A Simulation Analysis of Redundancy and Reliability in Primary Storage Deduplication

Min Fu, Shujie Han, Patrick P. C. Lee, Dan Feng, Zuoning Chen, Yu Xiao

Abstract—Deduplication has been widely used to improve storage efficiency in modern primary and secondary storage systems, yet how deduplication fundamentally affects storage system reliability remains debatable. This paper aims to analyze and compare storage system reliability with and without deduplication in primary workloads using public file system snapshots from two research groups. We first study the redundancy characteristics of the file system snapshots. We then propose a trace-driven, deduplication-aware simulation framework to analyze data loss in both chunk and file levels due to sector errors and whole-disk failures. Compared to without deduplication, our analysis shows that deduplication consistently reduces the damage of sector errors due to intra-file redundancy elimination, but potentially increases the damages of whole-disk failures if the highly referenced chunks are not carefully placed on disk. To improve reliability, we examine a deliberate copy technique that stores and repairs first the most referenced chunks in a small dedicated physical area (e.g., 1% of the physical capacity), and demonstrate its effectiveness through our simulation framework.

Index Terms—deduplication, reliability, primary storage systems, experiments and implementation

1 Introduction

Modern storage systems adopt *deduplication* to achieve storage savings, by referencing data copies with identical content to the same physical copy in order to eliminate storage redundancy. Deduplication has been widely adopted in secondary storage (e.g., backup and archival) [10], [23], [44]; recently, it has also been studied and deployed in primary storage (e.g., file systems) [9], [11], [31], [40], [43]. Despite the wide adoption, how deduplication affects storage system reliability remains debatable when compared to without deduplication. On one hand, deduplication mitigates the possibility of data loss by reducing storage footprints (assuming that data loss events equally occur across the entire disk space); on the other hand, it amplifies the severity of each data loss event, which may corrupt multiple chunks or files that share the same lost data.

A number of studies in the literature have addressed deduplication storage reliability in different ways. For example, some studies (e.g., [6], [10], [25]) add redundancy via replication or erasure coding to post-deduplication data for fault tolerance. Other studies (e.g., [20], [35], [36]) propose quantitative methods to evaluate deduplication storage reliability. However, there remain two key open reliability issues, which are further complicated by the data sharing nature of deduplication.

- **Loss variations:** Storage systems are susceptible to both device failures and latent sector errors, yet they incur different amounts of data loss. Also, the impact of data loss depends on how we define the granularities of storage (e.g., a chunk or a file with multiple chunks). Thus, the actual impact of data loss can vary substantially.
- **Repair strategies:** The importance of data in deduplication varies, as each data copy may be shared by a different number of other copies. When a storage system experiences failures, its repair strategies determine whether important data copies are repaired first, and hence affect reliability in different ways.

Our work is motivated by the importance of analyzing and comparing storage system reliability with and without deduplication. Traditional reliability analysis often uses the Mean Time to Data Loss (MTTDL) metric to characterize storage system reliability. MTTDL assumes independent exponential distributions of failure and repair events, and its closed-form solution can be obtained from Markov modeling. Its popularity is mainly attributed to its simplicity of modeling the reliability of a wide variety of system configurations. On the other hand, some studies [12], [13], [17] have shown that MTTDL is inaccurate for reliability analysis, due to its over-simplistic assumptions in modeling the actual failure nature of real-world storage systems. In deduplication storage, we conjecture that MTTDL is inappropriate for its reliability analysis, due to the varying severity of data loss. Thus, we advocate *simulation* for accurate reliability analysis, at the expense of intensive computations [12].

In this paper, we conduct redundancy and reliability analysis on primary storage deduplication, which is less explored than secondary storage deduplication but has received increasing attention. Specifically, we examine public datasets of real-life file system snapshots collected by two different research groups, including nine Mac OS or Linux file system snapshots from the File system and Storage

- *M. Fu, D. Feng and Y. Xiao are with Wuhan National Lab for Optoelectronics School of Computer, Huazhong University of Science and Technology, Wuhan, China (*{fumin, dfeng, yuxiao}*@hust.edu.cn).*
- *S. Han and P. Lee are with the Chinese University of Hong Kong, Shatin, N.T., Hong Kong (*{sjhan, pclee}*@cse.cuhk.edu.hk).*
- *Z. Chen is with National Engineering Research Center for Parallel Computer, Beijing, China (*chenzuoning@vip.tom.com*).*
- *An earlier conference version of this paper appeared at 2016 IEEE International Symposium on Workload Characterization (IISWC 2016) [15]. In this journal version, we include additional analysis results of a much larger-scale dataset and redundancy characteristics of primary storage deduplication.*

Lab (FSL) at Stony Brook University [2] and 903 Windows file system snapshots from Microsoft [30]. We make the following contributions.

First, we study the redundancy characteristics of the file system snapshots from two aspects: the reference counts of chunks and the redundancy sources of duplicate chunks. We observe that most chunks are referenced only once or twice, but there exist a few extremely popular chunks. Also, intra-file redundancy, duplicate files, and similar files are the major sources of duplicate chunks. Our redundancy study provides insights into our following reliability analysis.

Second, we propose a trace-driven, deduplication-aware simulation framework to analyze and compare storage system reliability with and without deduplication. Specifically, we start with a RAID disk array setting, and extend the notion of NOrmalized Magnitude of Data Loss (NOMDL) [17] to define new reliability metrics for deduplication storage. Our simulation framework takes file system snapshots as inputs, and performs Monte Carlo simulation to analyze the loss impact in both chunk and file levels due to uncorrectable sector errors and unrecoverable disk failures. Our reliability study enables us to identify any possible solution to improve storage system reliability should deduplication be deployed.

Third, we apply our simulation framework and show the following key findings of our reliability analysis:

- Compared to without deduplication, deduplication does not change the expected amounts of corrupted chunks caused by uncorrectable sector errors, and it consistently reduces the expected amounts of corrupted files due to intra-file redundancy elimination. Thus, individual chunk corruptions caused by uncorrectable sector errors do not pose extra vulnerability concerns under deduplication.
- On the other hand, the impact of unrecoverable disk failures is highly related to chunk fragmentation caused by deduplication [22] and disk repair operations. If the highly referenced chunks are neither carefully placed nor preferentially repaired, the amounts of corrupted chunks and files can significantly increase under deduplication.
- We observe that highly referenced chunks occupy a large fraction of logical capacity, but only a small fraction of physical capacity after deduplication. To reduce the significance of unrecoverable disk failures, we explore a deliberate copy technique that allocates a small dedicated physical area (with only 1% of physical capacity) for the most referenced chunks and first repairs the physical area during RAID reconstruction. Our simulation results show that the technique can significantly reduce the expected amounts of corrupted chunks and files, while incurring only limited storage overhead.

The source code of our simulation framework is available at **http://adslab.cse.cuhk.edu.hk/software/simdedup**. The datasets that we use are publicly available and can be verified with our simulation framework.

The rest of the paper proceeds as follows. Section 2 presents the background and related work. Section 3 describes the datasets for our simulation study. Section 4 analyzes the redundancy characteristics of the datasets. Section 5 presents the design of our simulation framework. Section 6 presents our simulation results of our reliability analysis. Finally, Section 7 concludes the paper.

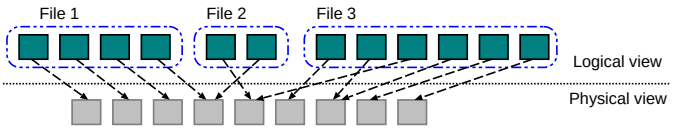


Fig. 1. Logical and physical views of a deduplication system.

2 Background and Related Work

2.1 Deduplication Basics

Deduplication is a technique that reduces storage space by eliminating content redundancy. Practical deduplication often operates at the granularity of non-overlapping data units called *chunks*, each of which is identified by a *fingerprint* formed by the cryptographic hash (e.g., SHA-1) of the chunk content. Deduplication treats two chunks with the same (resp. different) fingerprint as duplicate (resp. unique) chunks, and the probability of having two unique chunks with the same fingerprint is practically negligible [33]. It keeps only one copy of the chunk in storage, and refers other duplicate chunks to the copy via small-size references.

