media server running backup software acting as middleware to coordinate backup transfers. To effectively synthesize full backups from incrementally changed data, a new interface is needed that is deduplication aware. This has motivated closer integration between backup and primary storage systems. It is becoming more common for backup agents to run directly on primary storage and direct backups without transferring data through a separate media server (such as DDBOOST [207], RMAN [208], NetBackup [209], and Avamar [210]. In some cases, this has increased CPU loads on the primary storage system, while allowing a backup system to handle more concurrent writes. In other models, backup storage may appear as a mountable device or file system to primary storage. Deduplicated backup storage, instead of being limited to tape replacement, has become another tier in the storage hierarchy. While deduplicated backup storage remains slower than primary storage, demands for more performance continue to grow.

Customers would like their backups to be more useful than static content that is only read back in the rare case of a data loss. While the backups should be write-protected, there are several circumstances under which customers might like to read the data. As an example, in the case of data loss, a customer may like to browse their files on backup storage and view several files to determine which version is most useful. As a second example, data analytics could be performed on a database stored on the backup server when it is impractical to perform the same analysis on an active version that is concurrently serving client requests. In certain cases, a customer's primary storage system may be inoperable for an extended period of time. While new primary storage hardware is provisioned and configured, a customer may want to run an application from backup storage. Customers would typically copy a backup, mark it as writeable, and then actively use that version. While the backup system may not have the performance of primary storage, it may be fast enough to serve limited workloads. As an example, a VM could use the backup storage as a data store for the VM image. A second example is that a customer may like to test a software upgrade for incompatibilities on a system running from backup storage before upgrading the primary version.

In a little over a decade, deduplicated backup storage has evolved from tape replacement to a tier of storage systems that may serve as limited primary storage. While we have provided a brief summary of this history from a use-case perspective, numerous technical improvements were necessary to enable these changes, including performance and management improvements discussed in this survey.

C. Deduplication in the Open-Source Community

In this section, we will list some publicly available open-source projects, data sets, and traces for data deduplication, which are a useful and valuable resources for the research community.

- Open-source projects. One of the most well-known open-source projects is LBFS,[1] which implements CDC-based deduplication in a network file system [14]. Opendedup[2] is a deduplication-based filesystem designed to provide inline deduplication and flexibility for applications on the block devices [188]. FS-C[3] is a tool developed by Mainz University [57], [70], [174] that allows researchers to analyze the internal and temporal redundancy of file system directories. Destor[4] is a platform for deduplication evaluation on fingerprint indexing and restoring, which has been used in several research papers [103], [129], [211]. In addition, Dmdedup[5] is a practical primary storage deduplication platform that runs deduplication at the block layer [143] and Lessfs[6] is an inline data deduplicating filesystem for Linux [189].
- Data Sets, data traces, and benchmarks. The archives of large open-source projects (i.e., the tarred source code), such as Linux,[7] GCC,[8] etc., have been evaluated for deduplication in many research papers [55], [60], [75], [129]. Virtual machine images,[9] such as Ubuntu, Fedora, etc.,

[1]http://pdos.csail.mit.edu/lbfs/index.html
[2]https://code.google.com/p/opendedup/
[3]https://code.google.com/p/fs-c/
[4]https://github.com/fomy/destor
[5]http://git.fsl.cs.sunysb.edu/linux-dmdedup.git/
[6]http://www.lessfs.com/wordpress/
[7]ftp://ftp.kernel.org/
[8]http://ftp.gnu.org/gnu/
[9]http://www.thoughtpolice.co.uk/vmware

have also been evaluated [68], [132], [134], [192]. The SyLab of FIU [179] has published I/O traces[10] (including the hash value of the content for each I/O) for a three-week interval. It has a total size of about 1TB and has mainly been used for evaluating several primary deduplication schemes [130], [201], [212]. The File systems and Storage Lab (FSL) at Stony Brook University has published their traces collected from several graduate students' home directories and MacOS Snapshots with a total size of several terabytes,[11] and developed a tool to regenerate data sets according to the deduplication traces [213]. Additionally, there are several benchmarks for generating workloads of users' files [190], [214] and compressible contents [215] that can be used for deduplication analysis.

D. Open Problems and Future Directions

Although data deduplication has been studied for more than ten years, many open problems and challenges remain to be addressed, particularly as the size of digital data continues to grow exponentially and the need for long-term storage management becomes increasingly urgent. Based on the studies discussed in Sections III and IV, in what follows we outline the open problems and possible future directions for deduplication.

- To compare by hash, or not to compare? The mainstream data deduplication systems, such as LBFS [14], Venti [11], and DDFS [13], use SHA1-based fingerprints to represent and identify data chunks, which is justified by the argument that the probability of a cryptographic hash collision between any two randomly generated chunks is many orders of magnitude smaller than that of many hardware errors [11]. Henson [65] points out that the hash-based comparison approach is not risk-free and the errors and bugs caused by hash collisions for deduplication are difficult to discover and repair in a long-lived storage systems. Black *et al.* [66] argue that it would cost about \$80 000 000 and two years to find a collision of SHA1 in 2006, and thus suggest that compare-by-hash is completely reasonable. Thus, some systems resolve this problem by checking the content [60], [187] after a compare-by-hash. Some other systems choose to use stronger cryptographic hash, such as SHA-256 [73], [74], for fingerprinting.
- Standard for deduplicating transport. With the proliferation of commercially available deduplicated storage systems, a potential next stage to improve the efficiency of storage is to create a transport protocol that is deduplication aware. We briefly summarize several currently available APIs. Symantec OpenStorage [216] was introduced circa 2007 to provide an API for integration of disk-based backup platforms with Symantec NetBackup [209]. Data Domain's DDBOOST API, introduced in 2010, similarly integrates numerous backup software systems with the Data Domain appliance [207]. By providing a plug-in where the backup software can perform chunking and fingerprinting and allowing the backup appliance to determine which chunks are duplicates, DDBOOST distributes the processing and, when duplication is high, reduces network load as well. Can this technique for transferring only unique data achieve broader applicability? We already see recipe-based content-distribution systems such as Bittorrent [217]. It is possible that with increasing infrastructure available in the form of deduplicating, content-addressable file systems, network transfers beyond niche applications like Bittorrent will adapt to provide recipes for data rather than the data itself. With such a protocol in place, it will become possible to transfer data from one deduplicated product to a different deduplicated product without inflating the data during transfer.
- Restore and garbage collection after deduplication. Deduplication causes sequential or contiguous data to be fragmented, a serious issue facing the post-deduplication restore and garbage collection processes in both primary and secondary storage due to the sharing of data chunks. Existing solutions try to eliminate fragmentation by not deduplicating (i.e., rewriting) some of the duplicate but fragmented chunks [23], [129], which sacrifices deduplication efficiency and does not fully solve the data restore and garbage collection problems for long-lived primary or secondary storage systems. In addition, incorporating data deduplication with (post-deduplication) delta compression and traditional compression techniques increases the complexity of these problems; this is because recovering a delta compressed chunk requires reading the base and delta chunks, which is two disk I/Os instead of one.
- Secure deduplication is one of the most important concerns for users, especially in the cloud environment. Cross-user deduplication may leak user's confidential information [147] or lead to some possible new attacks by obtaining fingerprints that represent data contents [148]. To the

