DARE: A Deduplication-Aware Resemblance Detection and Elimination Scheme for Data Reduction with Low Overheads

Wen Xia, *Member, IEEE*, Hong Jiang, *Fellow, IEEE*, Dan Feng, *Member, IEEE*, and Lei Tian, *Senior Member, IEEE*

Abstract—Data reduction has become increasingly important in storage systems due to the explosive growth of digital data in the world that has ushered in the big data era. One of the main challenges facing large-scale data reduction is how to maximally detect and eliminate redundancy at very low overheads. In this paper, we present DARE, a low-overhead deduplication-aware resemblance detection and elimination scheme that effectively exploits existing *duplicate-adjacency* information for highly efficient resemblance detection in data deduplication based backup/archiving storage systems. The main idea behind DARE is to employ a scheme, call Duplicate-Adjacency based Resemblance Detection (*DupAdj*), by considering any two data chunks to be similar (i.e., candidates for delta compression) if their respective adjacent data chunks are duplicate in a deduplication system, and then further enhance the resemblance detection efficiency by an improved super-feature approach. Our experimental results based on real-world and synthetic backup datasets show that DARE only consumes about 1/4 and 1/2 respectively of the computation and indexing overheads required by the traditional super-feature approaches while detecting 2-10 percent more redundancy and achieving a higher throughput, by exploiting existing duplicate-adjacency information for resemblance detection and finding the "sweet spot" for the super-feature approach.

Index Terms—Data deduplication, delta compression, storage system, index structure, performance evaluation

1 Introduction

The amount of digital data is growing explosively, as evidenced in part by an estimated amount of about 1.2 zettabytes and 1.8 zettabytes respectively of data produced in 2010 and 2011 [1], [2]. As a result of this "data deluge", managing storage and reducing its costs have become one of the most challenging and important tasks in mass storage systems. According to a recent IDC study [3], almost 80 percent of corporations surveyed indicated that they were exploring data deduplication technologies in their storage systems to increase storage efficiency.

Data deduplication is an efficient data reduction approach that not only reduces storage space [4], [5], [6], [7], [8], [9], [10] by eliminating duplicate data but also minimizes the transmission of redundant data in low-bandwidth network environments [11], [12], [13], [14]. In general, a chunk-level data deduplication scheme splits data blocks of a data stream (e.g., backup files, databases, and virtual machine images) into multiple data chunks that are each uniquely identified and duplicate-detected by a secure SHA-1 or MD5 hash signature (also called a fingerprint) [5], [11]. Storage systems then remove duplicates of data chunks and store only one copy of them to achieve the goal of space savings.

While data deduplication has been widely deployed in storage systems for space savings, the fingerprint-based deduplication approaches have an inherent drawback: they often fail to detect the similar chunks that are largely identical except for a few modified bytes, because their secure hash digest will be totally different even only one byte of a data chunk was changed [4], [5], [12], [15], [16]. It becomes a big challenge when applying data deduplication to storage datasets and workloads that have frequently modified data, which demands an effective and efficient way to eliminate redundancy among frequently modified and thus similar data.

Delta compression, an efficient approach to removing redundancy among similar data chunks has gained increasing attention in storage systems [12], [17], [18], [19], [20]. For example, if chunk A_2 is similar to chunk A_1 (the base-chunk), the delta compression approach calculates and then only stores the differences (delta) and mapping relation between A_2 and A_1. Thus, it is considered a promising technique that effectively complements the fingerprint-based deduplication approaches by detecting similar data missed by the latter.

One of the main challenges facing the application of delta compression in deduplication systems is *how to accurately*

- *W. Xia is with the School of Computer Science and Technology, Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, HuBei, China. E-mail: xia@hust.edu.cn.*
- *H. Jiang is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, 217 Schorr Center, 1101 T Street, Lincoln, NE 68588-0150. E-mail: jiang@cse.unl.edu.*
- *D. Feng is with the Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, HuBei, China. E-mail: dfeng@hust.edu.cn.*
- *L. Tian is with Tintri, Mountain View, CA 94043. E-mail: leitian.hust@gmail.com.*

Manuscript received 25 July 2014; revised 29 June 2015; accepted 1 July 2015. Date of publication 12 July 2015; date of current version 16 May 2016. Recommended for acceptance by A. Mendelson. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2015.2456015

detect the most similar candidates for delta compression with low overheads. The state-of-the-art solutions [12], [15], [16], [17] detect similarity for delta compression by computing several Rabin fingerprints as *features* and grouping them into *super-fingerprints*, also referred to as ***super-features (SF)*** (detailed in Section 3.3). Nevertheless, to index a dataset of 80 TB and assuming an average chunk size of 8 KB and 16 bytes per index entry, for example, about 200 GB worth of super-feature index entries must be generated, which will still be too large to fit in memory [12]. Since the random accesses to on-disk index are much slower than that to RAM, the frequent accesses to on-disk super-features will cause the system throughput to become unacceptably low for the users [6], [12], [21].

The existing solutions to the indexing issue of delta compression either record the resemblance information for files, instead of data chunks, so that similarity index entries can fit in the memory [22], [23], or exploit the locality of backup data streams in deduplication-based backup/archiving systems, which avoids the global indexing on the disk [12], [17]. The first approach faces an implementation difficulty in large-scale data deduplication systems since it is hard to record all the resemblance or version information of files in such systems [12]. The second approach often fails to detect a significant amount of redundant data when the workloads lack locality. Another challenge facing the super-feature method is the high overhead in computing the super-features. According to a recent study of delta compression [17] and our experimental observation, the throughput of computing super-features is about 30 MB/s (see Section 3.3 for details), which may become a potential bottleneck for deduplication-based storage systems, particularly if most index entries are fit in memory or partially on SSD-based storage for which the throughput can be hundreds of MB per second or higher.

From our observation of duplicate and similar data of backup streams, we find that the non-duplicate chunks that are adjacent to duplicate ones could be considered good delta compression candidates in data deduplication systems. Thus we propose the approach of *Duplicate-Adjacency based Resemblance Detection*, or *DupAdj* for short. Exploiting this existing deduplication information (i.e., duplicate-adjacency) not only avoids the high overhead of super-feature computation but also reduces the size of index entries for resemblance detection. On the other hand, our study of the existing super-feature approaches reveals that *the traditional super-feature method can be improved with fewer features per super-feature*, which works very effectively on deduplication systems when combined with the aforementioned DupAdj approach.

In this paper, we propose DARE, a low-overhead Deduplication-Aware Resemblance detection and Elimination scheme for deduplication based backup and archiving storage system. The main idea of DARE is to effectively exploit existing duplicate-adjacency information to detect similar data chunks *(DupAdj)*, refine and supplement the detection by using an improved super-feature approach *(Low-Overhead Super-Feature)* when the existing duplicate-adjacency information is lacking or limited. In addition, we present an analytical study of the existing super-feature approach with a mathematic model and conduct an empirical evaluation of this approach with several real-world workloads in data deduplication systems.

Our experimental evaluation results, based on real-world and synthetic backup datasets, show that DARE significantly outperforms the traditional Super-Feature approach. More specifically, the DupAdj approach achieves a similar data reduction efficiency to the pure super-feature approach and DARE detects 2-10 percent more redundant data while achieving a higher throughput of data reduction than the pure super-feature approach. Meanwhile, DARE only consumes about 1/4 and 1/2 respectively of the computation and indexing overheads required by the traditional super-feature approach for resemblance detection. It is important to note that our evaluation also demonstrates the superior data-restore performance of the DARE-enhanced deduplication system over the deduplication-only systems via delta compression, where the former outperforms the latter by a factor of 2 (2×).

The rest of the paper is organized as follows. Section 2 presents background and motivation for this work. Section 3 describes the architecture, key data structures, and resemblance detection schemes of DARE. Section 4 presents our experimental evaluation of the DARE prototype and discusses the results. Section 5 draws conclusions and provides directions for future work.

2 Background and Motivation

In this section, we first present the necessary background knowledge about resemblance detection for data reductions in storage systems, then provide analytical and experimental observations that motivate our research on resemblance detection for data reduction.

2.1 Resemblance Detection Based Data Reduction

Data deduplication is becoming increasingly popular in data-intensive storage systems as one of the most efficient data reduction approaches in recent years. Fingerprint-based deduplication techniques eliminate duplicate chunks by checking their secure-fingerprints (i.e., SHA-1/SHA-256 signatures), which has been widely used in commercial backup and archiving storage systems [6], [24], [25], [26], [27].

Previous studies on data deduplication have shown that one challenge lies in the system scalability issue of index-lookup. That is, the fingerprints of a multi-TB-scale storage system will be too large to fit in memory and must be moved to the disk, which causes long latencies of random disk I/Os for fingerprint index-lookup. Most existing solutions to this problem aim to make full use of RAM, by putting only the hot fingerprints into RAM to reduce accesses to on-disk index. DDFS [6] and Sparse Indexing [25] attempt to avoid the disk bottleneck for deduplication indexing by exploiting the inherent locality of the backup streams and preserving this locality in the memory to increase cache hit ratio. Locality here means that the chunks of a backup stream will appear in approximately the same order in each full backup with a high probability. Extreme Binning [28] and SiLo [29] exploits similarity-only and similarity & locality of the backup data streams respectively to minimize RAM overhead for

TABLE 1
Comparisons between Duplicate Detection and Resemblance Detection for Data Reduction Systems

	Duplicate Detection	Resemblance Detection
Objects	Duplicate data	Similar data
Granularity	Chunk-level	Byte-level
Rep. Methods	Secure-Fingerprint based Deduplication	Super-Feature based Delta Compression
Scalability	Strong	Weak
Rep. Systems	LBFS [11], Venti[5], DDFS [6]	REBL[16], DERD[15], SIDC[12]

deduplication index-lookup. ChunkStash puts the fingerprint index on SSD by means of a memory-efficient data structure called cuckoo hash to accelerate index-lookup for data deduplication [21].