Deduplication performs *chunking* to divide data into fixed-size chunks or variable-size content-defined chunks. Fixed-size chunking is mostly used for high computational performance. On the other hand, variable-size chunking defines chunk boundaries by content so as to be robust against content shifts, and generally achieves higher deduplication efficiency than fixed-size chunking. Variable-size chunking can be implemented by Rabin Fingerprinting [34], which computes a rolling hash over a sliding window of file data and identifies boundaries whose rolling hashes match some target pattern. To effectively remove duplicate chunks, the average chunk size is typically on the order of kilobytes (e.g., 8KB [44]).

A deduplication system keeps fingerprints of all stored chunks in a key-value store called the *fingerprint index*. For each input chunk, the system checks by fingerprint if a duplicate chunk has been stored, and stores only unique chunks. For each file, the system also stores a *file recipe*, which lists the references to all chunks of the file for file reconstruction.

In deduplication storage, we need to differentiate the *logical* and *physical* views, which describe the storage organizations before and after deduplication, respectively. For example, referring to Figure 1, the logical view shows three files with a total of 12 chunks, while the physical view shows only nine chunks that are actually stored. From a reliability perspective, the logical and physical views of a deduplication system have different implications of data loss, which we aim to analyze in this work.

2.2 Related Work

Many measurement studies focus on characterizing the storage efficiency of deduplication for both primary and secondary storage environments. For example, Jin *et al.* [19] and Jayaram *et al.* [18] show that deduplication effectively reduces the storage of virtual machine disk images, even with fixed-size chunking. Meyer *et al.* [30] analyze hundreds of Windows file system snapshots at Microsoft, and show that file-level deduplication can eliminate content redundancy as effectively as chunk-level deduplication. Lu *et al.*

TABLE 1
Statistics of file system snapshots in the FSL dataset.

Snapshot	OS	Date	Raw Size (GB)	# Files	# Chunks	Size Reduction (%)
Mac	OS X	01/01/2013	224.55	1,486,819	28,162,208	33.8%
U11	Linux	01/12/2011	289.86	2,457,630	33,726,865	36.0%
U12	Linux	21/05/2013	251.01	44,129	26,407,044	64.6%
U14	Linux	19/04/2012	161.19	1,339,088	16,707,076	61.1%
U15	Linux	17/04/2013	202.10	310,282	23,280,718	49.6%
U20	Linux	15/12/2011	592.73	836,974	47,884,281	79.8%
U21	Linux	29/03/2012	140.50	63,451	14,291,544	56.7%
U24	Linux	20/12/2011	168.70	212,939	20,657,959	24.4%
U26	Linux	31/03/2014	154.24	88,050	16,435,825	33.3%

[26] propose different techniques on improving deduplication effectiveness in primary storage. Wallace *et al.* [42] analyze over 10,000 EMC Data Domain backup systems, and observe that deduplication is essential for achieving high write throughput and scalability. Meister *et al.* [29] analyze four HPC centers and observe that deduplication can achieve 20-30% of storage savings. Sun *et al.* [41] focus on individual user data over 2.5 years and analyze their deduplication patterns.

In terms of storage system reliability, some measurement studies investigate the failure patterns of disk-based storage systems in production environments, such as whole-disk failures [32], [38] and latent sector errors [4], [37]. On the other hand, there are only limited studies on analyzing the reliability of deduplication systems. Most studies propose to improve reliability of deduplication systems through controlled redundancy, either by replication [6] or erasure coding [10], [20], [25], but they do not analyze the reliability affected by deduplication. Li *et al.* [20] propose combinatorial analysis to evaluate the probability of data loss of deduplication systems. Rozier *et al.* [35], [36] propose automata-based frameworks to quantitatively evaluate the reliability of deduplication systems under disk failures and sector errors. Our work complements the above studies by: (i) adopting more robust reliability metrics, (ii) focusing on primary storage workloads, and (iii) comparing the impact of loss variations and repair strategies on storage system reliability with and without deduplication.

3 Datasets

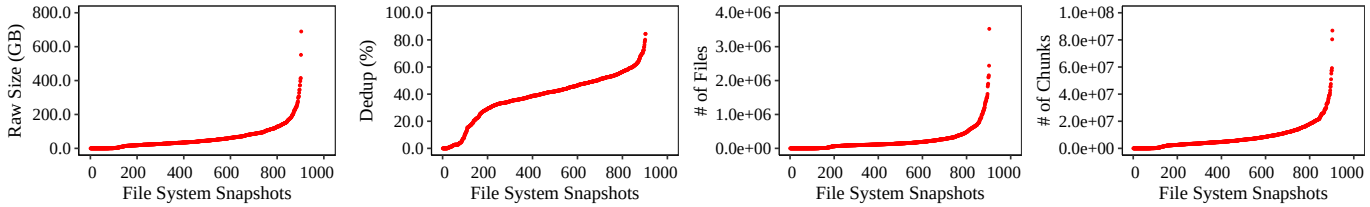
Our analysis focuses on primary storage deduplication, in which we consider public real-world file system snapshots collected by two different research groups. Both datasets also correspond to different types of operating systems. Due to privacy concerns, both datasets only contain chunk fingerprints but not the chunk contents.

The first dataset, which we refer to as *FSL*, consists of nine file system snapshots collected by the File system and Storage Lab (FSL) at Stony Brook University [2]. The original repository has hundreds of file system snapshots that span three years, but our analysis focuses on the ones whose sizes are sufficiently large for generating meaningful statistical distributions. Specifically, we pick nine random snapshots with raw size at least 100GB each. One of the snapshots, denoted by Mac, is taken from a Mac OS X server that hosts server applications (e.g., SMTP, Mailman, HTTP, MySQL, etc.); the other eight snapshots, denoted by U11–U26, are taken from different users' home directories with various types of files (e.g., documents, source code, binaries, virtual disk images, etc.). Here, U11 refers to a snapshot of user 011 in the FSL repository, and the same meanings hold for other users' snapshots. Each selected snapshot lists the 48-bit truncated MD5 fingerprints and the chunk sizes of all chunks, obtained from Rabin fingerprinting with the average, minimum, and maximum chunk sizes configured as 8KB, 2KB, and 16KB, respectively. While the short fingerprint length implies a high collision rate that is inadequate for real deployment, the collision rate remains small and suffices for analysis, as pointed out by the dataset owners [41]. Table 1 summarizes the statistics of each snapshot, including the snapshot name, OS, collection date, raw data size before deduplication, number of files, number of chunks, and percentage of reduction of storage size after deduplication (a larger percentage implies deduplication is more effective in terms of storage saving).

The second dataset, which we refer to as *MS*, is collected at Microsoft [30] and publicized on SNIA [1]. The original repository contains Windows file system snapshots that span 8 weeks from September 5 to October 31, 2009. Each file system snapshot includes system settings (e.g., hardware and software configurations), file metadata (e.g., timestamps, path, file name, file name extension, and attribute flags), and the fingerprints of all chunks of different chunk sizes obtained from Rabin fingerprinting. In our MS dataset, we focus on a total of 903 file system snapshots that are collected in a single week (the week of September 18, 2009) and configured with the average chunk size of 8KB.

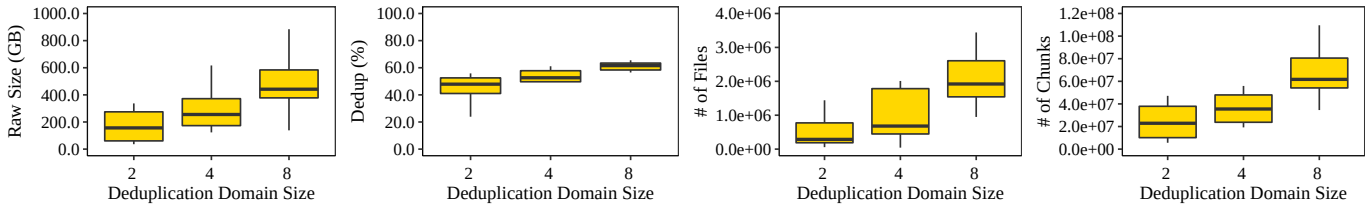
Figure 2 shows the statistics of the file system snapshots in the MS dataset, where the x-axis refers to each file system snapshot sorted by the y-axis value in ascending order. To summarize, the raw sizes of the snapshots range from 20.0KB to 689.7GB, among which 17.2% of them are of more than 100GB (Figure 2(a)). The reductions of storage size after deduplication have an average of 38.2% and range from 0% to 84.5%, among which 74.9% of them have 30-70% of storage savings (Figure 2(b)). Also, 95% of file system snapshots have fewer than 1.5 million files (Figure 2(c)) and fewer than 30 million chunks (Figure 2(d)). The statistics are fairly consistent with those of the FSL dataset.