[10]http://sylab-srv.cs.fiu.edu/doku.php$?$id=projects:iodedup:start
[11]http://tracer.filesystems.org/

Table 10 List of Acronyms and Terms

Terms	Explanations
Bloom filter	A memory-efficient indexing data structure
CDC	Content-Defined Chunking: Dividing data into smaller chunks based on characteristics of the data [14]
CE	Convergent Encryption used for secure deduplication [56]
Chunk	The minimum storage unit for deduplication
Container	A fixed-size storage unit that stores deduplicated chunks, usually size of several megabytes [13]
DDFS	Data Domain File System [13]
Fingerprint	Cryptographically secure hash digest of a chunk or a file, e.g., SHA1
GC	Garbage Collection: reclaiming free space after delete operations
HDD	Hard Disk Drive
LBFS	Low-Bandwidth network File System [14]
LZ	Lempel-Ziv: the universal compression algorithm [16, 17]
Rabin	Rabin hash algorithm used for CDC [77]
SSD	Solid State Disk
Venti	A deduplication based archival storage system [11]

best of our knowledge, while most existing studies focus on security of the data deduplication technique, there are no published studies discussing the security concerns when data compression and deduplication are jointly used. In fact, data compression is implemented after data deduplication in current backup storage systems [7], [13], while the classic convergent encryption approaches [150] first encrypt data and then compute fingerprints on cipher text for deduplication. This is contradictory since the encryption makes the data randomized and thus incompressible. In addition, assured deletion [144], [218] is also a promising security topic in deduplication based cloud environment.

- Reliability is another important problem for long-lived deduplication-based storage systems. Existing approaches that replicate unique chunks with a high reference count are a good solution to the reliability problem for deduplication. It would be interesting to see how such approaches may be integrated with the data restore and garbage collection processes for better storage performance after deduplication. Additionally, the long-term reliability analysis of deduplicated data, which is still lacking, is very important in the long-term primary or secondary storage systems.
- Primary storage deduplication. Deduplication is increasingly popular in primary storage systems with the rapid growth of application data. Employing deduplication to improve I/O performance [130] will play a more important role in high-performance storage systems than saving storage space. Read and delete operations frequently occur in primary storage systems. Addressing these problems with variable primary storage workloads in systems with different storage devices, such as, disks, DRAM memory, non-volatile memory devices, will be an interesting future research direction.
- Emerging applications. As introduced in Section IV, deduplication can benefit applications besides disk storage such as employing deduplication to extend the lifetime of SSDs and PCMs [185], [201] and eliminating visual redundancy for images and videos [175], [176]. We believe that there will be more applications for deduplication, such as storage systems for tapes [219] or shingled disks, since it will help reduce the growing redundant data in large-scale storage systems.

V. SUMMARY

Data deduplication is a scalable and efficient redundant data reduction technique for large-scale storage systems, which addresses the challenges imposed by the explosive growth in demand for data storage capacity. In this comprehensive survey study, we review the background of data deduplication and the differences between data deduplication and traditional data compression. We also comprehensively study the state-of-the-art work on data deduplication, classifying them into six general categories based on data deduplication workflow, and then create a taxonomy for each category, which provides insights into the pros and cons of existing solutions. Applications that use data deduplication are also examined in depth in this paper. Further, publicly available open-source projects, data sets, and traces are summarized for the convenience of the research community to further research and development. Finally, the open problems and research challenges are outlined according to our comprehensive studies on existing works and applications of deduplication techniques.

APPENDIX

Some frequently appearing acronyms and terminologies appear in Table 10. ■

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and feedback. The authors would also like to thank S. Mandal, G. Kuenning, V. Tarasov, and E. Zadok for valuable discussions about deduplicated storage literature.

REFERENCES

[1] "The data deluge," *The Economist*, Feb. 25, 2010. [Online]. Available: http://www.economist.com/node/15579717

[2] IDC, "The 2011 digital universe study," Tech. Rep., Jun. 2010. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf

[3] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, biggest growth in the far east," *IDC iView: IDC Analyze the Future*, 2012. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf

[4] "The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things," EMC Digital Universe with Research & Analysis by IDC, Apr. 2014. [Online]. Available: http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[5] D. Meyer and W. Bolosky, "A study of practical deduplication," in *Proc. USENIX Conf. File Storage Technol.*, San Jose, CA, USA, Feb. 2011, pp. 229–241.

[6] A. El-Shimi, R. Kalach, and A. Kumar *et al.*, "Primary data deduplication-large scale study and system design," in *Proc. Conf. USENIX Annu. Tech. Conf.*, Jun. 2012, pp. 1–12.

[7] G. Wallace *et al.*, "Characteristics of backup workloads in production systems," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 1–14.

[8] P. Shilane *et al.*, "WAN optimized replication of backup datasets using stream-informed delta compression," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 1–14.

[9] L. DuBois, M. Amaldas, and E. Sheppard, "Key considerations as deduplication evolves into primary storage," White Paper 223310, Mar. 2011. [Online]. Available: http://www.bedrock-tech.com/wp-content/uploads/2010/05/wp_key-considerations.pdf

[10] W. J. Bolosky *et al.*, "Single instance storage in Windows 2000," in *Proc. 4th USENIX Windows Syst. Symp.*, Aug. 2000, pp. 13–24.

[11] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proc. USENIX Conf. File Storage Technol.*, Jan. 2002, pp. 1–13.

[12] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data," in *Proc. USENIX Annu. Tech. Conf. Gen. Track*, Jun. 2004, pp. 73–86.

[13] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. 6th USENIX Conf. File Storage Technol.*, Feb. 2008, vol. 8, pp. 1–14.

[14] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Proc. ACM Symp. Oper. Syst. Principles*, Oct. 2001, pp. 1–14.