Another challenge for data deduplication is how to maximally detect and eliminate data redundancy in storage systems by determining appropriate data chunking schemes. In order to find more redundant data, the content-defined chunking (CDC) approach was proposed in LBFS to find the proper cut-point of each chunk in the files and address the boundary-shift problem [9], [11], [30]. Re-chunking approaches were also proposed to divide those non-duplicate chunks into smaller ones to expose and detect more redundancy [31], [32], [33].

Resemblance detection with delta compression[15], [16], [26], as another approach to data reduction in storage systems, was proposed more than 10 years ago but was later overshadowed by fingerprint-based deduplication [6], [24], [25] due to the former's scalability issue. Table 1 compares these two data reduction approaches. Resemblance detection detects redundancy among similar data at the byte level while duplicate detection finds totally identical data at the chunk level, which makes the latter much more scalable than the former in mass storage systems.

REBL[16] and DERD [15] are typical super-feature-based resemblance detection approaches for data reduction. They compute the features of the data stream (e.g., Rabin Fingerprints [34]) and group features into super-features to capture the resemblance of data and then delta compress the data. TAPER [35] presents a Bloom-Filter solution that measures the similar files based on the chunk fingerprints recorded in Bloom Filters. All these approaches require high computation and indexing overheads for resemblance detection. As a result, the simpler and faster deduplication method has become a more popular data reduction approach in the last five years [6], [7], [8].

Nevertheless, resemblance detection is gaining increasing tractions in storage systems because of its ability to capture and eliminate data redundancy among similar but non-duplicate data chunks that effectively complements fingerprint-based deduplication. Difference Engine [20] employs Xdelta [23] to further eliminate memory redundancy and thus enlarge the logical RAM space in VM environments. I-CASH [18] delta compresses similar data to enlarge the logical space of SSD caches. Shilane et al. [12] proposed a stream-informed delta compression (SIDC) approach to reducing similar data transmission and thus accelerating data replication in a WAN environment. This approach is super-feature based and complements the chunk-level deduplication by only detecting resemblance among non-duplicate chunks in the cache that preserves the backup stream locality. It avoids the costly global indexing, at a limited loss of resemblance detection. While the combined detection of duplicate and resemblance promises to achieve a superior data reduction performance, challenges of relatively high computation and indexing overheads stemming from resemblance detection remain [17].

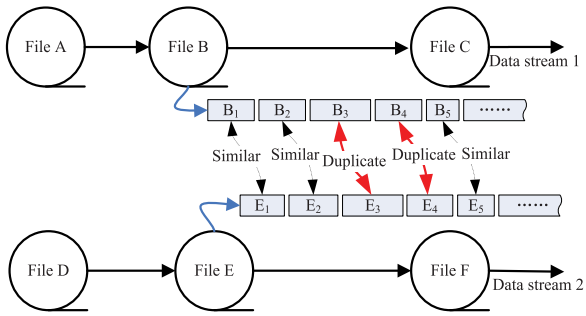


Fig. 1. A conceptual illustration of the duplicate adjacency. The non-duplicate chunks adjacent to duplicate ones are considered potentially similar and thus good delta compression candidates.

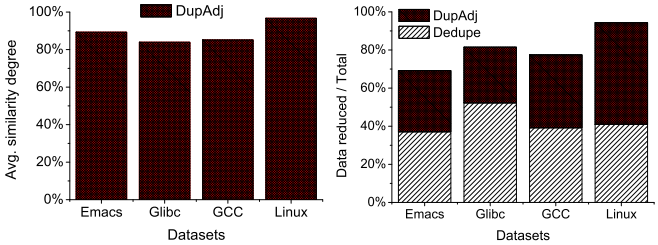
Note that SIDC [12] is the most related work to DARE. Different from SIDC that implements traditional super-feature based delta compression in a stream-informed (i.e., locality preserved) cache, DARE first employs a duplicate-adjacency based resemblance detection scheme (see Section 3.2) and then an improved super-feature based approach (see Section 3.3) to jointly and more effectively reduce the indexing and computation overheads for delta compression.

2.2 Fact of Duplicate Adjacency

As discussed in Section 2.1, the modified chunks may be very similar to their previous versions in a backup system while unmodified chunks will remain duplicate and are easily identified by the deduplication process. *For those non-duplicate chunks that are location-adjacent to known duplicate data chunks in a deduplication system, it is intuitive and quite possible that only a few bytes of them are modified from the last backup, making them potentially excellent delta compression candidates.*

Fig. 1 illustrates a case of duplicate data chunks and their immediate non-duplicate neighbors. As mentioned above, our intuition is that the latter are highly likely to be similar and thus good delta compression candidates. Specifically, since chunks B_3 & B_4 are duplicates of chunks E_3 & E_4 in Fig. 1 respectively, their immediate neighbors, the chunk-pairs B_1 & E_1, B_2 & E_2, and B_5 & E_5, are then considered good delta compression candidates, which is consistent with the aforementioned backup-stream locality [6], [12], [25], [29], [36].

If we can make full use of the existing knowledge about duplicate data chunks in a deduplication system, it is possible for us to detect similar chunks without the overheads of computing and storing features & super-features and then accessing their on-disk index. Fig. 2 shows important preliminary results of this duplicate-adjacency-based



(a) Similarity degree of the DupAdj detected chunks

(b) Redundancy reduced by DupAdj-based delta compression

Fig. 2. A study of redundancy elimination on the four real-world tarred datasets by 8KB-level deduplication and then DupAdj-based delta compression.

resemblance detection approach, called DupAdj, on several real-world datasets whose workload characteristics are detailed in Table 2 in Section 4.1. First, the similarity degree (i.e., $\frac{delta\ \ compressed\ \ size}{chunk\ \ size}$) of the DupAdj-detected chunks tends to be very high, on average, about 84-96 percent on the four backup datasets as shown in Fig. 2a. Second, by exploiting this duplicate adjacency information, the DupAdj-based post deduplication delta compression approach can further detect and eliminate about 30-50 percent redundancy from the non-duplicate but duplicate-adjacent chunks as shown in Fig. 2b. Hence, this DupAdj approach, detailed in Section 3.2, is very effective for detecting possible similar chunks and then delta encoding them to further remove redundancy in deduplication-based backup systems while significantly simplifying the resemblance detection process.

2.3 Rethinking of the Super-Feature Approaches

Similar data, like duplicate data, are in wide existence in backup systems [8], [17]. Meister and Brinkmann [37] find that small semantic changes on documents may result in big modifications in the binary representation of files, and delta compression is more effective in eliminating redundancy in such cases. To support delta compression, resemblance detection will be required for selecting suitable similar candidates.

Some early research on resemblance detection of near-duplicate or similar files was performed by Broder for search engine results [38], [39] and Manber for filesystems [40]. REBL [16] and DERD [15] used an efficient super-feature approach to eliminating redundancy with delta compression in the early data reduction systems. However, their super-feature approaches are arguably very different from the most recently required resemblance detection in the current large-scale storage systems in that:

- The datasets used in REBL and DERD are more likely of primary storage workloads and only several hundreds of MB in size, in contrast to typical datasets of deduplication sytems that are usually of the TB/PB scale [8], [12], [41].
- The chunk size tested in their papers is in the 1-4 KB range and hashing region of each feature is of 4-22 bytes, in contrast to typical deduplication systems that adopt the larger chunk size of 8 KB and larger feature hashing region of 32 or 48 bytes (i.e., CDC sliding windows) [6], [8], [11].

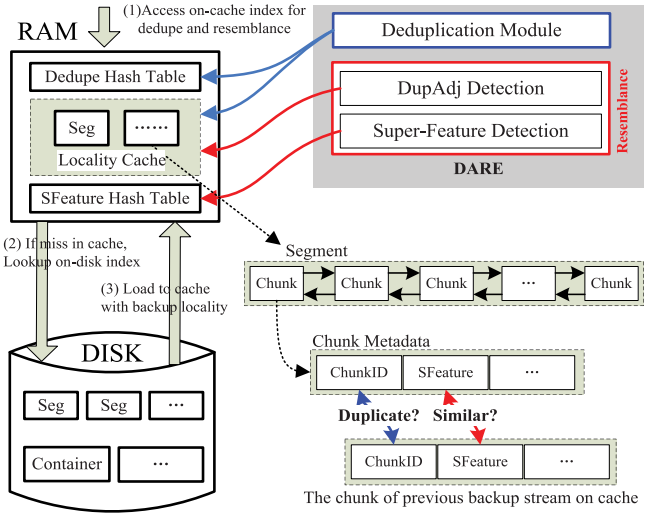


Fig. 3. Architecture and key data structures of the DARE system that combines duplicate detection and resemblance detection for data reduction.

- The post-deduplication chunks tend to be more frequently modified in backup systems, which may make the resemblance of these non-duplicate and less similar chunks more difficult to detect.