In addition to studying individual file system snapshots, we also consider the aggregates of multiple file system snapshots in the MS dataset. Motivated by [30], we introduce the notion of a *deduplication domain*, which represents a set of file system snapshots over which we perform deduplication,



(a) Raw size (GB) (b) Dedup (%) (c) Number of files (d) Number of chunks

Fig. 2. Statistics of individual file system snapshots in the MS dataset (the x-axis is sorted by the y-axis values in ascending order).



(a) Raw size (GB) (b) Dedup (%) (c) Number of files (d) Number of chunks

Fig. 3. Boxplots of different deduplication domain sizes in the MS dataset.

while the duplicate chunks across different deduplication domains are considered to be unique. The deduplication domain size specifies the number of file system snapshots included in a deduplication domain. We mainly focus on the deduplication domain sizes equal to one, two, four, and eight; note that when the size is one, it is equivalent to considering individual file system snapshots. When the deduplication domain size is greater than one, instead of enumerating all possible sets of file system snapshots, we follow the approach in [30], in which we generate 10 random deduplication domains for each deduplication domain size, such that each deduplication domain includes a number (given by the deduplication domain size) of file system snapshots that are randomly selected from the dataset. Note that the file system snapshots selected in each deduplication domain may belong to the same or different users. As reported in [30], the standard deviations for the measurement results when using 10 random deduplication domains are very small, so we do not consider more deduplication domains in our following analysis. Figure 3 shows the boxplots[1] of our generated deduplication domains for different domain sizes larger than one. In particular, the storage size reduction after deduplication increases with the domain size (see Figure 3(b)), for example, from 45.7% to 60.3% on average when the deduplication domain size increases from two to eight. The reason is that more duplicate chunks are found and deduplicated across multiple file system snapshots.

4 Redundancy Analysis

We analyze the redundancy characteristics of our datasets under deduplication. Our goal is to provide insights into our subsequent reliability analysis.

1. A boxplot shows the minimum, lower quartile, median, upper quartile, maximum, as well as outliers, of collected samples. In our case, when the deduplication domain size is one, the collected samples refer to the 903 file system snapshots; when the deduplication domain size is greater than one, the collected samples are the 10 randomly generated deduplication domains.

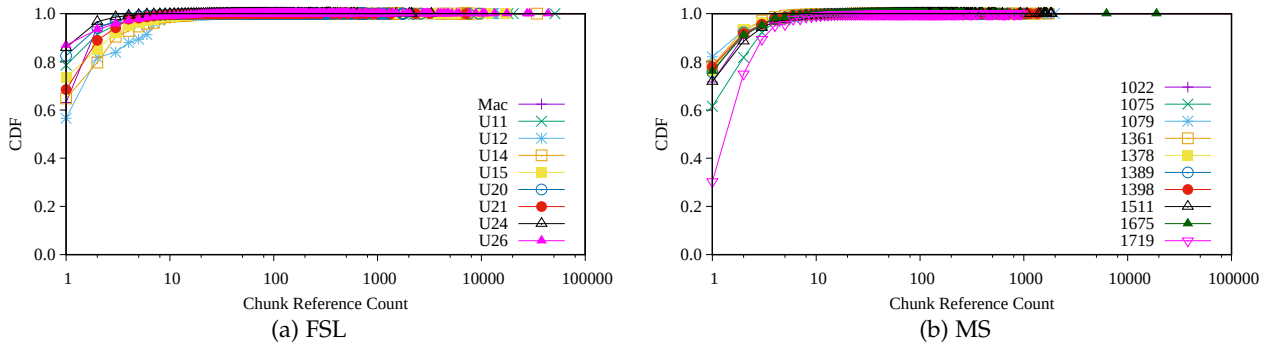
4.1 Reference Counts

We first analyze the distributions of chunk reference counts, based on our intuition that the importance of a chunk is proportional to its reference count [35]. Figure 4 shows the distributions of chunk reference counts in both FSL and MS datasets; for the MS dataset, we only plot the results of 10 file system snapshots with median raw sizes, ranging 44.69GB to 45.81GB (see Figure 2(a)). We observe that both datasets have similar distributions of chunk reference counts. First, the majority of chunks have small reference counts. For example, 56.5–86.9% and 30.6–82.1% of the chunks are referenced by exactly once, while 79.7–96.7% and 61.8–95.3% of the chunks are referenced by at most twice, in the FSL and MS datasets, respectively. However, there exist a few highly referenced chunks in both datasets. For example, in the FSL dataset, the Mac snapshot has the maximum reference count equal to 26,395, and U20 even has the maximum reference count equal to 28,402,757. In the MS dataset, if we examine all its 903 file system snapshots, we find that 79.6% of them have the maximum reference count at least 3 million, while one of them has the maximum reference count even equal to 28,403,618. The implication is that losing the highly referenced chunks may lead to severe loss of information as well as high deviations in the reliability simulations.

4.2 Redundancy Sources

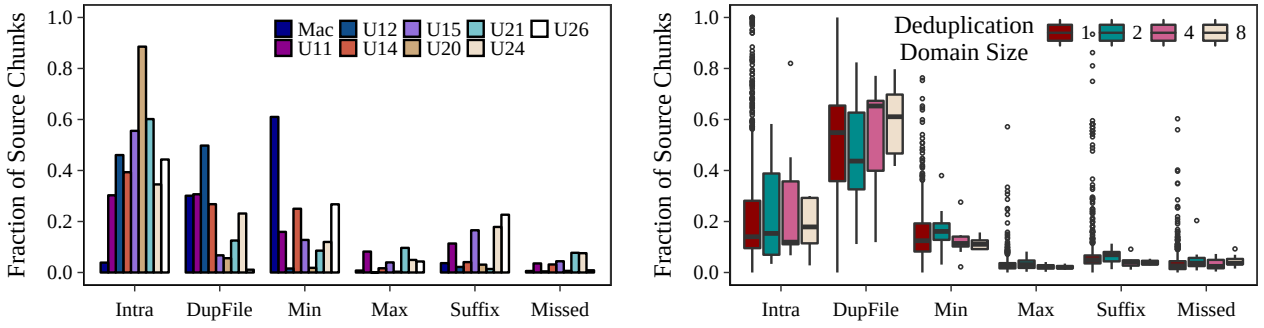
We now study the redundancy sources of duplicate chunks. Specifically, for each input chunk, the deduplication process checks if there exists a duplicate chunk that has already been stored (called the *source chunk*). Here, we consider the following six distinct types of source chunks:

- **Intra-file redundancy (Intra):** It means that both the input chunk and the source chunk belong to the same file.
- **Duplicate files (DupFile):** It means that the input chunk and the source chunk belong to different copies of files with the same content.
- **Min:** It means that the input chunk and the source chunk are stored in different files that share the same minimum



(a) FSL (b) MS

Fig. 4. Distributions of chunk reference counts in both FSL and MS datasets.



(a) FSL (b) MS

Fig. 5. Fractions of source chunks in both FSL and MS datasets.

chunk fingerprint. This implies that the two files are similar and likely to share a large proportion of duplicate chunks due to Broder's theorem [5], [8].

- **Max:** It means that the input chunk and the source chunk are stored in different files that do not belong to **Min** but share the maximum chunk fingerprint. It provides an additional indicator if the two files are similar.
- **Suffix:** It means that the input chunk and the source chunk are stored in different files that do not belong to **Min** nor **Max** but have the same extension. Again, it provides an additional indicator if the two files are similar.
- **Missed:** It means that the input chunk and the source chunk are stored in different files that do not belong to any of the above types.