[15] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "VMFlock: Virtual machine co-migration for the cloud," in *Proc. 20th Int. Symp. High Performance Distrib. Comput.*, Jun. 2011, pp. 159–170.

[16] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, 1977.

[17] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[18] M. Oberhumer, "LZO real-time data compression library," User manual for LZO version 0.28, Feb. 1997. [Online]. Available: http://www.infosys.tuwien.ac.at/Staff/lux/marco/lzo.html

[19] M. R. Nelson, "LZW data compression," *Dr. Dobb's J.*, vol. 14, no. 10, pp. 29–36, 1989.

[20] L. P. Deutsch, "DEFLATE compressed data format specification version 1.3," RFC Editor, 1996. [Online]. Available: http://tools.ietf.org/html/rfc1951

[21] X. Lin, G. Lu, F. Douglis, P. Shilane, and G. Wallace, "Migratory compression: Coarse-grained data reordering to improve compressibility," in *Proc. 12th USENIX Conf. File Storage Technol.*, Feb. 2014, pp. 257–271.

[22] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[23] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 183–197.

[24] D. Hamilton, "Deduplication-methods for achieving data efficiency," 2008. [Online]. Available: http://www.snia.org/sites/default/education/tutorials/2008/spring/data-management/Hamilton-D_Deduplication_Methods_Data_Efficiency.pdf

[25] T. Riveria, "Understanding data deduplication," 2009. [Online]. Available: http://www.snia.org/sites/default/education/tutorials/2009/fall/data/ThomasRivera_UnderstandingDeduplication_A_Tutorial_Understanding_Dedupe_9-15-09.pdf

[26] A. Brinkmann, "Data deduplication—Tutorial," 2011. [Online]. Available: https://pc2.uni-paderborn.de/fileadmin/pc2/media/staffweb/Andre_Brinkmann/Courses/Speichersysteme_SS_2011/Deduplication_-_Eurosys_Tutorial.pdf

[27] T. Riveria and G. Nagle, "Advanced dedupe concepts," 2011. [Online]. Available: http://www.snia.org/sites/default/education/tutorials/2011/fall/DataProtectionManagement/ThomasRiveria_Advanced_Dedupe_Concepts_FINAL.pdf

[28] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, "Demystifying data deduplication," in *Proc. ACM/IFIP/USENIX Middleware Conf. Companion*, Dec. 2008, pp. 12–17.

[29] A. F. Banu and C. Chandrasekar, "A survey on deduplication methods," *Int. J. Comput. Trends Technol.*, vol. 3, no. 3, pp. 364–368, 2012.

[30] Q. He, Z. Li, and X. Zhang, "Data deduplication techniques," in *Proc. Int. Conf. Future Inf. Technol. Manage. Eng.*, Aug. 2010, vol. 1, pp. 430–433.

[31] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 11, 2014.

[32] P. Neelaveni and M. Vijayalakshmi, "A survey on deduplication in cloud storage," *Asian J. Inf. Technol.*, vol. 13, no. 6, pp. 320–330, 2014.

[33] P. Christen, *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, Duplicate Detection*. New York, NY, USA: Springer-Verlag, 2012.

[34] F. Naumann and M. Herschel, "An introduction to duplicate detection," *Synthesis Lectures Data Manage.*, vol. 2, no. 1, pp. 1–87, 2010.

[35] J. A. Storer, *Data Compression: Methods and Theory*. New York, NY, USA: Computer Science Press, 1988.

[36] Theory of Data Compression. [Online]. Available: http://www.data-compression.com/

[37] J. Gailly and M. Adler, "The gzip compressor," 1991. [Online]. Available: http://www.gzip.org/

[38] M. W. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards, Practice*. New York, NY, USA: Springer-Verlag, 2002, vol. 1.

[39] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 5, no. 1, pp. 3–55, 2001.

[40] D. A. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[41] G. G. Langdon, Jr., "An introduction to arithmetic coding," *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 135–149, 1984.

[42] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[43] Deflate compression. [Online]. Available: http://zh.wikipedia.org/zh-cn/DEFLATE, 1991.

[44] 7zip. [Online]. Available: http://www.7-zip.org/

[45] J. Gilchrist, "Parallel data compression with bzip2," in *Proc. 16th Int. Conf. Parallel Distrib. Comput. Syst.*, Jul. 2004, vol. 16, pp. 559–564.

[46] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," Digital SRC Res. Rep., 1994.

[47] W. F. Tichy, "RCSła system for version control," *Softw., Practice Exp.*, vol. 15, no. 7, pp. 637–654, 1985.

[48] J. J. Hunt, K.-P. Vo, and W. F. Tichy, "Delta algorithms: An empirical analysis," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, no. 2, pp. 192–214, 1998.

[49] J. MacDonald, "File system support for delta compression," M.S. thesis, Dept. Electr. Eng. Comput. Sci., Univ. California at Berkeley, Berkeley, CA, USA, 2000.

[50] A. Tridgell and P. Mackerras, "The rsync algorithm," 1996.

[51] N. Jain, M. Dahlin, and R. Tewari, "TAPER: Tiered approach for eliminating redundancy in replica synchronization," in *Proc. USENIX Conf. File Storage Technol.*, Mar. 2005, pp. 281–294.

[52] T. Suel and N. Memon, "Algorithms for delta compression and remote file synchronization," Lossless Compression Handbook, 2002.

[53] R. C. Burns and D. D. Long, "Efficient distributed backup with delta compression," in *Proc. 5th Workshop I/O Parallel Distrib. Syst.*, Nov. 1997, pp. 27–36.

[54] P. Shilane, G. Wallace, M. Huang, and W. Hsu, "Delta compressed and deduplicated storage using stream-informed locality," in *Proc. 4th USENIX Conf. Hot Topics Storage File Syst.*, Jun. 2012, pp. 201–214.

[55] W. Xia, H. Jiang, D. Feng, and L. Tian, "Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets," in *Proc. IEEE Data Compression Conf.*, Mar. 2014, pp. 203–212.

[56] W. J. Bolosky, J. R. Douceur, and D. Ely *et al.*, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," *ACM SIGMETRICS Performance Eval. Rev.*, vol. 28, no. 1, pp. 34–43, 2000.

[57] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proc. SYSTOR, Israeli Exp. Syst. Conf.*, May 2009, pp. 1–12.

[58] M. Lillibridge *et al.*, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. 7th USENIX Conf. File Storage Technol.*, Feb. 2009, vol. 9, pp. 111–123.

[59] C. Dubnicki *et al.*, "HYDRAstor: A scalable secondary storage," in *Proc. USENIX Conf. File Storage Technol.*, Feb. 2009, vol. 9, pp. 197–210.