This motivates us to rethink the traditional super-feature-based approaches in the context of deduplication based backup/archiving systems for the purpose of detecting resemblance among the post-deduplication chunks, which is detailed in Section 3.3.

3 Design and Implementation

In this section, we will first describe the architecture and key data structures of DARE, followed by detailed discussions of its design and implementation issues.

3.1 Architecture Overview

DARE is designed to improve resemblance detection for additional data reduction in deduplication-based backup/archiving storage systems. As shown in Fig. 3, the DARE architecture consists of three functional modules, namely, the Deduplication module, the DupAdj Detection module, and the improved super-feature module. In addition, there are five key data structures in DARE, namely, Dedupe Hash Table, SFeature Hash Table, Locality Cache, Container, Segment, and Chunk, which are defined below:

- A *chunk* is the atomic unit for data reduction. The non-duplicate chunks, identified by their SHA-1 fingerprints, will be prepared for resemblance detection in DARE.
- A *container* is the fixed-size storage unit that stores sequential and NOT reduced data, such as non-duplicate & non-similar or delta chunks, for better storage performance by using large I/Os [6], [36].
- A *segment* consists of the metadata of a number of sequential chunks (e.g., 1 MB size), such as the chunk fingerprints, size, etc., which serves as the atomic unit in preserving the backup-stream logical locality [36] for data reduction. Here DARE

uses a data structure of *doubly-linked* list to record the chunk adjacency information for the DupAdj detection. Note that the SFeature in the segment may be unnecessary if the DupAdj module has already confirmed this chunk as being similar for delta compression.

- *Dedupe hash table* serves to index fingerprints for duplicate detection for the deduplication module.
- *SFeature hash table* serves to index the super-features after the DupAdj resemblance detection. It manages the super-features of non-duplicate and non-similar chunks.
- *Locality cache* contains the recently accessed data segments and thus preserves the backup-stream locality in memory, to reduce accesses to the on-disk index from either duplicate detection or resemblance detection.

Here we describe a general workflow of DARE. For the input data stream, DARE will first detect duplicate chunks by the *Deduplication* module. Any of the many existing deduplication approaches [36] can be implemented here and the preservation of the backup-stream logical locality in the **segments** is required for further resemblance detection. For each non-duplicate chunk, DARE will first use its *DupAdj Detection* module (*see Section 3.2*) to quickly determine whether it is a delta compression candidate. If it is not a candidate, DARE will then compute its features and super-features, using its improved *Super-Feature Detection* module (*see Section 3.3*), to further detect resemblance for data reduction.

Because DARE adopts a caching scheme that exploits the backup-stream logical locality [36] in a way similar to the Sparse Indexing [25], SiLo [29], and BLC [42] approaches, the indexing hit ratio in the locality cache for both the deduplication and resemblance detection modules will be very high. Upon a miss in the locality cache, DARE will load the missing segment from the latest backup to the RAM with the LRU replacement policy. It is noteworthy that, after deduplication, the cached segments *that have preserved the logical-locality of chunks, including the adjacency information of the duplicate-detected chunks*, will be further exploited by DARE to detect possible resemblance among the non-duplicate data chunks, as detailed in the next section.

3.2 DupAdj: Duplicate-Adjacency Based Resemblance Detection

As a salient feature of DARE, the DupAdj approach detects resemblance by exploiting existing duplicate-adjacency information of a deduplication system. The main idea behind this approach is to consider chunk pairs closely adjacent to any confirmed duplicate-chunk pair between two data streams as resembling pairs and thus candidates for delta compression, as conceptually illustrated in Fig. 1.

According to the description of the DARE data structures in Fig. 3, DARE records the backup-stream logical locality of chunk sequence by a *doubly-linked list*, which allows an efficient search of the duplicate-adjacent chunks for resemblance detection by traversing to prior or next chunks on the list, as shown in Fig. 1. When the DupAdj Detection module of DARE processes an input segment, it will traverse all the chunks by the aforementioned *doubly-linked list* to find the already duplicate-detected chunks. If chunk A_m of the input segment A was detected to be a duplicate of chunk B_n of segment B, DARE will traverse the doubly-linked list of B_n in both directions (e.g., A_{m+1} & B_{n+1} and A_{m-1} & B_{n-1}) in search of potentially similar chunk pairs between segments A and B, until a dissimilar chunk or an already detected duplicate or similar chunk is found. Note that the detected chunks here are considered dissimilar (i.e., NOT similar) to others if their similarity degree (i.e., $\frac{dela\ compressed\ size}{chunk\ size}$) is smaller than a predefined threshold, such as 0.25, a false positive for resemblance detection. Actually, the similarity degree of the DupAdj-detected chunks tends to be very high, larger than 0.88, as shown in Fig. 2 in Section 2.2.

In general, the overheads for the DupAdj based approach are twofold:

- *Memory overhead*: Each chunk will be associated with two pointers (about 8 or 16 Bytes) for building the doubly-linked list when DARE loads the segment into the locality cache. But when the segment is evicted from the cache, the doubly-linked list will be immediately freed. Therefore, this RAM memory overhead is arguably negligible given the total capacity of the locality cache.
- *Computation overhead*: Confirming the similarity degree of the DupAdj-detected chunks may introduce additional but ommitted computation overhead. First, the delta encoding results for the confirmed resembling (i.e., similar) chunks will be directly used as the final delta chunk for storage. Second, the actual extra computation overhead occurs when the DupAdj-detected chunks are NOT similar, which is a very rare event as discussed in the previous paragraph.

In all, the DupAdj detection approach only adds a doubly-linked list to an existing deduplication system, DARE avoids the computation and indexing overheads of the conventional super-feature approach. In case where the duplicate-adjacency information is lacking, limited, or interrupted due to operations such as file content insertions/deletions or new file appending, DARE will use an improved super-feature approach to further detect and eliminate resemblance as discussed in the next section.

3.3 Improved Super-Feature Approach

As mentioned in Section 2.1, traditional super-feature approaches generate features by Rabin fingerprints and group these features into super-features to detect resemblance for data reduction. For example, $Feature_i$ of a chunk (length = N), is uniquely generated with a randomly predefined value pair m_i & a_i and N Rabin fingerprints (as used in content-defined chunking [11]) as follows:

$$Feature_i = Max_{j=1}^{N}\{(m_i * Rabin_j + a_i)\, mod\, 2^{32}\}. \quad (1)$$

A super-feature of this chunk, $SFeature_x$, can then be calculated by several such features as follows:

$$SFeature_x = Rabin(Feature_{x*k}, \ldots, Feature_{x*k+k-1}). \quad (2)$$

For example, to generate two super-features with k=4 features each, we must first generate eight features, namely, features 0...3 for $SFeature_1$ and features 4...7 for $SFeature_2$. For similar chunks that differ only in a tiny fraction of bytes, most of their features will be identical due to the random distribution of the chunk's maximal-feature positions [38]. Thus two data chunks can be considered very similar if any one of their super-features matches. The state-of-the-art studies on delta compression and resemblance detection [12], [16] recommend the use of 4 or more features to generate a super-feature to minimize false positives of resemblance detection.

However, our theoretical analysis and experimental observations suggest that the probability of false positives resulting from feature collision is extremely low but increasing the number of features per super-feature actually decreases the efficiency of resemblance detection. First, the false positives of 64-bit Rabin fingerprints tend to be very low as discussed in [34], [43]. This means that two chunks will have the same content of hashing region (32 or 48 bytes) with a very high probability if they have the same Rabin fingerprint. Next, the probability of two similar chunks having the same feature is highly dependent upon their similarity degree according to Broder's theorem [39]. The less similar two data chunks are to each other, the smaller the probability there will be of them having the same feature. Thus, the probability of two data chunks S_1 and S_2 being detected as resembling to each other by N features can be computed as follows:

$$Pr[\bigcap_{i=1}^{N} max_i(H(S_1)) = max_i(H(S_2))] = \left\{\frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}\right\}^N = \gamma^N. \quad (3)$$

This probability is clearly decreasing as a function of the number of features used in a super-feature, as indicated by the above probability expression. Nevertheless, all recent studies on delta compression suggest to increase the number of super-features [12], [17]. If any one of the super-features of two data chunks matches, the two chunks are considered similar to each other. Thus, the probability of resemblance detection, expressed as $1-(1-\gamma^N)^M$, can be increased by the number of super-features, M.

For simplicity, assume that the similarity degree γ follows a uniform distribution in the range [0, 1] (note that the actual distribution may be much more complicated in real workloads), the expected value of resemblance detection can be expressed as a function of the number of features per super-feature and the number of super-features under the aforementioned assumption as:

$$\int_0^1 x(1-(1-x^N)^M)dx = \sum_{i=1}^{M} C_M^i(-1)^{i+1}\frac{1}{N*i+2}. \quad (4)$$

This expression of resemblance detection suggests that the larger the number of features used in obtaining a super-feature, N, is, the less capable the super-feature is of resemblance detection. On the other hand, the larger the number of super-features, M, is, the more resemblance can be detected and the more redundancy will be eliminated.