Figure 5 shows the fractions of source chunks over all file system snapshots in both FSL and MS datasets. We make the following observations for different types of redundancy sources.

First, there are significant fractions of intra-file redundancy, especially in the FSL dataset. All file system snapshots, except Mac, have at least 30% of source chunks from intra-file redundancy; in particular, U20 has 88.6% of such references. Thus, if we directly use the reference count to quantify the reliability importance of a chunk, it may be inaccurate as losing a highly referenced chunk does not necessarily imply significant file corruptions. We also examine the file types and find that virtual disk images and package files are the major contributors to intra-file redundancy in the FSL dataset. In the MS dataset, the median fractions of references from intra-file redundancy are 11.9–17.9% for different deduplication domain sizes, while the fraction can go to almost 100% in some outlier file system snapshots.

Duplicate files are the most common redundancy source in the MS dataset, and the results are consistent for different deduplication domain sizes. This implies that whole-file deduplication is effective, as also confirmed by Meyer *et al.* [30]. According to [30], the most popular file extensions of duplicate files are .dll, .lib, .pdb, empty suffix, .exe, etc. Duplicate files are also common in the FSL dataset. For example, Mac has 30% of references from duplicate files, while U12 has even 49.8% of such references.

The fraction of source chunks of type **Min** is also high in both FSL and MS datasets, implying that the minimum fingerprints of files can be effectively used as the indicators to find duplicate chunks across similar files [5], [8]. In the FSL dataset, all file system snapshots, except U12 and U20, have 5.7–61.0% of source chunks belonging to the **Min** type. In the MS dataset, a median of 12.5% of source chunks belong to the **Min** type when the deduplication domain size is one, while the fractions are 3.1–38.0%, 2.2–27.6%, and 8.9–15.7% when the deduplication domain sizes are two, four, and eight, respectively.

We also check the fractions of source chunks that belong to types **Max** or **Suffix**, both of which can provide additional indicators whether duplicate chunks belong to similar files. We observe that the effects are marginal. In the FSL dataset, 11.3–22.7% of source chunks belong to types **Max** or **Suffix**, while in the MS datasets, the fraction of such source chunks is less than 10% except for some outlier cases. In general, the fraction of type **Missed** is less than 10% in both FSL and MS datasets.

We further study the sizes of the files to which the source

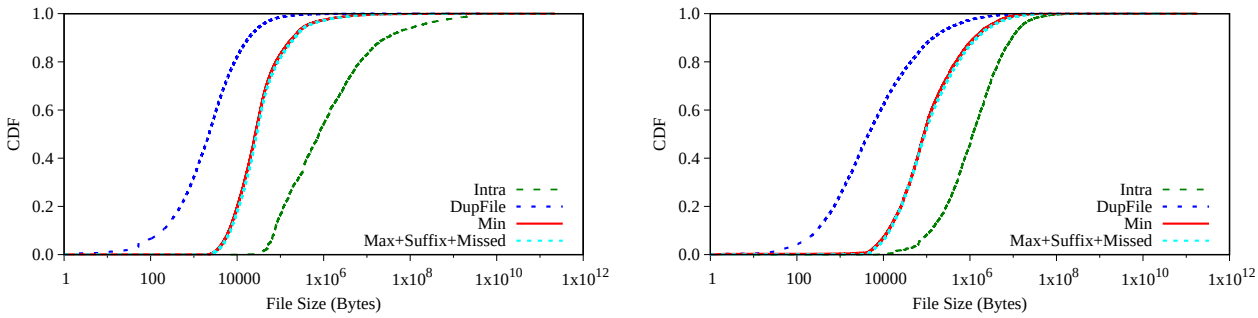


Fig. 6. Cumulative file size distributions of source chunks in both FSL and MS datasets (for the MS dataset, the deduplication domain size is one).

chunks belong. The file size distribution provides insights into how many bytes of a file are affected in the loss of a duplicate chunk. Figure 6 shows the cumulative file size distributions of different types of source chunks in both FSL and MS datasets (for the MS dataset, we only plot the results for the deduplication domain size equal to one, while the results are similar for the deduplication domain size greater than one). First, intra-file redundancy generally comes from large files, whose median file sizes are 757.8KB and 1.1MB in the FSL and MS datasets, respectively. On the other hand, duplicate files generally have small sizes, whose median file sizes are 2.2KB and 4.5KB in the FSL and MS datasets, respectively. Furthermore, the file sizes for type **Min** and the remaining types **Max**, **Suffix**, and **Missed** are similar, in which the median file sizes are 25.8KB (27.6KB for the remaining types) and 84.6KB (89.0KB for the remaining types) in the FSL and MS datasets, respectively. They may belong to the same kind of files, but just fail in the comparisons of minimum chunk fingerprints.

4.3 Summary

We summarize the key findings in our redundancy analysis:

- The majority of chunks in both FSL and MS datasets have small reference counts, while a few of them have extremely large numbers of reference counts. Losing highly reference chunks can imply the significant degradation of chunk-level reliability.
- Intra-file redundancy, duplicate files, and similar files sharing the same minimum chunk fingerprint are the major sources of duplicate chunks. For file system snapshots in which intra-file redundancy is dominant, losing a chunk may not necessarily imply the corruptions of many files.
- In general, files with intra-file redundancy are of large size, while duplicate files are of small size. The loss of a duplicate chunk may imply different amounts of bytes of a file being affected.

5 Simulation Framework

In this section, we design a simulation framework which analyzes and compares storage system reliability with and without deduplication. Our simulation framework builds on the High-Fidelity Reliability Simulator (HFRS) [16] and specifically addresses deduplication.

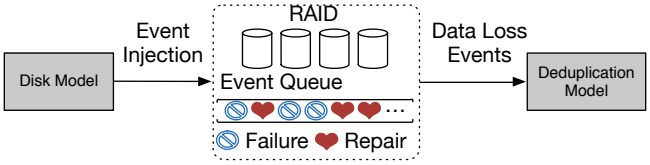


Fig. 7. Architecture of our simulation framework.

5.1 Architectural Overview

Figure 7 shows the architecture of our simulation framework. The framework targets primary storage deduplication for file system snapshots under a disk-based RAID setting. Specifically, it takes a file system snapshot or an aggregate of multiple file system snapshots (depending on the deduplication domain size), failure and repair distributions, and a system mission time (e.g., 10 years) as inputs. The *disk model* injects both failure events (including whole-disk failures and latent sector errors) and repair events to the simulated RAID array. Then the *event queue* sorts the failure and repair events in chronological order, and keeps only the events that stay within the system mission time. If a failure event incurs any data loss, it will trigger a data loss event to the *deduplication model*, which performs Monte Carlo simulation as in HFRS to calculate and output a set of reliability metrics based on the chunk-level and file-level data layouts of the input.

5.2 Design Assumptions

We make the following design assumptions in our simulation framework.

Failure patterns: Due to lack of field data, we make two assumptions in the failure patterns. First, we simulate only independent failures, although recent work also reveals that disk failures in the same RAID group are actually correlated [27]. Also, we assume constant failure rates, although failure rates actually change over age [12], [32], [38]. Nevertheless, we focus on *relative* analysis that compares reliability with and without deduplication, instead of quantifying absolute reliability values. We expect that our assumptions suffice for our purpose.

Metadata: Our analysis focuses on file data only, but excludes metadata, including *file metadata* (e.g., superblock, inodes, namespace, etc.) and *deduplication metadata* (e.g., file recipes, fingerprint index, etc.). File metadata changes

frequently and is unlikely to be deduplicated [24], so we expect that the same amount of file metadata is stored after deduplication. Thus, it makes no impact on our reliability comparisons with and without deduplication.