[60] D. Bhagwat *et al.*, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Proc. IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Sep. 2009, pp. 1–9.

[61] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: Speeding up inline storage deduplication using flash memory," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2010, pp. 1–14.

[62] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2011, pp. 1–14.

[63] 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 *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2011, pp. 285–298.

[64] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 24–37.

[65] V. Henson, "An analysis of compare-by-hash," in *Proc. 9th Workshop Hot Topics Oper. Syst.*, May 2003, pp. 13–18.

[66] J. Black, "Compare-by-hash: A reasoned analysis," in *Proc. USENIX Annu. Tech. Conf. Gen. Track*, May 2006, pp. 85–90.

[67] Y. Xing, Z. Li, and Y. Dai, "PeerDedupe: Insights into the peer-assisted sampling deduplication," in *Proc. IEEE 10th Int. Conf. Peer-to-Peer Comput.*, Aug. 2010, pp. 1–10.

[68] W. Xia *et al.*, "Ddelta: A deduplication-inspired fast delta compression approach," *Performance Eval.*, vol. 79, pp. 258–272, 2014.

[69] D. Eastlake and P. Jones, "US secure hash algorithm 1 (SHA1)," 2001.

[70] D. Meister *et al.*, "A study on data deduplication in HPC storage systems," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, Jun. 2012, pp. 1–11.

[71] C. Riggle and S. G. McCarthy, "Design of error correction systems for disk drives," *IEEE Trans. Magn.*, vol. 34, no. 4, pp. 2362–2371, 1998.

[72] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *Proc. 5th USENIX Conf. File Storage Technol.*, Feb. 2007, vol. 7, pp. 1–16.

[73] Zfs. [Online]. Available: http://en.wikipedia.org/wiki/ZFS

[74] I. Drago *et al.*, "Inside Dropbox: Understanding personal cloud storage services," in *Proc. ACM Conf. Internet Meas. Conf.*, Nov. 2012, pp. 481–494.

[75] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *Proc. 7th USENIX Conf. File Storage Technol.*, Feb. 2010, pp. 1–14.

[76] W. Xia *et al.*, "P-dedupe: Exploiting parallelism in data deduplication system," in *Proc. 7th Int. Conf. Netw. Architect. Storage*, Jun. 2012, pp. 338–347.

[77] M. O. Rabin, "Fingerprinting by random polynomials," Cntr. Res. Comput. Tech., Aiken Comput. Lab., 1981.

[78] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," Hewlett Packard Lab., Palo Alto, CA, USA, Tech. Rep. HPL-2005-30(R.1), 2005.

[79] D. Bobbarjung, C. Dubnicki, and S. Jagannathan, "Fingerdiff: Improved duplicate elimination in storage systems," in *Proc. Mass Storage Syst. Technol.*, May 2006, pp. 1–5.

[80] D. Teodosiu, N. Bjorner, Y. Gurevich, M. Manasse, and J. Porkka, "Optimizing file replication over limited bandwidth networks using remote differential compression," Microsoft Research TR-2006-157, 2006.

[81] B. Aggarwal *et al.*, "EndRE: An end-system redundancy elimination service for enterprises," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, Apr. 2010, pp. 14–28.

[82] Y. Zhang *et al.*, "AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1–9.

[83] C. Yu, C. Zhang, Y. Mao, and F. Li, "Leap-based content defined chunking—Theory and implementation," in *Proc. 31th Symp. Mass Storage Syst. Technol.*, Jun. 2015, pp. 1–12.

[84] B. Romański, Ł. Heldt, W. Kilian, K. Lichota, and C. Dubnicki, "Anchor-driven subchunk deduplication," in *Proc. 4th Annu. Int. Syst. Storage Conf.*, May 2011, pp. 1–13.

[85] G. Lu, Y. Jin, and D. H. Du, "Frequency based chunking for data de-duplication," in *Proc. IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Aug. 2010, pp. 287–296.

[86] B. Zhou and J. Wen, "Hysteresis re-chunking based metadata harnessing deduplication of disk images," in *Proc. 42nd Int. Conf. Parallel Process.*, Oct. 2013, pp. 389–398.

[87] A. Broder, "Some applications of Rabin's fingerprinting method," *Sequences II: Methods Commun. Security Comput. Sci.*, pp. 1–10, 1993.

[88] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao, "QuickSync: Improving synchronization efficiency for mobile cloud storage services," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, Sep. 2015, pp. 592–603.

[89] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: Findings and implications," in *Proc. 11th Int. Joint Conf. Meas. Model. Comput. Syst.*, Jun. 2009, pp. 37–48.

[90] N. Bjørner, A. Blass, and Y. Gurevich, "Content-dependent chunking for differential compression, the local maximum approach," *J. Comput. Syst. Sci.*, vol. 76, no. 3, pp. 154–203, 2010.

[91] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems," *ACM Trans. Storage*, vol. 2, no. 4, pp. 424–448, 2006.

[92] J. Min, D. Yoon, and Y. Won, "Efficient deduplication techniques for modern backup operation," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 824–840, 2011.

[93] C. Liu *et al.*, "A novel optimization method to improve de-duplication storage system performance," in *Proc. 15th Int. Conf. Parallel Distrib. Syst.*, Dec. 2009, pp. 228–235.

[94] J. Ma, B. Zhao, G. Wang, and J. Liu, "Adaptive pipeline for deduplication," in *Proc. 28th IEEE Symp. Mass Storage Syst. Technol.*, Apr. 2012, pp. 1–6.

[95] S. Al-Kiswany *et al.*, "Storegpu: Exploiting graphics processing units to accelerate distributed storage systems," in *Proc. 17th Int. Symp. High Performance Distrib. Comput.*, Jun. 2008, pp. 165–174.

[96] P. Bhatotia, R. Rodrigues, and A. Verma, "Shredder: GPU-accelerated incremental storage and computation," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 1–15.

[97] C. Kim, K.-W. Park, and K. H. Park, "GHOST: GPGPU-offloaded high performance storage I/O deduplication for primary storage system," in *Proc. Int. Workshop Programm. Models Appl. Multicores Manycores*, 2012, pp. 17–26.

[98] X. Lin *et al.*, "Metadata considered harmful…to deduplication," in *Proc. 7th USENIX Workshop Hot Topics Storage File Syst.*, Jul. 2015.

[99] J. Bowling, "Opendedup: Open-source deduplication put to the test," *Linux J.*, vol. 2013, no. 228, p. 2, 2013.