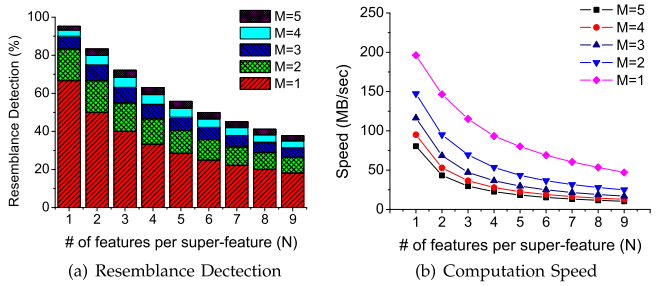


Fig. 4. The predicted data reduction efficiency and computation throughput of the super-feature approach as a function of the number of features per super-feature (N, *x*-axis) and the number of super-features (M, segments on each bar in (a) or lines in (b)).

Fig. 4a shows the trend of resemblance detection as a function of N and M. The need to increase the number of super-features suggested in Fig. 4a is consistent with the conclusion of the latest study of SIDC [12], while the suggested preference for a smaller number of features per super-feature is consistent with and verified by our experimental evaluation detailed in Section 4.2 of this paper. Please note that the computation overhead of the super-feature-based resemblance approach is proportional to the total number of features N*M, as illustrated in Fig. 4b.

In general, using fewer features per super-feature not only reduces the computation overhead but also detects more resemblance. Thus, DARE employs an improved super-feature approach with fewer features per super-feature and keeps the number of super-features stable to effectively complement the DupAdj resemblance detection. And our experimental results suggest that a configuration of three super-features and two features per super-feature appears to hit the "sweet spot" of resemblance detection in deduplication systems in terms of cost effectiveness.

3.4 Delta Compression

To reduce data redundancy among similar chunks, Xdelta [23], an optimized delta compression algorithm, is adopted in DARE after a delta compression candidate is detected by DARE's resemblance detection. DARE also only carries out the one-level delta compression for similar data as employed in DERD [15] and SIDC [12]. This is because we aim to minimize the data fragmentation problem that would cause a single read request to issue multiple read operations to multiple data chunks, a likely scenario if multi-level delta compression is employed. In other words, in DARE, delta compression will not be applied to a chunk that has already been delta compressed to avoid recursive backward referencing. And DARE records the similarity degree as the ratio of $\frac{compressed\ size}{original\ size}$ after delta compression (note that "compressed size" here refers to the size of redundant data reduced by delta compression). For example, if delta compression removes 4/5 of data volume in the input chunks detected by DARE, then the similarity degree of the input chunks is 80 percent, meaning that the volume of the input chunks can be reduced to 1/5 of its original volume by the resemblance detection and delta compression techniques.

Since delta compression needs to frequently read the base-chunks to delta compress the candidate chunks identified by resemblance detection, these frequent disk reads

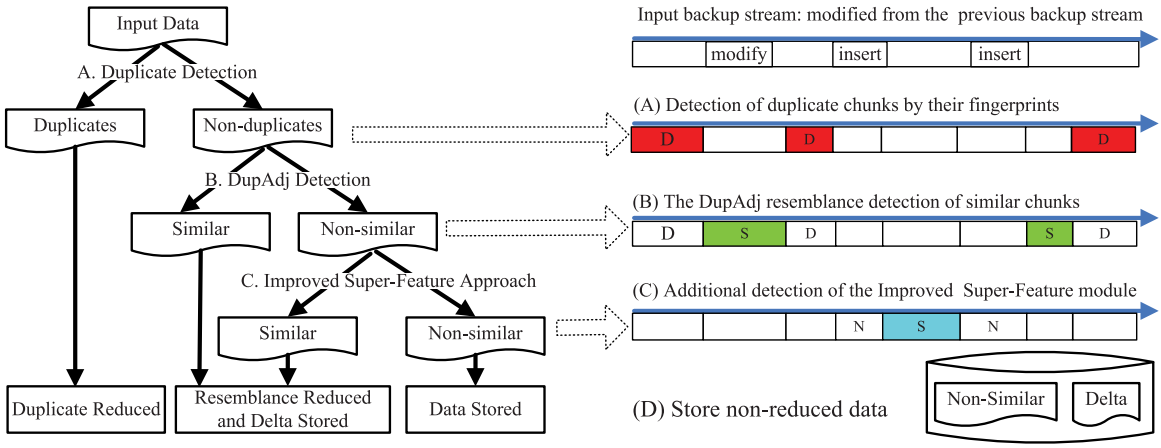


Fig. 5. The data reduction workflow of DARE, showing an example of resemblance detection for delta compression first by the DupAdj approach and then by the super-feature approach. 'D', 'S' and 'N' here refer to a duplicate chunk, a similar chunk, and a chunk that is neither duplicate nor similar, respectively.

will inevitably slow down the process of data reduction. In order to minimize disk reads, an LRU-based and backup-stream locality-preserved cache of base-chunks is implemented in DARE to load the entire container containing the missing base-chunk to the memory. While our exploitation of the backup-stream locality to prefetch base-chunks can reduce disk reads, some random accesses to on-disk base-chunks are still unavoidable as discussed in [17] and in our evaluation (see Sections 4.4 and 4.5).

3.5 Putting It All Together

To put things in perspective, Fig. 5 shows a detailed case of the processes of DARE system. For an incoming backup stream, DARE goes through the following four key steps:

1) *Duplicate detection.* The data stream is first chunked, fingerprinted, duplicate-detected, and then grouped into segments of sequential chunks to preserve the backup-stream logical locality [36]. Note that the locality information will be exploited by the following DupAdj resemblance detection.
2) *Resemblance detection.* The DupAdj resemblance detection module in DARE first detects duplicate-adjacent chunks in the segments formed in step (1). After that, DARE's improved super-feature module further detects similar chunks in the remaining non-duplicate and non-similar chunks that may have been missed by the DupAdj detection module when the duplicate-adjacency information is lacking or weak.
3) *Delta compression.* For each of the resembling chunks detected in step (2), DARE reads its base-chunk, then delta encodes their differences. In order to reduce disk reads, an LRU and locality-preserved cache is implemented here to prefetch the base-chunks in the form of data segments.
4) *Storage management.* The data NOT reduced, i.e., non-similar and delta chunks, will be stored as containers on the disk. The file mapping relationships among the duplicate chunks, resembling chunks, and non-similar chunks will also be recorded as the file recipes [28], [44] to facilitate future data restore operations in DARE.

For the restore operation, DARE will first read the referenced file recipes and then read the duplicate and non-similar chunks one by one from the referenced segments on disk according to mapping relationships in the file recipes. For the resembling chunks, DARE needs to read both delta data and base-chunks and then delta decode them to the original ones.

By exploiting the duplicate-adjacency information in resemblance detection and further improving the super-feature approach, DARE is able to maximize data reduction while reducing the overheads of resemblance detection in existing deduplication systems, as quantitatively demonstrated in Section 4.

4 Performance Evaluation

In order to evaluate DARE, we have implemented a prototype of DARE that allows us to examine several important design parameters to provide useful insights. We compare DARE with the latest super-feature approaches in terms of data reduction, computation & indexing overheads, data-reduction throughput, and restore speed.

4.1 Experimental Setup

Platform of the DARE prototype. We have implemented a prototype of DARE and tested it on the Ubuntu 12.04 operating system running on a quad-core Xeon E5606 processor at 2.13 GHz, with a 16 GB RAM, a 14 TB RAID6 disk array that consists of sixteen 1 TB disks, and a 120 GB SSD of KINGSTON SVP200S37A120G.

Configurations for data reduction. DARE employs the widely used Rabin algorithm [34], [43] and SHA-1 hash function [6] respectively for chunking and fingerprinting for data deduplication, and an average chunk size of 8KB. For the resemblance detection, DARE adopts a CDC sliding window size of 48 bytes to generate features and Xdelta [23] to compress the detected similar chunks.

Evaluation metrics. We evaluate DARE in the key metrics of data reduction, similarity degree, computation & indexing overheads, data-reduction throughput, and restore speed. Data reduction here is defined as the percentage of redundancy removed by deduplication and resemblance detection. The similarity degree of resemblance-detected

chunks is measured by the ratio of (compressed size) / (original size) as discussed in Section 3.4. The indexing overhead is measured by the number of indexing super-features (SF for short) stored in RAM, while the computation overhead is defined by the number of features computed for resemblance detection. Data-reduction throughput is measured by the rate at which datasets are processed, including deduplicating, detecting resemblance, and delta compressing. Restore speed is measured by the rate at each version of the datasets are restored, including reading the non-similar chunks, base-chunks, and delta decoding. It is worth noting that our evaluation testbed is not a production-quality deduplication system, but a research prototype. The results hence should be interpreted as of approximate and comparative in nature.

Datasets. Six well-known open-source projects [45], [46], [47] representing typical workloads of deduplication and resemblance detection are used in the evaluation of DARE as shown in Table 2. These datasets consist of large tarred files representing sets of source code files or objects concatenated together by backup software [8].

In order to test the scalability of DARE, we generate two larger synthetic backup datasets according to the principles of synthesizing datasets outlined in recent studies [41], [44]. We obtain the first version of the dataset from our research group of 16 users with 192 K files and totaling 42GB, mutate the data by the operations of "modify", "delete" and "new" for 20 percent, 1 percent, and 1 percent of the files respectively in the "Freq" dataset and 10, 0.5, and 0.5 percent of the files respectively in the "Less" dataset, and then concatenate individual files to the tarred files. The file modifications are also applied in the beginning, middle, and end of the files as suggested in [41]. The difference between our synthesizing scheme and the one proposed in [41] is that we mutate the dataset by file modifications on the real data, which more closely emulates the process by which similar/redundant data are generated, while their approach mutates on the fingerprint sets and re-generates random data by the mutated fingerprints.