On the other hand, deduplication metadata is critical for the reliability of the whole system (e.g., the loss of file recipes can compromise file reconstruction). Given the critical nature, we assume that we apply extra protection for deduplication metadata, such as increasing its redundancy protection via replication or erasure coding, and exclude its impact from our analysis. Nevertheless, we argue that deduplication metadata incurs limited storage overhead based on the analysis in [42], especially for primary storage deduplication. Let f be the metadata size divided by average chunk size, and D be the raw deduplication ratio of logical to physical size (excluding metadata). Then the storage overhead of deduplication metadata after deduplication is $f(1 + D)$. Suppose that $f = 0.4\%$ [42] and $D \leq 2$ [40] (the latter is derived in primary workloads). The storage overhead is no more than 1.2%, which remains small and has limited impact on our reliability comparisons.

Data layout: The data layout determines the loss and repair patterns in our simulation. In this paper, we assume a log-structured data layout, in which unique chunks after deduplication are sequentially appended to the end of the last write position. Note that log-structured data layout is also used in deduplication for primary (e.g., [40]) and secondary (e.g., [33]) storage. For the case without deduplication, the log-structured data layout implies that all chunks (either unique or duplicate) are sequentially stored, and hence both logical and physical views are identical. Also, we do not consider file-level fragmentation, which is not common [30].

5.3 Reliability Metrics

Given the limitations of traditional MTTDL (see Section 1), we consider new reliability metrics for accurate characterization. We start with the reliability metric called NOrmalized Magnitude of Data Loss (NOMDL) [17], which denotes the expected amount of data loss in bytes normalized to the storage capacity within the system mission time. NOMDL is shown to be comparable [17], allowing us to compare reliability with and without deduplication. In this work, we extend NOMDL for deduplication.

Note that the different logical and physical views in deduplication (see Section 2.1) imply different magnitudes of data loss and hence reliability interpretations. For example, losing an 8KB chunk that is referenced 10 times implies 80KB loss in the logical view as opposed to 8KB in the physical view. In this work, our reliability analysis focuses on the logical view, in which we measure the magnitude of data loss in the logical view normalized to the logical storage capacity. We believe that this reflects a more accurate reliability characterization to user applications, which perceive the logical view rather than the physical view.

Based on NOMDL, we define four normalized reliability metrics: (1) expected number of corrupted chunks per TB, (2) expected number of corrupted files per TB, (3) expected size (in bytes) of corrupted chunks per TB, and (4) expected size (in bytes) of corrupted files per TB. We say that a chunk or file is corrupted if any of its byte is corrupted. The first two metrics are called *non-weighted* metrics, while the other two are called *weighted* metrics to indicate the varying impact of a lost chunk or file, depending on its size.

TABLE 2
Parameters of our disk model.

	η (in hours)	β
Time-to-Failure	302,016	1.13
Time-to-Repair	22.7	1.65
Time-to-Scrub	186	1
Time-to-LSE	12,325	1

5.4 Disk Model

The disk model generates the failure and repair events according to some specified distributions. We consider two types of failures: *whole-disk failures* [32], [38] and *latent sector errors (LSE)* [4], [37]. A whole-disk failure triggers a repair operation, which uses the remaining operational disks to reconstruct the data of the failed disk into a new disk. On the other hand, an LSE indicates a corrupted sector that cannot be recovered by the internal error correction codes (ECC). It will not be detected until the affected sector is accessed. Since modern disks employ periodic scrubbing operations to proactively detect and correct LSEs [37], the disk model is designed to generate scrubbing events as well.

In this paper, we choose the parameters based on the near-line 1TB SATA Disk A model in [13], while the parameters of other disk models in [13] are also applicable and only change the absolute output numbers. Table 2 shows the parameters, all of which follow a Weibull distribution, where η denotes the characteristic life and β denotes the shape parameter (if $\beta = 1$, the distribution is exponential).

Our disk model generates two types of data loss events due to failures: *unrecoverable disk failures (UDFs)* and *uncorrectable sector errors (USEs)*. A UDF occurs when the number of failed disks exceeds the repair capability (e.g., a double-disk failure in RAID-5). Since multiple disks unlikely fail at the same time, the amount of lost data depends on how much data has been repaired in any earlier failed disk. For example, in RAID-5, if another whole-disk failure occurs while only 40% of the earlier failed disk has been repaired, then 60% of its sectors are lost. In this case, we assume that all the stripes (i.e., 60% of data in the disk array) associated with the lost sectors are corrupted. On the other hand, a USE occurs when the disk array is no longer fault-tolerant (e.g., a single-disk failure in RAID-5) and an LSE appears in a stripe (in any remaining operational disk) that has not been repaired. For example, in RAID-5, if only 40% of the earlier failed disk has been repaired, then an LSE becomes a USE with a 60% probability. Here, we ignore the data loss due to multiple simultaneous LSEs in the same stripe, since the probability of its occurrence is very small [16].

We use RAID-6 (with double-disk fault tolerance) as an example to explain the workflow of the disk model. Initially, the disk model calculates the lifespan of each disk in RAID, and pushes a whole-disk failure event of each disk to the event queue (based on the Time-to-Failure distribution). When the event queue pops a whole-disk failure event, the disk model calculates the repair time needed to reconstruct the failed disk (based on the Time-to-Repair distribution)

and pushes a repair event at the end of the repair time to the event queue. Once the event queue pops a repair event, the disk model calculates the lifespan of the new disk and pushes a new whole-disk failure event to the event queue. If a popped event exceeds the system mission time, the simulation stops.

When a whole-disk failure event is popped up, the RAID-6 disk array is in one of the three cases: (1) all other disks are operational, (2) there is an earlier failed disk under repair, and (3) there are two earlier failed disks under repair. For the first case, the disk array remains operational, and no data is lost. For the second case, the disk array is no longer fault tolerant, and any LSE would lead to data loss. To derive the LSE rate, we first compute the duration of the current scrubbing period (based on the Time-to-Scrub distribution), and then calculate the number of LSEs within this period (based on the Time-to-LSE distribution). If we quantify the repair progress of the earlier failed disk P_r as $(t_c - t_s)/(t_e - t_s)$, where t_c is the current time, t_s is the start time of the repair operation (i.e., the time when the whole-disk failure of the earlier failed disk occurs), and t_e is the expected end time of the repair operation, then an LSE becomes uncorrectable (i.e., a USE is triggered) with probability $1 - P_r$. Finally, for the third case, we trigger a UDF, and a fraction of $1 - P_r$ stripes are lost (where P_r is calculated as above). Due to the severity of a UDF, we ignore the already observed USEs in the current iteration, and proceed to the next iteration immediately.

5.5 Deduplication Model

The deduplication model computes the reliability metrics in the logical view based on the failure and repair patterns in the disk model that are actually defined in the physical view. We consider two levels of reliability metrics: *chunk level* and *file level*.

For a UDF, the magnitude of data loss depends on the logical repair progress, which we quantify as the fraction of repaired chunks or files in the logical view:

$$R_L = \sum_i \frac{|c_i| \times r_i}{C_L}, \quad (1)$$

where $|c_i|$ is the number (resp. size) of the i-th repaired physical chunk or file, r_i is the reference count for chunk c_i, and C_L is the total number (resp. size) of chunks (or files) in storage for the non-weighted (resp. weighted) case. Since the RAID layer is generally unaware of deduplication and cannot determine how data is shared and which chunks (or files) should be repaired first to minimize the impact of data loss. Thus, we consider two baseline repair strategies: *forward* and *backward*, in which the RAID layer repairs a failed disk from the beginning to the end of the log and from the end to the beginning of the log, respectively. Since the highly referenced chunks are more likely to appear near the beginning of the log, we expect that forward repair restores logical chunks at a faster rate than backward repair, and hence return better reliability metrics in both chunk and file levels. The two strategies hence serve as a better case and a worse case, respectively. Note that when there is no deduplication, both forward and backward repairs always restore logical data at the same rate.

For a USE, we assume that it corrupts a single physical sector that is uniformly selected from the entire disk space, and hence the associated physical chunk (or file). The number of corrupted logical chunks (or files) is the corresponding reference count. We expect that a larger chunk (or file) is more likely to be corrupted as it occupies more sectors.