[100] W. Xia, H. Jiang, D. Feng, and L. Tian, "Accelerating data deduplication by exploiting pipelining and parallelism with

multicore or manycore processors," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 1–2.

[101] A. Gharaibeh, S. Al-Kiswany, and S. Gopalakrishnan *et al.*, "A GPU accelerated storage system," in *Proc. 19th ACM Int. Symp. High Performance Distrib. Comput.*, Jun. 2010, pp. 167–178.

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

[103] M. Fu *et al.*, "Design tradeoffs for data deduplication performance in backup workloads," in *Proc. 13th USENIX Conf. File Storage Technol.*, Feb. 2015, pp. 331–344.

[104] J. Wei *et al.*, "MAD2: A scalable high-throughput exact deduplication approach for network backup services," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, May 2010, pp. 1–14.

[105] Y. Tan *et al.*, "SAM: A semantic-aware multi-tiered source de-duplication framework for cloud backup," in *Proc. 39th Int. Conf. Parallel Process.*, Sep. 2010, pp. 614–623.

[106] D. Meister, J. Kaiser, and A. Brinkmann, "Block locality caching for data deduplication," in *Proc. 6th Int. Syst. Storage Conf.*, Jun. 2013, pp. 1–12.

[107] L. Aronovich *et al.*, "The design of a similarity based deduplication system," in *Proc. SYSTOR, Israeli Exp. Syst. Conf.*, May 2009, pp. 1–12.

[108] D. Meister and A. Brinkmann, "dedupv1: Improving deduplication throughput using solid state drives (SSD)," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, May 2010, pp. 1–6.

[109] G. Lu, Y. J. Nam, and D. H. Du, "BloomStore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, Apr. 2012, pp. 1–11.

[110] T. Yang *et al.*, "DEBAR: A scalable high-performance de-duplication storage system for backup and archiving," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2010, pp. 1–12.

[111] W. Dong *et al.*, "Tradeoffs in scalable data routing for deduplication clusters," in *Proc. 9th USENIX Conf. File Storage Technol.*, Feb. 2011, pp. 15–29.

[112] Y. Fu, H. Jiang, and N. Xiao, "A scalable inline cluster deduplication framework for big data protection," in *Proc. ACM/IFIP/USENIX Middleware Conf.*, Dec. 2012, pp. 354–373.

[113] J. Kaiser, D. Meister, A. Brinkmann, and S. Effert, "Design of an exact data deduplication cluster," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, Apr. 2012, pp. 1–12.

[114] D. Frey, A.-M. Kermarrec, and K. Kloudas, "Probabilistic deduplication for cluster-based storage systems," in *Proc. 3rd ACM Symp. Cloud Comput.*, Oct. 2012, pp. 1–12.

[115] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compress. Complexity Sequences*, Jun. 1997, pp. 21–29.

[116] B. Debnath, S. Sengupta, and J. Li, "FlashStore: High throughput persistent key-value store," *VLDB Endowment*, vol. 3, no. 1/2, pp. 1414–1425, 2010.

[117] B. Debnath, S. Sengupta, and J. Li, "SkimpyStash: RAM space skimpy key-value store on flash-based storage," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 25–36.

[118] R. Pagh and F. F. Rodler, "Cuckoo hashing," *J. Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.

[119] A. Broder, "Identifying and filtering near-duplicate documents," in *Proc. Combinatorial Pattern Matching*, Jun. 2000, pp. 1–10.

[120] P. Kulkarni, F. Douglis, J. D. LaVoie, and J. M. Tracey, "Redundancy elimination within large collections of files," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2012, pp. 1–14.

[121] D. Gupta *et al.*, "Difference Engine: Harnessing memory redundancy in virtual machines," in *Proc. 8th Symp. Oper. Syst. Design Implement.*, Dec. 2008, pp. 309–322.

[122] D. Trendafilov, N. Memon, and T. Suel, "Zdelta: An efficient delta compression tool," Dept. Comput. Inf. Sci., Polytechnic Univ., Tech. Rep., 2002.

[123] U. Manber, "Finding similar files in a large file system," in *Proc. USENIX Winter*, Jan. 1994, vol. 94, pp. 1–10.

[124] F. Douglis and A. Iyengar, "Application-specific delta-encoding via resemblance detection," in *Proc. USENIX Annu. Tech. Conf. Gen. Track*, Jun. 2003, pp. 113–126.

[125] L. L. You, K. T. Pollack, and D. D. Long, "Deep Store: An archival storage system architecture," in *Proc. 21st Int. Conf. Data Eng.*, Apr. 2005, pp. 804–815.

[126] Q. Yang and J. Ren, "I-CASH: Intelligently coupled array of SSD and HDD," in *Proc. 17th IEEE Int. Symp. High Performance Comput. Architect.*, Feb. 2011, pp. 278–289.

[127] W. Xia, H. Jiang, D. Feng, and L. Tian, "DARE: A deduplication-aware resemblance detection and elimination scheme for data reduction with low overheads," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1–14, 2016.

[128] Y. J. Nam, D. Park, and D. H. Du, "Assuring demanded read performance of data deduplication storage with backup datasets," in *Proc. IEEE 20th Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Aug. 2012, pp. 201–208.

[129] M. Fu *et al.*, "Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2014, pp. 181–192.

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

[131] M. Kaczmarczyk, M. Barczynski, W. Kilian, and C. Dubnicki, "Reducing impact of data fragmentation caused by in-line deduplication," in *Proc. 5th Annu. Int. Syst. Storage Conf.*, Jun. 2012, pp. 1–12.

[132] C.-H. Ng and P. P. Lee, "RevDedup: A reverse deduplication storage system optimized for reads to latest backups," in *Proc. 4th Asia-Pacific Workshop Syst.*, 2013, p. 15.

[133] Y. Tan *et al.*, "CABdedupe: A causality-based deduplication performance booster for cloud backup services," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2011, pp. 1266–1277.

[134] B. Mao *et al.*, "SAR: SSD assisted restore optimization for deduplication-based storage systems in the cloud," in *Proc. IEEE 7th Int. Conf. Netw. Architect. Storage*, Jun. 2012, pp. 328–337.

[135] R. Lai *et al.*, "A near-exact defragmentation scheme to improve restore performance for cloud backup systems," in *Proc. 14th Int. Conf. Algorithms Architect. Parallel Process.*, 2014, pp. 457–471.

[136] Y. Nam, G. Lu, N. Park, W. Xiao, and D. H. Du, "Chunk fragmentation level: An effective indicator for read performance degradation in deduplication storage," in *Proc. IEEE 13th Int. Conf. High Performance Comput. Commun.*, Sep. 2011, pp. 581–586.