RDB is a dataset collected from the Redis key-value store database [48]. The database has 5 million records, requiring 5 GB storage space. We ran YCSB [49] to update the database in a Zipfian distribution. The update rate is 1 percent on average. We backup the uncompressed dump.

TABLE 2
Workload Characteristics of Six Open-Source Project Datasets, Two Synthetic Backup Datasets, and One Database Dataset Used in the Performance Evaluation

Datasets	versions	Size
Emacs-21.4∼Emacs-23.4	8	1.15 GB
GDB-6.7∼GDB-7.4.1	10	1.37 GB
Glibc-2.1.1∼Glibc-2.15	35	3.18 GB
SciLab-5.0.1∼SciLab-5.3.2	10	4.94 GB
GCC-4.3.4∼GCC-4.7.0	20	8.91 GB
Linux-3.0.0∼Linux-3.0.39	40	16.8 GB
Freq (20 percent inc. of newer version)	20	857 GB
Less (10 percent inc. of newer version)	30	1372 GB
RDB (backups of Redis database)	100	540 GB

rdb files as the snapshots of the Redis database, and collect 100 backups, which represents a typical database workload for data reduction.

4.2 An Empirical Study of the Super-Feature Approach

We first examine the impact of the number of features per SF and the number of SFs used in resemblance detection via a real-world dataset driven evaluation, which helps answer question (1) of Section 4.1.

In Fig. 6, we plot the trend of data reduction of the traditional super-feature approach as a function of the number of features per SF **'N'** and the number of SFs **'M'**. We find a general tendency of redundancy elimination being a decreasing function of **'N'** and an increasing function of **'M'**, which is consistent with the results of our mathematical analysis illustrated in Fig. 4 in Section 3.3. That is, the more SFs are used, the more resemblance can be exposed to eliminate more redundancy. On the other hand, the more features used in generating each SF, the less redundancy will be eliminated, because the probability of more features in an SF being identical is smaller than that of fewer features in an SF.

Note that the redundancy elimination of Linux with one feature in an SF is less than that with two features in an SF. This is likely due to the repeated headers in the source files [12], which will be explained later in Fig. 7c. Nevertheless, in general, a smaller number of features per SF are shown to detect more resemblance on all of our datasets, with the exception on Linux.

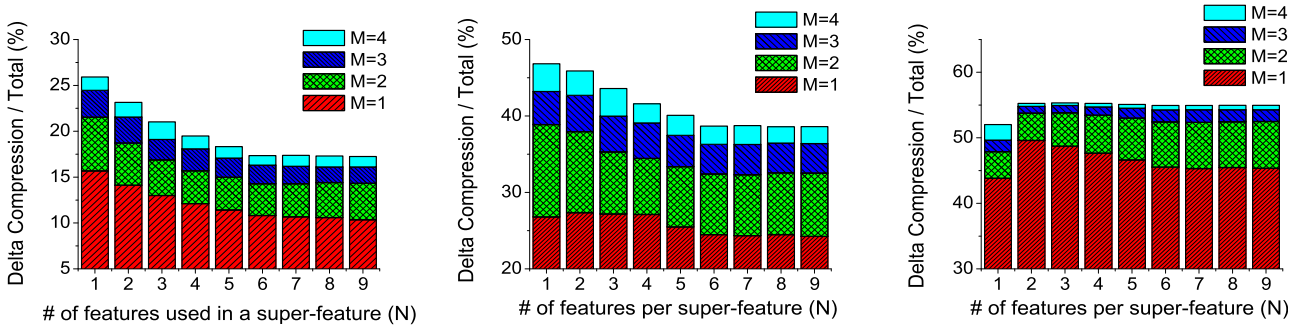


Fig. 6. Data reduction of the super-feature approach as a function of the number of features per super-feature and the number of super-features (M, shaded segments on each bar) on three typical datasets.

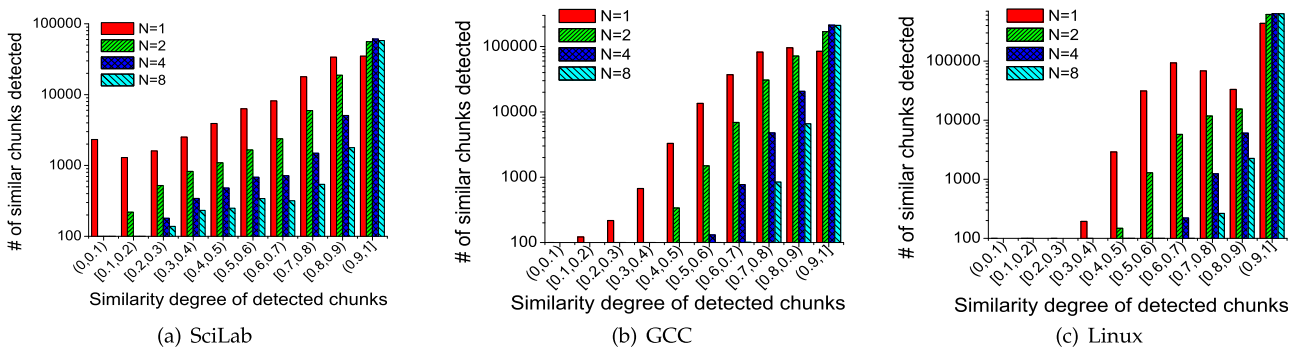


Fig. 7. Similarity degree distribution of detected chunks as a function of the number of features per SF (N, shaded bars in each similarity-degree range) on the three typical datasets.

To explain the reason why a smaller number of features per SF can achieve better data reduction efficiency, we show in Fig. 7 the similarity distribution of the super-feature approach. Here, the similar chunks are divided into 10 disjoint groups based on their similarity degree (defined in Section 3.4), with each group corresponding to one of the 10 ranges of (0, 0.1),...[0.9, 1.0]. Thus, the figure plots the number of similar chunks detected as a function of the similarity degree and the number of features per SF. Fig. 7 clearly and consistently shows that the number of similar chunks detected is a decreasing function of the number of features per SF across all the datasets, which validates our analytical results in Section 3.3. Fig. 7 also suggests that the false positive rate of resemblance detection for the SF with a smaller number of features in them is very low. In fact, less than 5 percent of the two-feature per SF detected chunks have a similarity degree smaller than 60 percent. Note that the SF with one feature tends to detect more similar chunks on the Linux dataset, which helps explain the data reduction result of the one-feature per SF in Fig. 6c for the Linux dataset.

To evaluate the overall performance impact of the number of features per SF, we plot in Fig. 8 the data-reduction throughput of the super-feature approaches running on the RAID as a function of the number of features per SF. We find that the one-feature-per-SF approach has a lower throughput than the two- or three-features-per-SF approaches where the highest throughputs are achieved. This is because the former detects more similar chunks than the latter and thus induces more random I/Os in reading

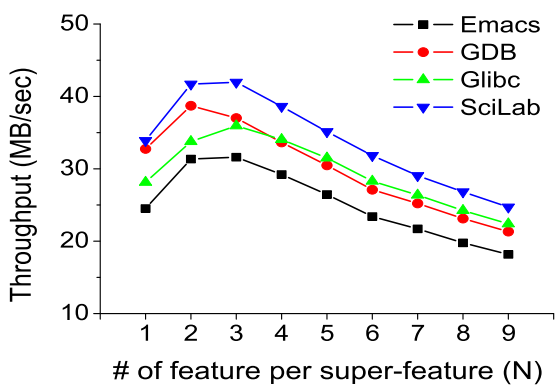


Fig. 8. Data-reduction throughput of the four super-features approach as a function of number of features per SF.

the base-chunks of the resemblance-detected chunks for delta compression. On the other hand, while approaches based on three or more features detect less resemblance, they incur higher computation latency (overhead), which lowers the throughput.

Fig. 9 shows the data reduction results of the deduplication and resemblance detection approaches as a function of the average chunk sizes. It demonstrates that resemblance detection is very efficient in supplementing deduplication for data reduction. The larger the average chunk size is, the less duplicate data are detected. But the resemblance detection approach can detect almost all the redundant data (e.g., similar data) that deduplication fails to identify, regardless of the average chunk size.

As the Linux dataset has very strong similarity, the two-feature and four-feature approaches perform almost identically in Fig. 9. But, generally speaking, the two-feature approach can detect more redundancy on our real-world datasets as shown in Fig. 6 and achieves the highest throughput as shown in Fig. 8. Therefore, we choose to use a low-overhead super-feature approach in DARE where there are fewer super-features and each super-feature has only two features.

4.3 Deduplication-Aware Resemblance Detection

In this section, we evaluate DARE's resemblance detection and elimination schemes, i.e., the DupAdj resemblance detection and the improved super-feature approach with three SFs and two features per SF., We compare DARE with the super-feature-only approaches based on three SFs with two-features per SF (SF-2F) and three SFs with four-features per SF (SF-4F). Note that *DARE's resemblance detection here is thus DupAdj supplemented by SF-2F (i.e.,* **DARE = DupAdj + SF-2F***)*, where SF-2F is applied only to chunks that DupAdj has failed to detect as being similar.