6 Results

We now conduct reliability analysis via our simulation framework to the datasets. We evaluate the impact on storage reliability when deduplication is applied to individual file system snapshots (in both FSL and MS datasets) and deduplication domains with multiple file system snapshots (in the MS dataset), compared to without deduplication. Our analysis focuses on the most prevalent RAID-6 configuration, with 16 1TB disks and a 10-year system mission time [13]. We run 1.025 trillion simulation iterations to obtain enough loss events. Each iteration returns either the magnitudes of data loss should UDFs or USEs happen, or zero otherwise. We plot the average results over all iterations and the relative errors with 95% confidence (some results may have very small confidence intervals that are invisible in the plots). In all iterations, we observe a total of 1,389,250 UDFs and 332,993,652 USEs, or equivalently, the probabilities that a system suffers from a UDF or a USE are 1.36×10^{-6} and 3.25×10^{-4}, respectively. Then we compute the corresponding reliability metrics. To this end, we make key observations from our analysis. We also consider a deduplication strategy that improves reliability at the expense of (slight) storage overhead.

6.1 Uncorrectable Sector Errors

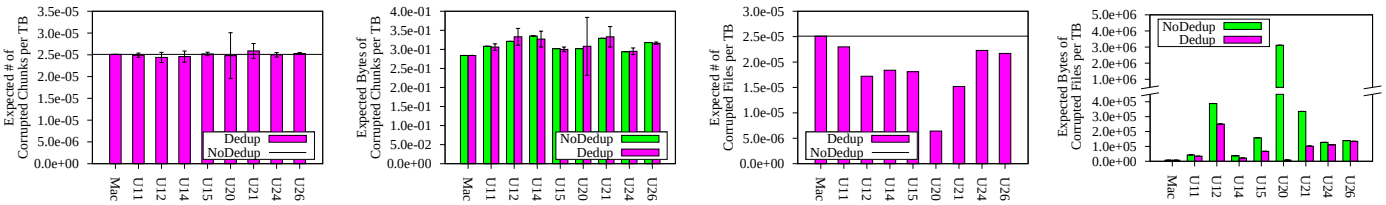
As expected, USEs occur more frequently than UDFs. We study the reliability due to USEs with deduplication (denoted by *Dedup*) and without deduplication (denoted by *NoDedup*). Figures 8 and 9 show the results of different reliability metrics in the FSL and MS datasets, respectively.

We first study the reliability in the FSL dataset (see Figure 8). Figure 8(a) shows the non-weighted chunk-level reliability in the FSL dataset. We observe no notable difference between *Dedup* and *NoDedup*, conforming to the conjecture in [20]. An intuitive explanation is that while deduplication reduces the probability of losing a physical chunk by some factor due to space reduction, it also increases the number of lost logical chunks by the same factor should a physical chunk be lost. Most cases have small relative errors, except U20. Our investigation is that a chunk in U20 is referenced by over 28 million times (see Section 4.1), so each loss of the chunk implies a high magnitude of loss and leads to a high deviation.

Figure 8(b) shows the weighted chunk-level reliability for the FSL dataset. We again observe that the reliability results are similar in both *Dedup* and *NoDedup*.

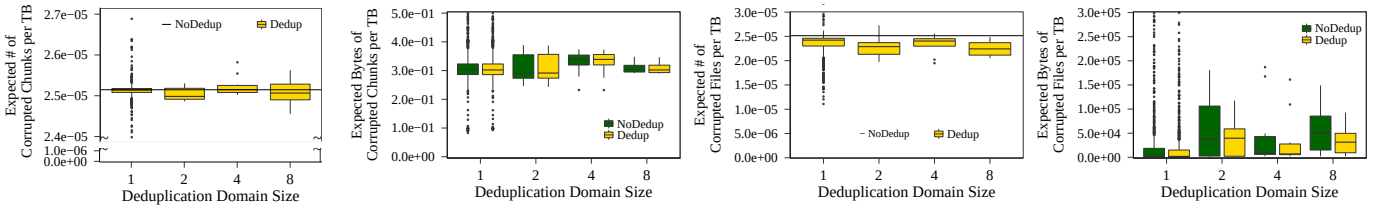
Observation (1) – *Deduplication will not significantly alter the expected amounts of corrupted chunks by USEs when compared to without deduplication.*

Figure 8(c) shows the non-weighted file-level reliability in the FSL dataset. We observe that *Dedup* reduces the expected number of corrupted files per TB by up to 74.4%



(a) Non-weighted chunk level (b) Weighted chunk level (c) Non-weighted file level (d) Weighted file level

Fig. 8. Reliability due to uncorrectable sector errors in the FSL dataset.



(a) Non-weighted chunk level (b) Weighted chunk level (c) Non-weighted file level (d) Weighted file level

Fig. 9. Reliability due to uncorrectable sector errors in the MS dataset.

when compared to *NoDedup*. Our investigation is that intra-file redundancy is prevalent in most snapshots, such that the references of a shared chunk mostly come from a single file. In particular, the virtual disk images and package files are major contributors to intra-file redundancy. Thus, if a highly referenced chunk is corrupted, it may only corrupt a single file rather than multiple files. We also observe that few snapshots have similar numbers of corrupted files in both *Dedup* and *NoDedup*, mainly due to very limited intra-file redundancy (e.g., Mac, as shown in Figure 5(a)).

Figure 8(d) shows the weighted file-level reliability in the FSL dataset. *Dedup* again reduces the expected size of corrupted files per TB by up to 99.7% when compared to *NoDedup*. Compared to non-weighted metrics, *Dedup* is more effective in mitigating data loss in weighted metrics, mainly because intra-file redundancy mostly comes from large files (see Figure 6). To understand the intuition behind, we consider a toy example. Suppose that we have two files, one with 10 chunks and another with 90 chunks in the logical views, and there are five duplicate chunks within one of the files. Now we encounter a USE. If the five duplicate chunks appear within the small file, the expected size of corrupted files is $5/95 \times 10 + 90/95 \times 90 = 85.79$ chunks; if the five duplicate chunks appear within the large file, the expected size of corrupted files is only $10/95 \times 10 + 85/95 \times 90 = 81.58$ chunks. Thus, if intra-file redundancy is more likely to occur in large files, the expected size of corrupted files also decreases.

Observation (2) – *In the presence of individual chunk corruptions caused by USEs, deduplication decreases the expected amounts of corrupted files, mainly because of the intra-file redundancy found in individual snapshots.*

Note that some existing work (e.g., [25]) applies additional replicas or more reliable erasure codes to highly referenced chunks to protect against individual chunk corruptions. Our findings suggest that this kind of failures is not a major threat to reliability in primary storage deduplication.

We now study the reliability in the MS dataset (see Figure 9). We find that the results are mostly consistent with those in the FSL dataset. Figures 9(a) and 9(b) show both non-weighted and weighted chunk-level reliability results in the MS dataset, respectively. We find that the reliability results are similar in both *Dedup* and *NoDedup*, regardless of the deduplication domain size.

Figure 9(c) shows the non-weighted file-level reliability in the MS dataset. In most cases, *Dedup* reduces the expected number of corrupted files per TB by up to 55.9% when compared to *NoDedup*. However, there exist some cases in which *Dedup* has more expected number of corrupted files than *NoDedup*. In those cases, the fraction of **DupFile** accounts for more than 80%. Thus, losing a chunk may lead to many files being corrupted when deduplication is applied, while in *NoDedup*, only one file is corrupted.

Figure 9(d) shows the weighted file-level reliability in the MS dataset. Similar to the FSL dataset, deduplication can reduce more expected bytes of corrupted files in the MS dataset, since the likelihood of losing a chunk is lowered by space savings.

Observation (3) – *Both FSL and MS datasets show consistent reliability results due to USEs. The reliability results are also similar across different deduplication domain sizes in the MS datasets.*

6.2 Unrecoverable Disk Failures

We now study the impact of UDFs. We first show how the logical repair progress is related to the physical repair progress, and identify potential problems. We further compare storage system reliability with and without deduplication under UDFs (i.e., *Dedup* and *NoDedup*, respectively).