[137] Y. Li, M. Xu, C.-H. Ng, and P. P. Lee, "Efficient hybrid inline and out-of-line deduplication for backup storage," *ACM Trans. Storage*, vol. 10, no. 2, pp. 2–21, 2014.

[138] M. Fu *et al.*, "Reducing fragmentation for in-line deduplication backup storage via exploiting backup history and cache knowledge," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 3, pp. 855–868, 2016.

[139] L. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.

[140] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian, "Read-performance optimization for deduplication-based storage systems in the cloud," *ACM Trans. Storage*, vol. 10, no. 2, p. 6, 2014.

[141] D. N. Simha, M. Lu, and T. Chiueh, "A scalable deduplication and garbage collection engine for incremental backup," in *Proc. 6th Int. Syst. Storage Conf.*, Jun. 2013, pp. 1–12.

[142] P. Strzelczak *et al.*, "Concurrent deletion in a distributed content-addressable storage system with global deduplication," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 161–174.

[143] V. Tarasov *et al.*, "Dmdedup: Device mapper target for data deduplication," in *Proc. Ottawa Linux Symp.*, Ottawa, ON Canada., Jul. 2014, pp. 83–87.

[144] F. C. Botelho, P. Shilane, N. Garg, and W. Hsu, "Memory efficient sanitization of a deduplicated storage system," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 81–94.

[145] F. C. Botelho, A. Lacerda, G. V. Menezes, and N. Ziviani, "Minimal perfect hashing: A competitive method for indexing internal memory," *Inf. Sci.*, vol. 181, no. 13, pp. 2608–2625, 2011.

[146] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, "Dark clouds on the horizon: Using cloud storage as attack vector and online slack space," in *Proc. 20th USENIX Security Symp.*, Aug. 2011, pp. 1–11.

[147] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security Privacy*, vol. 8, no. 6, pp. 40–47, 2010.

[148] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. 18th ACM Conf. Comput. Commun. Security*, Oct. 2011, pp. 491–500.

[149] T. Dilatush and D. L. Whiting, "System for backing up files from disk volumes on multiple nodes of a computer network," U.S. Patent 5 778 395, Jul. 7, 1998.

[150] J. R. Douceur *et al.*, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, Jul. 2002, pp. 617–624.

[151] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 285–298, 2002.

[152] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication," in *Proc. 4th ACM Int. Workshop Storage Security Survivability*, Oct. 2008, pp. 1–10.

[153] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," in *Proc. 23th Int. Conf. Large Installation Syst. Admin., Strategies Tools Tech.*, Dec. 2010, pp. 195–206.

[154] J. Li *et al.*, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1–11, 2013.

[155] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. Adv. Cryptology—EUROCRYPT 2013*, May 2013, pp. 296–312.

[156] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage," in *Proc. 22nd USENIX Security Symp.*, Aug. 2013, pp. 1–16.

[157] O. Heen, C. Neumann, L. Montalvo, and S. Defrance, "Improving the resistance to side-channel attacks on cloud storage services," in *Proc. 5th Int. Conf. New Technol. Mobility Security*, May 2012, pp. 1–5.

[158] R. C. Merkle, "A certified digital signature," in *Proc. Adv. Cryptol.*, 1989, pp. 218–238.

[159] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proc. 7th ACM Symp. Inf. Comput. Commun. Security*, May 2012, pp. 81–82.

[160] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *Proc. 2nd ACM Conf. Data Appl. Security Privacy*, Feb. 2012, pp. 1–12.

[161] J. Xu, E.-C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *Proc. 8th ACM SIGSAC Symp. Inf. Comput. Commun. Security*, May 2013, pp. 195–206.

[162] Y. Zhou *et al.*, "SecDep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management," in *Proc. IEEE 31st Symp. Mass Storage Syst. Technol.*, Jun. 2015, pp. 1–12.

[163] M. Li, C. Qin, and P. P. Lee, "CDStore: Toward reliable, secure, cost-efficient cloud storage via convergent dispersal," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Jul. 2015, pp. 111–124.

[164] M. Li, C. Qin, P. P. Lee, and J. Li, "Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds," in *Proc. 6th USENIX Workshop Hot Topics Storage File Syst.*, Jun. 2014, pp. 1–5.

[165] D. Bhagwat *et al.*, "Providing high reliability in a minimum redundancy archival storage system," in *Proc. 14th IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Sep. 2006, pp. 413–421.

[166] X. Li, M. Lillibridge, and M. Uysal, "Reliability analysis of deduplicated and erasure-coded storage," *ACM SIGMETRICS Performance Eval. Rev.*, vol. 38, no. 3, pp. 4–9, 2011.

[167] C. Liu, Y. Gu, L. Sun, B. Yan, and D. Wang, "R-ADMAD: High reliability provision for large-scale de-duplication archival storage systems," in *Proc. 23rd Int. Conf. Supercomput.*, Jun. 2009, pp. 370–379.

[168] K. Li, "Emerging Technology: DD200 Restorer," Apr. 2004. [Online]. Available: http://storageconference.us/2004/Presentations/Panel/KaiLi.pdf

[169] E. W. Rozier *et al.*, "Modeling the fault tolerance consequences of deduplication," in *Proc. 30th IEEE Symp. Reliable Distrib. Syst.*, Oct. 2011, pp. 75–84.

[170] C. Constantinescu and M. Lu, "Quick estimation of data compression and de-duplication for large storage systems," in *Proc. 1st Int. Conf. Data Compression Commun. Process.*, Jun. 2011, pp. 98–102.

[171] K. Tangwongsan, H. Pucha, D. G. Andersen, and M. Kaminsky, "Efficient similarity estimation for systems exploiting data redundancy," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[172] F. Xie, M. Condict, and S. Shete, "Estimating duplication by content-based sampling," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Jun. 2013, pp. 181–186.

[173] D. Harnik, O. Margalit, D. Naor, D. Sotnikov, and G. Vernik, "Estimation of deduplication ratios in large data sets," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, May 2012, pp. 1–11.

[174] D. Meister, A. Brinkmann, and T. Süß, "File recipe compression in data deduplication systems," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 175–182.

[175] A. Katiyar and J. Weissman, "ViDeDup: An application-aware framework for video de-duplication," in *Proc. 3rd USENIX Conf. Hot Topics Storage file Syst.*, Jun. 2011, pp. 1–5.

[176] D. Perra and J.-M. Frahm, "Cloud-scale image compression through content deduplication," in *Proc. British Mach. Vis. Conf.*, Sep. 2014, pp. 1–12.