Table 3 shows the additional data reduction on top of the conventional deduplication (Dedupe) achieved by the four resemblance detection schemes, DupAdj, DARE, SF-2F, and SF-4F, on the six real-world datasets (both tarred and untarred). Generally, the DupAdj approach achieves a resemblance-detection efficiency similar to the SF-4F approach and DARE detects about 2-6 percent and 3-10 percent more redundancy than the SF-2F and SF-4F approaches respectively. As indicated in Fig. 7 and discussed in Section 4.2, SF-2F is more sensitive than SF-4F to the chunks with a lower similarity degree. Thus DARE detects the most

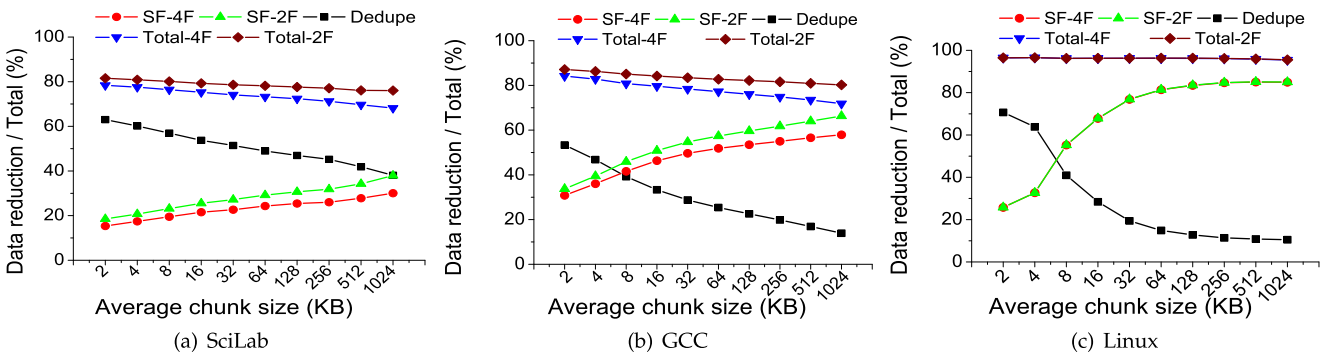


Fig. 9. Data reduction as a function of the average chunk size achieved by the approaches of conventional deduplication, the two-features-per-SF (SF-2F) and the four-features-per-SF (SF-4F). Total-2F and Total-4F refer to Dedupe + SF-2F and Dedupe + SF-4F respectively. Results of other datasets are similar to those of the GCC dataset and are omitted due to space limit.

TABLE 3
A Comparison among Deduplication, DupAdj, DARE, SF-2F, and SF-4F Approaches in the Data Reduction Measure under Six Real-World Datasets, Both Tarred and Untarred Versions for a Total of 12 Datasets

Datasets	Tarred						UnTarred					
	Emacs	GDB	Glibc	SciLab	GCC	Linux	Emacs	GDB	Glibc	SciLab	GCC	Linux
Versions	8	10	35	10	20	40	8	10	35	10	20	40
Dedupe	37.1%	48.7%	52.2%	56.9%	39.1%	40.9%	43.5%	70.6%	87.9%	77.5%	83.5%	96.7%
DupAdj	+32.1%	+33.5%	+29.2%	+19.5%	+38.2%	+53.4%	+29.6%	+10.9%	+2.9%	+5.2%	+7.2%	+0.7%
DARE	+41.0%	+40.8%	+36.9%	+25.2%	+46.7%	+54.1%	+37.7%	+18.2%	+7.3%	+10.4%	+9.9%	+1.0%
SF-2F	+33.7%	+36.4%	+35.3%	+22.6%	+45.2%	+54.4%	+31.7%	+16.5%	+6.6%	+9.6%	+9.1%	+0.9%
SF-4F	+28.2%	+33.4%	+30.4%	+18.8%	+40.6%	+53.5%	+28.0%	+14.2%	+5.7%	+8.1%	+7.3%	+0.6%

"+" denotes additional data reduction beyond deduplication.

resemblance by combining the DupAdj and SF-2F resemblance detection approaches.

Fig. 10 shows the similarity degree distribution of the resembling chunks detected by DupAdj, DARE, SF-2F, and SF-4F, which suggests that less than 1.2 and 2.8 percent of the DARE-detected chunks have a similarity degree smaller than 50 percent on the GDB and GCC datasets respectively. On the other hand, the SF-2F- and SF-4F-detected chunks tend to have a higher similarity degree, which means that the SF-2F and SF-4F approaches fail to detect the less resembling chunks and thus have a lower data reduction rate than DARE as shown in Table 3.

Fig. 11a shows that the average similarity degree of the resembling chunks detected by the DARE, SF-2F, and SF-4F approaches is about 0.890, 0.922, and 0.965 respectively. Meanwhile, the average similarity degree of the DupAdj-detected chunks is about 0.885 as suggested in Fig. 2b in Section 2.2. Therefore, the results shown in Figs. 2, 10, and 11a demonstrate that DupAdj and DARE are very effective and efficient in detecting resemblance among the post-deduplication chunks with a very low false-positive rate. In addition, the average similarity degree of DupAdj may be increased by appropriately adjusting the predefined resemblance detection threshold as discussed in Section 3.2.

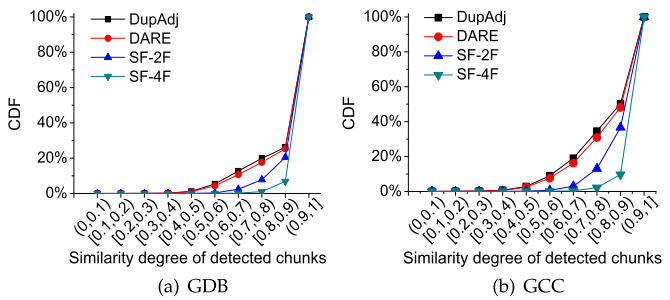
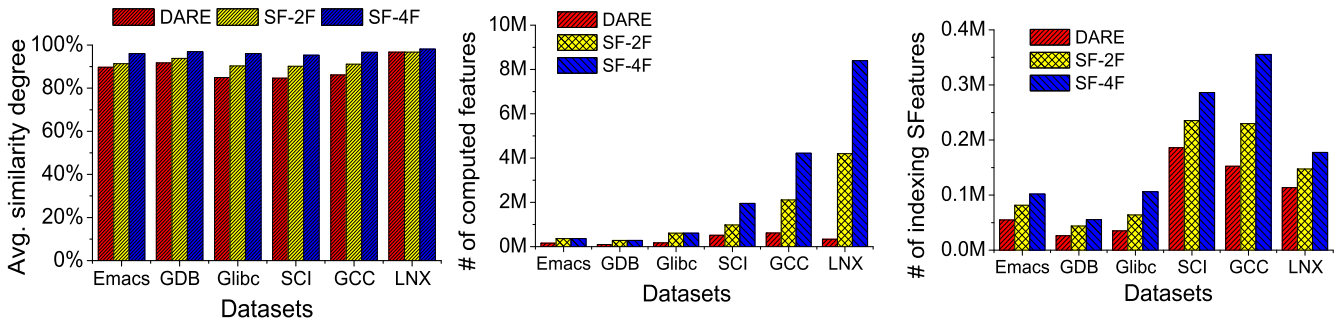


Fig. 10. CDF of the data chunks detected by the DupAdj, DARE, SF-2F, and SF-4F approaches as a function of the similarity degree on the two typical datasets.

Figs. 11b and 11c show the computation and indexing overheads incurred by the three resemblance detection schemes. Since the DupAdj approach only detects resemblance by exploiting the duplicate-adjacency information (with a doubly-linked list as shown in Fig. 3), here we only compare DARE with the SF-2F and SF-4F approaches for resemblance detection overheads. Obviously, SF-4F, which computes more features but detects less resemblance, consumes the most amounts of computation and indexing resources for resemblance detection. DARE uses the same super-feature parameters as SF-2F but incurs only half of the computation and indexing overheads of the SF-2F approach because of DupAdj's very effective pre-screening of similar chunks. In fact, DARE can further reduce the number of super-features while achieving a comparable resemblance detection efficiency to the SF-2F approach.

4.4 Scalability of DARE

In order to better evaluate the scalability of DARE on three larger datasets, we have implemented the schemes of stream-informed delta compression [12] in SiLo [29], a memory-efficient deduplication system that exploits the backup-stream similarity and locality. As introduced in Section 2.1, SIDC only detects resemblance in the



(a) Average similarity degree (b) Computation Overhead (c) Indexing Overhead

Fig. 11. A comparison among DARE, SF-2F, and SF-4F in terms of the similarity degree and the computation & indexing overheads.

backup-stream locality-preserved cache that can reduce the indexing overhead of SFs and scales well in large-scale deduplication system. Thus we employ their method to test the scalability of different resemblance detection schemes and implement SiLo with a 20 MB locality cache (similar to SIDC [12]) and a segment size of 1MB. The stream-informed approaches are denoted by the "Cached/Cac." prefix in Table 4 and Fig. 13.

Table 4 shows the data reduction results of different deduplication and resemblance detection schemes. Since SiLo achieves nearly 99 percent deduplication efficiency of the exact deduplication (i.e., full index in memory) while requiring substantially less memory overhead (about 1/250 of the exact deduplication approach) for fingerprint indexing [29], we only discuss the results of resemblance detection after SiLo's deduplication here. The first column under a dataset (Freq or Less or RDB) in Table 4 shows the percentages of data reduction and the second column shows the data reduction factor (the ratio of before/after data reduction).