6.2.1 Logical Repair Progress

Figure 10 shows the forward and backward repair strategies (see Section 5.5). Here, we only focus on the analysis in the FSL dataset in the interest of space. The X-axis represents the physical repair progress in 1% granularity, while the Y-axis represents the relative logical repair progress. Given a physical repair progress, we apply Equation (1) to calculate

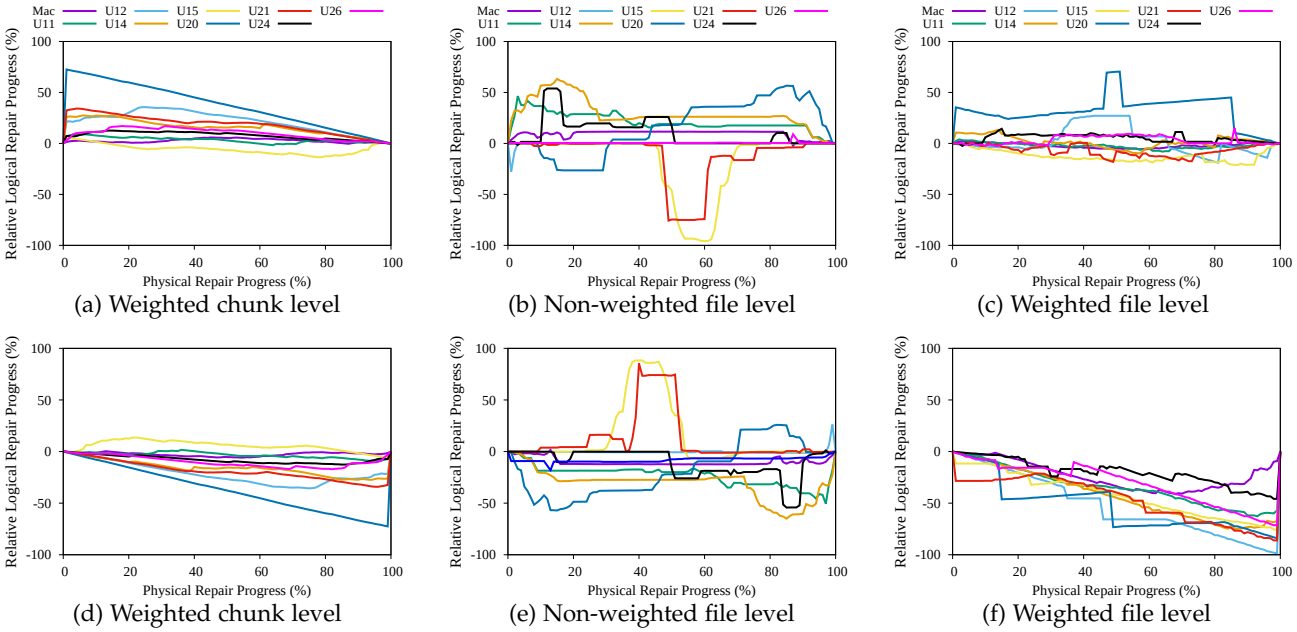


Fig. 10. The relative logical repair progress versus the physical repair progress under deduplication in the FSL dataset. We consider two repair strategies: forward repair (figures (a)-(c)) and backward repair (figures (d)-(f)).

the logical repair progress for both *NoDedup* and *Dedup*, denoted by L_n and L_d, respectively. We then calculate the relative logical repair progress defined as $L_d - L_n$, which specifies the amounts of logical chunks or files that have been repaired under *Dedup* relative to those under *NoDedup*. If it is positive (resp. negative), it means that *Dedup* improves (resp. degrades) the repair speed when compared to *NoDedup*. Note that if we have repaired 0% or 100% of physical chunks, the relative logical repair progress is zero.

Figures 10(a) and 10(d) show the weighted chunk-level reliability for the forward and backward repair strategies, respectively; the non-weighted results are similar and hence omitted. In forward repair, we observe positive results in most snapshots except U15, which shows slightly negative results. On the other hand, backward repair is exactly opposite, in which deduplication degrades the logical repair progress in most snapshots. The results are expected, since the highly referenced chunks are mainly appear at the log beginning, and repairing them first in forward repair can help the logical repair progress. We expect that deduplication can exacerbate UDFs in the chunk level if the highly referenced chunks are not carefully placed and preferentially repaired.

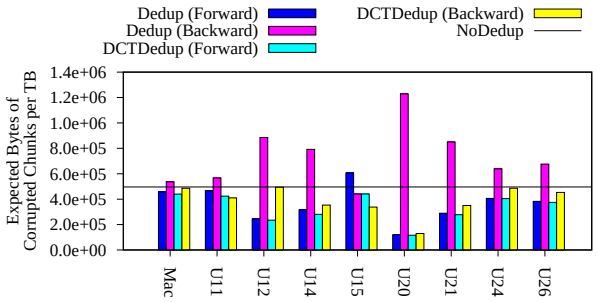
Figures 10(b) and 10(e) show the non-weighted file-level reliability for the forward and backward repair strategies, respectively. The results are similar to the chunk-level ones, such that forward repair shows positive results in most snapshots while backward repair shows the opposite. Since the non-weighted metric is only related to the number of repaired files rather than the file size and the majority of files have small sizes in each snapshot (as confirmed by [41]), the non-weighted metric actually reflects the repair progress of small files. We observe that small files tend to be completely deduplicated with other files rather than partially deduplicated. Hence, the results are related to the locations of duplicate small files. For example, forward repair makes positive logical repair progress in U11 and U14, mainly because a small file is copied by 561 times in U11 and a number of small files are copied by 8 times in U14, both of which happen near the log beginning. On the other hand, forward repair makes negative logical repair progress in U15 and U21 (around the middle of the physical repair progress), mainly because there are a number of duplicate small files that appear closer to the log end than the log beginning.

Figures 10(c) and 10(f) show the weighted file-level reliability for the forward and repair strategies, respectively. We see that in backward repair, all snapshots show significantly negative results. The reason is that large files are dominant in the weighted metric, and large files tend to be partially deduplicated with other files rather than completely duplicated. Sharing chunks among large files lead to significant *chunk fragmentation* [22], meaning that the chunks of individual files are scattered across storage rather than sequentially stored. Thus, restoring more chunks does not necessarily imply that the large files are completely restored (i.e., a large size of data is still considered to be corrupted), since some chunks may be deduplicated with the chunks of other files that are not yet restored. We expect that chunk fragmentation caused by deduplication can significantly exacerbate UDFs in weighted file-level metric.

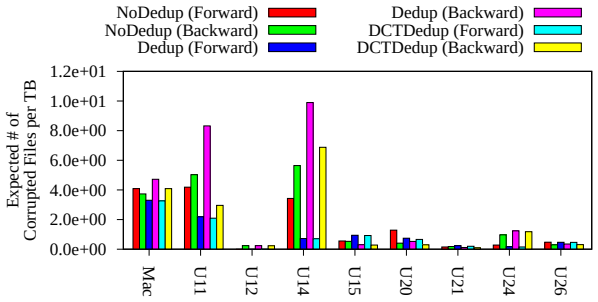
Observation (4) – *The logical repair progress is affected by the placement of highly referenced chunks and the severity of chunk fragmentation.*

6.2.2 Chunk-Level and File-Level Reliability

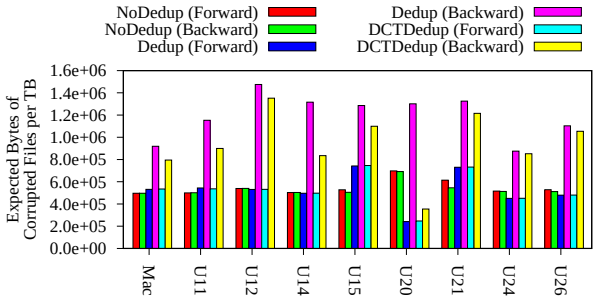
We now compare the impact of UDFs with and without deduplication. Figures 11 and 12 show the results of different reliability metrics in the FSL and MS datasets, respectively. Since the non-weighted and weighted chunk-level results are very similar, we only show the weighted