[177] Y. Hua, H. Jiang, and D. Feng, "FAST: Near real-time searchable data analytics for the cloud," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, 2014, pp. 754–765.

[178] S. Dewakar *et al.*, "Storage efficiency opportunities and analysis for video repositories," in *Proc. 7th USENIX Workshop Hot Topics Storage File Syst.*, Jul. 2015.

[179] R. Koller and R. Rangaswami, "I/O deduplication: Utilizing content similarity to improve I/O performance," *ACM Trans. Storage*, vol. 6, no. 3, pp. 13, 2010.

[180] M. Vrable, S. Savage, and G. M. Voelker, "Cumulus: Filesystem backup to the cloud," *ACM Trans. Storage*, vol. 5, no. 4, pp. 14, 2009.

[181] C. A. Waldspurger, "Memory resource management in VMware ESX server," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, 2002.

[182] A. T. Clements *et al.*, "Decentralized deduplication in SAN cluster file systems," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2009, pp. 1–14.

[183] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 87–95, 2000.

[184] A. Anand, V. Sekar, and A. Akella, "SmartRE: An architecture for coordinated network-wide redundancy elimination," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 87–98, 2009.

[185] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. 9th USENIX Conf. File Storage Technol.*, Feb. 2011, pp. 1–14.

[186] C. Li *et al.*, "Nitro: A capacity-optimized SSD cache for primary storage," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2014, pp. 501–512.

[187] S. Rhea, R. Cox, and A. Pesterev, "Fast, inexpensive content-addressed storage in foundation," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2008, pp. 143–156.

[188] Opendedup. [Online]. Available: http://www.opendedup.org/

[189] P. Koutoupis, "Data deduplication with linux," *Linux J.*, vol. 2011, no. 207, pp. 7, 2011.

[190] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, "Benchmarking personal cloud storage," in *Proc. Conf. Internet Meas. Conf.*, Oct. 2013, pp. 205–212.

[191] K. Miller, F. Franz, M. Rittinghaus, M. Hillenbrand, and F. Bellosa, "XLH: More effective memory deduplication scanners through cross-layer hints," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2013, pp. 279–290.

[192] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proc. SYSTOR'09, Israeli Exp. Syst. Conf.*, May 2009, pp. 1–14.

[193] J. Ren and Q. Yang, "A new buffer cache design exploiting both temporal and content localities," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Jun. 2010, pp. 273–282.

[194] C.-H. Ng, M. Ma, T.-Y. Wong, P. P. Lee, and J. Lui, "Live deduplication storage of virtual machine images in an open-source cloud," in *Proc. 12th Int. Middleware Conf.*, Dec. 2011, pp. 80–99.

[195] X. Zhang *et al.*, "Exploiting data deduplication to accelerate live virtual machine migration," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2010, pp. 88–96.

[196] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proc. 20th Int. Symp. High Performance Distrib. Comput.*, Jun. 2011, pp. 135–146.

[197] H. Pucha, D. G. Andersen, and M. Kaminsky, "Exploiting similarity for multi-source downloads using file handprints," in *Proc. 4th USENIX Conf. Netw. Syst. Design Implement.*, Apr. 2007, pp. 1–14.

[198] S. Sanadhya *et al.*, "Asymmetric caching: Improved network deduplication for mobile devices," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, Aug. 2012, pp. 161–172.

[199] Y. Hua, X. Liu, and D. Feng, "Smart in-network deduplication for storage-aware SDN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 2013, pp. 509–510.

[200] Y. Hua and X. Liu, "Scheduling heterogeneous flows with delay-aware

deduplication for avionics applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 9, pp. 1790–1802, 2012.

[201] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND flash-based SSDs," in *Proc. 9th USENIX Conf. File Storage Technol.*, Feb. 2011, pp. 91–103.

[202] Y. Tang and J. Yang, "Secure deduplication of general computations," in *Proc. USENIX Annu. Tech. Conf.*, Jul. 2015, pp. 319–331.

[203] HP, "Eliminate the boundaries of traditional backup and archive," Apr. 2014. [Online]. Available: http://www8.hp.com/us/en/products/data-storage/storage-backup-archive.html

[204] "HYDRAstor—Scale-out Grid Storage Platform," Apr. 2014. [Online]. Available: http://www.necam.com/hydrastor/

[205] Commvault Simpana software. [Online]. Available: http://www.commvault.com/simpana-software

[206] IDC, "Worldwide Purpose-Built Backup Appliance (PBBA) Market Posts 9.7% Year-Over-Year Revenue Growth in Fourth Quarter of 2013," Mar. 2014. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS24762914

[207] "Is DDBoost a 'standard'," Apr. 2014. [Online]. Available: http://www.emc.com/data-protection/data-domain/data-domain-boost.htm

[208] *Oracle Database Backup and Recovery User's Guide*. [Online]. Available: http://docs.oracle.com/cd/E11882_01/backup.112/e10642/rcmquick.htm#BRADV89346

[209] "Symantec looks beyond vrtual tape," Nov. 2006. [Online]. Available: http://www.enterprisestorageforum.com/technology/news/article.php/3643846/Symantec-Looks-Beyond-Virtual-Tape.htm

[210] "Avamar deduplication backup software and system." [Online]. Available: http://www.emc.com/domains/avamar/index.htm

[211] J. Liu, Y. Chai, X. Qin, and Y. Xiao, "PLC-Cache: Endurable SSD cache for deduplication-based primary storage," in *Proc. 30th IEEE Symp. Mass Storage Syst. Technol.*, Jun. 2014, pp. 1–6.

[212] A. Wildani, E. L. Miller, and O. Rodeh, "Hands: A heuristically arranged non-backup in-line deduplication system," in *Proc. IEEE 29th Int. Conf. Data Eng.*, Apr. 2013, pp. 446–457.

[213] V. Tarasov *et al.*, "Generating realistic datasets for deduplication analysis," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Jun. 2012, pp. 24–34.

[214] J. Paulo, P. Reis, J. Pereira, and A. Sousa, "Dedisbench: A benchmark for deduplicated storage systems," in *Proc. Move Meaningful Internet Syst. (OTM)*, 2012, pp. 584–601.

[215] R. Gracia-Tinedo *et al.*, "SDGen: Mimicking datasets for content generation in storage benchmarks," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 317–330.

[216] Symantec OpenStorage. [Online]. Available: http://www.symantec.com/page.jsp?id=openstorage, Nov. 2006.

[217] B. Cohen, "The BitTorrent protocol specification," 2008.