As shown in Table 4, resemblance detection further reduces the storage space, after the deduplication process, by a factor of about 2.5, meaning that we can save about 60 percent of the post-deduplication storage space by resemblance-detecting post-deduplication chunks. DARE achieves a superior data reduction efficiency on both datasets while the cached SF-2F and SF-4F schemes detect less resemblance, which is consistent with the results on our real-world datasets summarized in Table 3. Table 4 also shows that the cached SF-2F and SF-4F schemes fail to detect a noticeable amount of resemblance due to the sometimes weak or lack thereof locality in the backup stream. The SIDC study also discussed this issue as the limitations of stream-informed cache [12]. We argue that DARE solves this problem reasonably well by detecting resemblance based on duplicate-adjacency in the stream-informed cache.

To further understand the scalability of DARE, Fig. 12 shows the percentages of resemblance detected by its DupAdj detection and by the SF-2F and SF-4F approaches showing the individual contributions by the first SF, the second SF, and the third SF, on the three backup datasets. DupAdj is very effective and efficient in detecting resemblance in the deduplication system with abundant duplicate-adjacency information (i.e., dedupe factor > 5), leaving DARE's improved super-feature approach very little (smaller than 1 percent), if any, additional resemblance to detect (see the DARE bars).

Fig. 13 shows that DARE achieves the highest throughputs among all the resemblance detection enhanced data reduction approaches compared running on both RAID-structured HDDs and the SSD. DARE achieve lower throughputs than the SiLo/D approach, which is because SiLo/D only does deduplication (i.e., Rabin-based chunking and SHA1-based fingerprinting) while DARE involves more computation tasks and I/Os by delta compression (i.e., resemblance detection, reading base chunks, and delta encoding). Cached SF-4F has the lowest throughput because it incurs the largest computation overhead for resemblance detection. It is noteworthy that DARE's average data-reduction throughput on RAID, at 50 MB/s, is much lower than DARE's average throughput of 85 MB/s on SSD. The root cause of RAID's inferior data-reduction performance (in Fig. 13a) mainly lies in the random reads of the base-chunks

TABLE 4
A Comparison among the Duplicate-Detection and Resemblance-Detection Approaches in Terms of Data Reduction Efficiency on Three Larger Backup Datasets Freq, Less, and RDB

Datasets	Freq		Less		RDB	
Dedupe	84.6%	6.5X	91.2%	11.4X	95.7%	23.2
SiLo/D	84.4%	6.4X	91.2%	11.4X	95.7%	23.2
Cac. DARE	+9.28%	*2.5X	+5.01%	*2.3X	+2.97%	*3.2X
Cac. SF-2F	+7.01%	*1.8X	+4.31%	*2.0X	+2.95%	*3.1X
Cac. SF-4F	+5.32%	*1.5X	+3.27%	*1.6X	+2.82%	*2.9X
Ful. SF-2F	+9.61%	*2.6X	+5.04%	*2.3X	+2.97%	*3.2X

The "+" ("") sign in front of a reduction percentage (factor) indicates the "additional" post-deduplication data reduction.*

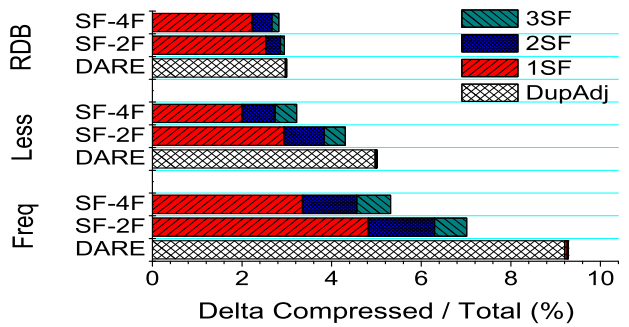


Fig. 12. Percentages of data reduced by DupAdj, and the first SF, second SF, third SF of the super-feature approach respectively in the stream-informed DARE, SF-2F, and SF-4F approaches.

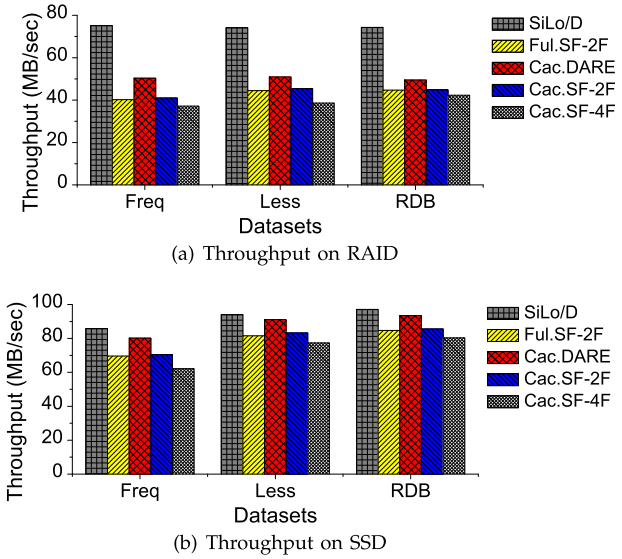


Fig. 13. Throughputs of four resemblance detection enhanced data reduction approaches (i.e., deduplication + delta compression) on the three backup datasets. The deduplication-only approach (i.e., SiLo/D) is also shown for comparison.

due to the lack or limitation of accessing locality [17]. This is the reason why DARE on SSD (in Fig. 13b) achieves a similar throughput to the deduplication-only approach (averaged 91 MB/s on SSD and 74 MB/s on RAID) while reducing more redundant data to be stored.

In general, DARE achieves a superior performance of both throughput and data reduction efficiency among all the resemblance detection enhanced data reduction approaches. Furthermore and importantly, our analysis of the execution times by the four data-reduction steps (see Fig. 5 and Section 3.5) based on data from Fig. 13 suggests that the average throughputs of resemblance detection on non-duplicate data of DARE, SF-2F, and SF-4F are 154, 60, and 30 MB/s respectively. The throughput results of SF-2F and SF-4F are consistent with the results shown in Fig. 4b. The main contributor to DARE's much higher resemblance-detection throughput is its DupAdj based resemblance detection scheme, as implied in Fig. 12. This suggests that DARE is shifting the bottleneck of data reduction from Step 2 to the other steps (e.g., Step 1 or 3), and the performance impact of this shifting will likely be much more pronounced when larger indexing cache is used and/or when the emerging storage class memory (SCM) devices (e.g., Nand Flash, PCM, STTRAM, etc.) replace or supplement HDDs.

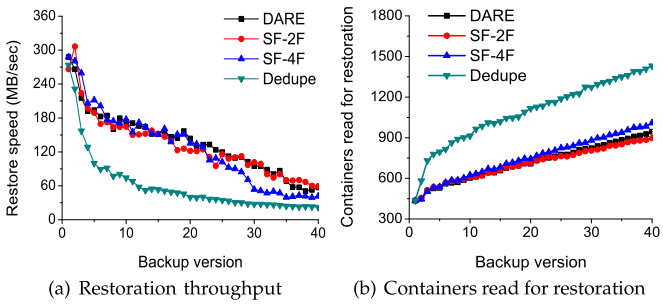


Fig. 14. Data-restore performance as a function of the backup version on the Linux dataset with an LRU cache of size 256 MB.

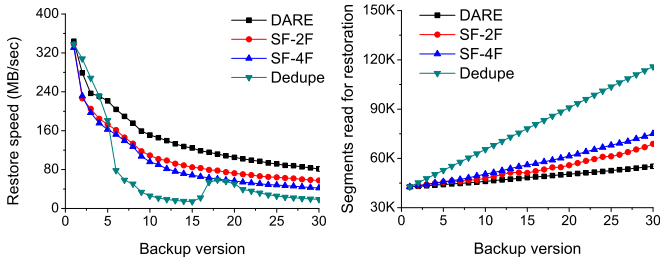


Fig. 15. Data-restore performance as a function of the backup version on the Less dataset with an LRU cache of size 512 MB. Results of other datasets are similar to the Less dataset and are omitted due to space limit.

4.5 Restoration Performance

Intuitively, delta compression should slow down the data-restore performance of a data-reduction system since it needs to restore the resembling chunks by two reads, one for the delta data and the other for the base-chunk, and then delta decode them. But in our evaluation of the restore operations for resembling chunks, we find that the speed of delta decode (i.e., Xdelta [23]) tends to be very fast, about 1 GB/s in the DARE system. Another interesting observation is that, for a restoration cache of a given size, DARE will effectively cache more logical content than a deduplication-only system because of DARE's additional enhanced delta compression, which has a space-efficient effect similar to Difference Engine [20] and I-CASH [18] that employ delta compression to enlarge the memory and SSD cache space respectively.

Figs. 14a and 15a show that DARE on average doubles the data-restore speed of the deduplication-only system (both running on the RAID). Figs. 14b and 15b clearly show that the reason lies in the fact that DARE reads half as many the containers for restoration as the deduplication-only system. The superior data-restore performance of DARE, SF-2F, and SF-4F to the deduplication-only system is attributed to their data reduction efficiency (see Tables 3 and 4). The sudden increase in the data-restore performance of the deduplication-only approach at the backup version 17 (Fig. 15a), we observe, is due to the fact that most of the backed-up sources targeted for restoration are from the current and recent backups and thus have fewer random reads for restoration.