[218] Y. Tang, P. P. Lee, J. Lui, and R. Perlman, "Secure overlay cloud storage with access control and assured deletion," *IEEE Trans. Depend. Secure Comput.*, vol. 9, no. 6, pp. 903–916, 2012.

[219] A. Gharaibeh *et al.*, "DedupT: Deduplication for tape systems," in *Proc. IEEE 30th Symp. Mass Storage Syst. Technol.*, Jun. 2014, pp. 1–11.

ABOUT THE AUTHORS

Wen Xia (Member, IEEE) received the Ph.D. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014.

He is currently an Assistant Professor in the School of Computer Science and Technology, HUST. His research interests include deduplication, data compression, storage systems, cloud storage, etc. He has published more than 20 papers in major journals and international conferences including IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, USENIX ATC, USENIX FAST, INFOCOM, IFIP Performance, IEEE DCC, MSST, IPDPS, HotStorage, etc.

Dr. Xia is a member of the Association for Computing Machinery (ACM), the China Computer Federation (CCF), and USENIX.

Hong Jiang (Fellow, IEEE) received the B.Sc. degree in computer engineering from Huazhong University of Science and Technology, Wuhan, China, in 1982, the M.A.Sc. degree in computer engineering from the University of Toronto, Toronto, ON, Canada, in 1987, and the Ph.D. degree in computer science from the Texas A&M University, College Station, TX, USA, in 1991.

He is currently Chair and Wendell H. Nedderman Endowed Professor of Computer Science and Engineering Department at the University of Texas at Arlington (UTA), Arlington, TX, USA. Prior to joining UTA, he served as a Program Director at the National Science Foundation (2013–2015), and he was at the University of Nebraska—Lincoln, Lincoln, NE, USA (since 1991), where he was Willa Cather Professor of Computer Science and Engineering. He has graduated 13 Ph.D. students who upon their graduations either landed academic tenure-track positions in Ph.D.-granting U.S. institutions or were employed by major U.S. IT corporations. He has over 200 publications in major journals and international conferences in these areas, including IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, Proceedings of the IEEE, ACM-TACO, JPDC, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by the National Science Foundation (NSF), the Department of Defense (DOD), the State of Texas, and the State of Nebraska. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, performance evaluation.

Dr. Jiang recently served as an Associate Editor of the IEEE Transactions on Parallel and Distributed Systems. He is a Member of the Association for Computing Machinery (ACM).

Dan Feng (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science and technology from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1991, 1994, and 1997, respectively.

She is a Professor and the Dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications in major journals and international conferences, including IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, ACM-TOS, JCST, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP.

Dr. Feng has served as the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012, 2015. She is a member of the Association for Computing Machinery (ACM).

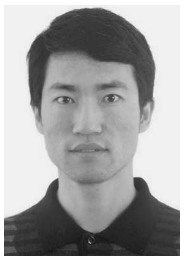
Fred Douglis (Senior Member, IEEE) received the B.S. degree in computer science from Yale University, New Haven, CT, USA, in 1984 and the M.S. and Ph.D. degrees in computer science from the University of California Berkeley, Berkeley, CA, USA, in 1987 and 1990, respectively.

He has worked for EMC Corporation, Princeton, NJ, USA, since 2009, focusing on systems and storage technologies such as flash memory, deduplication, compression, load balancing, and others. He previously worked in other industrial applied research organizations, including Matsushita, AT&T (Bell) Labs, and IBM Research, and he has been a visiting professor at VU Amsterdam and Princeton University. He has authored more than 50 papers in major journals and conferences such as ACM ToS, FAST, USENIX ATC, ACM Middleware, World Wide Web, and others, and he is an inventor of approximately 60 issued U.S. patents.

Dr. Douglis served as the Editor-in-Chief of IEEE Internet Computing from 2007 to 2010 and has been on its editorial board since 1999. He is a member of the IEEE Computer Society Board of Governors from 2016 to 2018, an Associate Editor of the IEEE Transactions on Computers, and a member of the Association for Computing Machinery (ACM) and USENIX.

Philip Shilane received the B.S. and M.S. degrees in computer science from Stanford University, Stanford, CA, USA, in 2000 and 2001, respectively, and the M.A. and Ph.D. degrees in computer science from Princeton University, Princeton, NJ, USA, in 2004 and 2008, respectively.

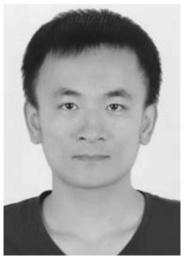
Since 2007, he has worked for Data Domain and then EMC Corporation, Princeton, NJ, USA, in research and advanced development within a CTO organization in the areas of computer storage systems, deduplication, compression, data characterization, flash caching, and nonvolatile memory. He has more than 25 publications in journals and conferences including ACM ToS, ACM ToG, SIGGRAPH, USENIX ATC, FAST, Middleware, SMI, MSST, LISA, HotStorage, etc. He is an inventor of over 30 patents.



Yu Hua (Senior Member, IEEE) received the B.E. and Ph.D. degrees in computer science from the Wuhan University, Wuhan, China, in 2001 and 2005, respectively.

He is a Professor at the Huazhong University of Science and Technology, Wuhan, China. His research interests include computer architecture, cloud computing, and network storage. He has more than 60 papers to his credit in major journals and international conferences including the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, USENIX ATC, FAST, INFOCOM, SC, ICDCS, and MSST.

Dr. Hua has served on the program committee for multiple international conferences, such as INFOCOM, RTSS, ICDCS, MSST, ICNP, ICPP, IWQoS. He is a senior member of CCF.

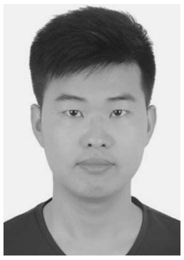


Min Fu is currently working toward the Ph.D. degree in computer architecture at Huazhong University of Science and Technology, Wuhan, China.

His current research interests include data deduplication, storage systems, and reliability. He has several papers in major journals and conferences including IEEE Transactions on Parallel and Distributed Systems, USENIX ATC, FAST, etc.

Yucheng Zhang is currently working toward the Ph.D. degree in computer architecture at Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interests include data deduplication, storage systems, etc. He has several papers in refereed journals and conferences including the IEEE Transactions on Computers and IEEE INFOCOM.



Yukun Zhou is currently working toward the Ph.D. degree in computer architecture at Huazhong University of Science and Technology (HUST), Wuhan, China.

His research interests include data deduplication, storage security, etc. He has several papers in refereed journals and conferences including PEVA, MSST, etc.