Nevertheless, the trend in the evaluation results of Figs. 14 and 15 clearly indicates that in general the data-restore performance in data reduction systems decreases as a function of the number of backups (version), and DARE and other super-feature based approaches outperform the deduplication-only approach by further removing redundancy and thus effectively enlarging the restoration cache via delta compression. In addition, DARE achieves a better restore performance than other super-feature based approaches because of its higher compressibility by combining its proposed DupAdj and the improved super-feature schemes.

5 Conclusion and Future Work

In this paper, we present DARE, a deduplication-aware, low-overhead resemblance detection and elimination scheme for data reduction in backup/archiving storage

systems. DARE uses a novel approach, DupAdj, which exploits the duplicate-adjacency information for efficient resemblance detection in existing deduplication systems, and employs an improved super-feature approach to further detecting resemblance when the duplicate-adjacency information is lacking or limited.

Results from experiments driven by real-world and synthetic backup datasets suggest that DARE can be a powerful and efficient tool for maximizing data reduction by further detecting resembling data with low overheads. Specifically, DARE only consumes about 1/4 and 1/2 respectively of the computation and indexing overheads required by the traditional super-feature approaches while detecting 2-10 percent more redundancy and achieving a higher throughput. Furthermore, the DARE-enhanced data reduction approach is shown to be capable of improving the data-restore performance, speeding up the deduplication-only approach by a factor of 2(2X) by employing delta compression to further eliminate redundancy and effectively enlarge the logical space of the restoration cache.

Our preliminary results on the data-restore performance suggest that supplementing delta compression to deduplication can effectively enlarge the logical space of the restoration cache, but the data fragmentation in data reduction systems remains a serious problem. We plan to further study and improve the data-restore performance of storage systems based on deduplication and delta compression in our future work.

Acknowledgments

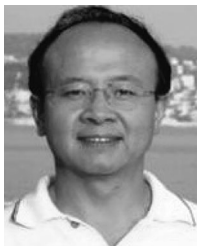
The authors are grateful to the anonymous reviewers for their insightful comments and constructive suggestions. The work was partly supported by Chinese 973 Program No. 2011CB302301; NSFC No. 61025008, 61173043, 61232004, and 61303046; 863 Project 2013AA013203; US National Science Foundation (NSF) under Grants CNS-111660, and CNS-1016609; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2014QNRC019. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China. The work of Lei Tian was done while he was working at the CSE Dept. of UNL. Dan Feng is the corresponding author. A 10-page conference version of this paper appeared in Proceedings of the 2014 Data Compression Conference (DCC'14). In this journal version, we included more substantial descriptions of research motivation & implementation on DARE and additional measurement results from our analysis and testbed experiments.

References

[1] The data deluge [Online]. Available: http://econ.st/fzkuDq
[2] J. Gantz and D. Reinsel, "Extracting value from chaos," *IDC Rev.*, vol. 1142, pp. 1–12, 2011.
[3] L. DuBois, M. Amaldas, and E. Sheppard, "Key considerations as deduplication evolves into primary storage," White Paper 223310, Framingham, MA, USA: IDC, Mar. 2011.
[4] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, "Single instance storage in windows 2000," in *Proc. 4th USENIX Windows Syst. Symp.*, Aug. 2000, pp. 13–24.
[5] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proc. USENIX Conf. File Storage Technol.*, Jan. 2002, pp. 89–101.
[6] 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.
[7] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Trans. Storage*, vol. 7, no. 4, p. 14, 2012.
[8] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 33–48.
[9] A. El-Shimi, R. Kalach, A. Kumar, A. Ottean, J. Li, and S. Sengupta, "Primary data deduplication-large scale study and system design," in *Proc. Conf. USENIX Annu. Tech. Conf.*, Jun. 2012, pp. 285–296.
[10] 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.
[11] 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.
[12] P. Shilane, M. Huang, G. Wallace, and W. Hsu, "WAN optimized replication of backup datasets using stream-informed delta compression," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, pp. 49–64.
[13] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: Virtual machine co-migration for the cloud," in *Proc. 20th Int. Symp. High Perform. Distrib. Comput.*, Jun. 2011, pp. 159–170.
[14] X. Zhang, Z. Huo, J. Ma, and D. Meng, "Exploiting data deduplication to accelerate live virtual machine migration," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2010, pp. 88–96.
[15] F. Douglis and A. Iyengar, "Application-specific delta-encoding via resemblance detection," in *Proc. USENIX Annu. Tech. Conf., General Track*, Jun. 2003, pp. 113–126.
[16] 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. 59–72.
[17] 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.
[18] Q. Yang and J. Ren, "I-cash: Intelligently coupled array of SSD and HDD," in *Proc. 17th IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 278–289.
[19] G. Wu and X. He, "Delta-FTL: Improving SSD lifetime via exploiting content locality," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, Apr. 2012, pp. 253–266.
[20] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference engine: Harnessing memory redundancy in virtual machines," in *Proc. 5th Symp. Oper. Syst. Design Implementation.*, Dec. 2008, pp. 309–322.
[21] 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.
[22] 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.
[23] J. MacDonald, "File system support for delta compression," Master's thesis, Dept. of Electr. Eng. Comput. Sci., Univ. California at Berkeley, Berkeley, CA, USA, 2000.
[24] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "Hydrastor: A scalable secondary storage," in *Proc. USENIX Conf. File Storage Technol.*, Feb. 2009, pp. 197–210.
[25] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. 7th USENIX Conf. File Storage Technol.*, Feb. 2009, pp. 111–123.
[26] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. T. Klein, "The design of a similarity based deduplication system," in *Proc. Israeli Experimental Syst. Conf.*, May 2009, pp. 1–12.
[27] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2011, pp. 271–284.
[28] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge, "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.

[29] 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.
[30] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," Hewlett Packard Labs., Palo Alto, CA, USA, Tech. Rep. HPL-2005-30(R.1), 2005.
[31] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *Proc. 7th USENIX Conf. File Storage Technol.*, 2010, p. 18.
[32] 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.
[33] 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.
[34] M. Rabin, "Fingerprinting by random polynomials," Center Res. Comput. Techn., Aiken Comput. Lab., Harvard Univ., Cambridge, MA, USA, 1981.
[35] 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.
[36] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, Y. Zhang, and Y. Tan, "Design tradeoffs for data deduplication performance in backup workloads," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 331–344.
[37] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proc. Israeli Experimental Syst. Conf.*, May 2009, pp. 13–24.
[38] A. Broder, "Identifying and filtering near-duplicate documents," in *Proc. 11th Annu. Symp. Combinatorial Pattern Matching*, Jun. 2000, pp. 1–10.
[39] A. Broder, "On the resemblance and containment of documents," in *Proc. Compression Complexity Sequences.*, Jun. 1997, pp. 21–29.
[40] U. Manber, et al., "Finding similar files in a large file system," in *Proc. USENIX Winter Tech. Conf.*, Jan. 1994, pp. 1–10.
[41] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Generating realistic datasets for deduplication analysis," in *Proc. USENIX Conf. Annu. Tech. Conf.*, Jun. 2012, pp. 261–272.
[42] D. Meister, J. Kaiser, and A. Brinkmann, "Block locality caching for data deduplication," in *Proc. 6th Int. Syst. Storage Conf.*, 2013, pp. 1–12.
[43] A. Broder, "Some applications of Rabin's fingerprinting method," in *Sequences II: Methods in Communications, Security, and Computer Science*. New York, NY, USA: Springer, 1993, pp. 1–10.
[44] 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.
[45] GNU software archive [Online]. Available: http://ftp.gnu.org/gnu/
[46] Scilab archive [Online]. Available: ftp://ftp.kernel.org
[47] Linux archive [Online]. Available: ftp://ftp.kernel.org
[48] Redis key-value database [Online]. Available: http://redis.io/
[49] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 143–154.

Wen Xia received the PhD degree in computer science from the 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, delta compression, storage systems, and cloud storage. He publishes more than 10 papers in major journals and international conferences including IEEE TC, USENIX ATC, USENIX FAST, INFOCOM, IFIP Performance, IEEE DCC, MSST, etc. He is a member of the IEEE.



Hong Jiang received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto, Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, Texas, in 1991. Since August 1991, he has been at the University of Nebraska-Lincoln, Lincoln, NE, where he is a Willa Cather professor of computer science and engineering. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, performance evaluation. He has more than 200 publications in major journals and international Conferences in these areas, including *IEEE-TC, IEEE-TPDS, ACM-TACO, JPDC*, ISCA, MICRO, USENIX ATC, FAST, LISA, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, ICPP. He is a fellow of the IEEE and a member of the ACM.

Dan Feng received the BE, ME, and PhD degrees in computer science and technology in 1991, 1994, and 1997, respectively, from the Huazhong University of Science and Technology (HUST), China. She is a professor and vice 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-TC, IEEE-TPDS, ACM-TOS, JCST*, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. She serves as the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012, 2015. She is a member of the IEEE and a member of ACM.

Lei Tian received the PhD degree in computer engineering from the Huazhong University of Science and Technology (HUST) in 2010. His research interests mainly lie in storage systems, distributed systems, cloud computing, and big data. He has more than 30 publications in major journals and conferences including FAST, HOTSTORAGE, ICS, SC, HPDC, ICDCS, MSST, MASCOTS, ICPP, IPDPS, CLUSTER, *IEEE TC, IEEE TPDS, ACM TOS*, etc. He is a senior member of the IEEE, and a member of the ACM and USENIX.