(a) Weighted chunk level

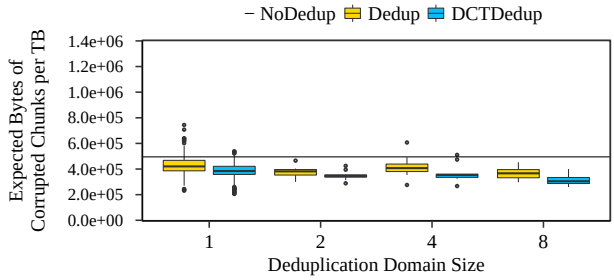


(b) Non-weighted file level

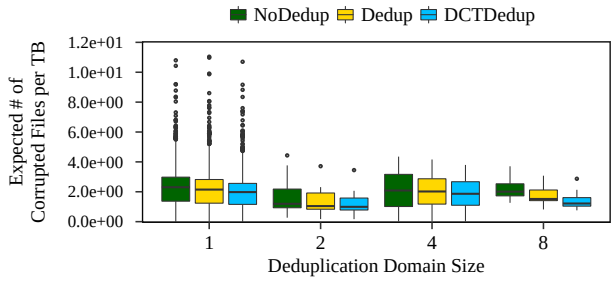


(c) Weighted file level

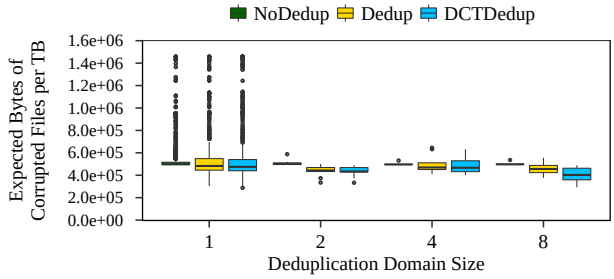
Fig. 11. Reliability due to unrecoverable disk failures in the FSL dataset.



(a) Weighted chunk level



(b) Non-weighted file level



(c) Weighted file level

Fig. 12. Reliability due to unrecoverable disk failures in the MS dataset.

chunk-level reliability (see Figures 11(a) and 12(a)). In the FSL dataset, we consider both forward and backward repair strategies, while in the MS dataset, we assume that only forward repair is used. Note that *DCTDedup* shows the results of the deliberate copy technique, which will be discussed in Section 6.2.3).

Figure 11(a) shows the weighted chunk-level reliability in the FSL dataset. In *NoDedup*, we see no difference between the forward and backward repair strategies, and UDFs will corrupt 495,880 bytes of chunks in the 10-year mission time. In forward repair, *Dedup* reduces the expected amounts of corrupted chunks caused by UDFs in most snapshots. The exception is U15, in which *Dedup* increases the expected bytes of corrupted chunks by 22.3%. Figure 10(a) explains the reasons. For example, in U15, *Dedup* degrades the logical repair progresses as some highly referenced chunks unfortunately appear closer to the log end (as confirmed by Figures 10(b) and 10(e)). In backward repair, deduplication degrades reliability in most snapshots, as highly referenced chunks likely appear in the log beginning.

Figure 12(a) shows the weighted chunk-level reliability in the MS dataset. In general, *Dedup* reduces the expected amount of corrupted chunks in most snapshots, and has higher reliability than *NoDedup* when the deduplication domain size is greater than one. However, there remain some exceptional individual file system snapshots (where the deduplication domain size is one) in which *Dedup* has lower reliability than *NoDedup*, since highly referenced chunks are stored closer to the log end.

Thus, while the log-structured layout is an ideal assumption, the highly referenced chunks can actually appear in any physical location in practice, especially when involving chunk migration in garbage collection [7]. Since RAID is unaware of deduplication semantic, there is no guarantee that the highly referenced chunks would be repaired by forward repair preferentially in the presence of a UDF. As a consequence, deduplication potentially exacerbates UDFs.

Observation (5) – *If we do not carefully place highly referenced chunks and repair them preferentially, deduplication can lead to more corrupted chunks in the presence of UDFs.*

We now study the impact of UDFs in the file level. Figure 11(b) shows the non-weighted file-level reliability in the FSL dataset. In *NoDedup*, the expected number of corrupted files caused by UDFs varies in different snapshots, due to the varying distributions of the numbers of

files and their sizes. On average, UDFs corrupt 1.6 files in forward repair and 1.9 in backward repair. Similar to chunk-level results, *Dedup* on average reduces the expected number of corrupted files by 14.6% in forward repair, but increases the number by 18.3% in backward repair. This is related to the locations of popular duplicate small files, which more possibly appear at the beginning of the log. For example, some popular duplicate small files appear at the beginning of the logs of U11 and U14, and hence we observe significantly positive results in the forward case but negative results in the backward case. Figure 12(b) shows the non-weighted file-level reliability in the MS dataset. We see that *Dedup* generally reduces the expected number of corrupted files regardless of the deduplication domain size, when compared to *NoDedup*.

Figure 11(c) shows the weighted file-level reliability in the FSL dataset. *Dedup* generally achieves reliability comparable to *NoDedup* in forward repair, but significantly degrades reliability in backward repair (124.6% more bytes in corrupted files). Figure 10(e) explains the reason. Due to deduplication, the log end generally has a higher degree of chunk fragmentation than the log beginning. The repaired fragmented chunks cannot help completely restore large files, making the logical repair progress slow based on the weighted metric. Figure 12(c) shows the weighted file-level reliability in the MS dataset. The median amount of expected bytes of corrupted files in *Dedup* is generally lower than that in *NoDedup* when forward repair is used for different deduplication domain sizes. However, for some cases when the deduplication domain size is one, *Dedup* has a larger amount of expected bytes of corrupted files than *NoDedup*. The reason is that duplicate files are the dominant in MS (see Section 4.2), that only one chunk in duplicate files is not repaired timely leads to the duplicate files corrupted.

Deduplication systems are naturally more fragmented than non-deduplication systems. How to reduce the chunk fragmentation to improve read performance has been a hot topic [14], [21], [22]. Our observation shows that the chunk fragmentation also potentially exacerbates UDFs in terms of the file-level weighted metric. To improve reliability, a defragmentation algorithm to aggregate similar files (the files sharing many chunks) into continuous physical addresses is required, such as the inline defragmentation algorithms proposed by previous work [14], [22] and offline defragmentation tools (e.g., e4defrag in ext4 file system [28]). We plan to consider these issues as our future work.

Observation (6) – *Deduplication is significantly more vulnerable to UDFs in terms of the file-level metrics if popular small files and chunk fragmentation are not carefully handled.*

6.2.3 Deliberate Copy Technique

In order to reduce the negative impacts of UDFs, we propose the *deliberate copy technique (DCT)*. Our observation is that the highly referenced chunks only account for a small fraction of physical capacity after deduplication, and the chunk reference counts show a long-tailed distribution based on our investigation. Hence, it is possible to allocate a small dedicated physical area in RAID for storing extra copies of highly referenced chunks, and always preferentially repair the physical area during RAID reconstruction.

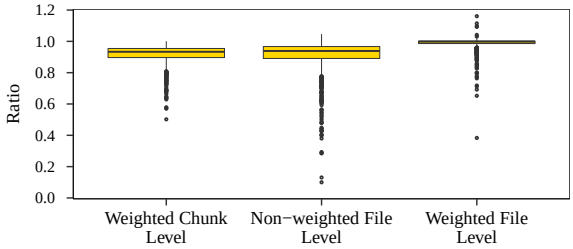


Fig. 13. Ratio of the expected amounts of corrupted chunks/files of DCT to that of *Dedup* for each reliability metric in the MS dataset when the deduplication domain size is one. A ratio smaller than one means DCT has better reliability than DCT, or vice versa.

We implement DCT in our simulation framework to show its effectiveness. Specifically, we allocate the first 1% of physical sectors for the highly referenced chunks (we explain the impact of the dedicated area size later in the discussion). In each snapshot, we sort the chunks by their reference counts, and fill the dedicated sectors with the top-1% most highly referenced chunks. While these chunks only occupy 1% of physical capacity, they account for 6%–50% of logical capacity and incur moderate storage overhead. Since the deliberate copies can be made offline, no change is required to the regular read/write path.

We revisit the reliability results in Figures 11 and 12. In addition, Figure 13 compares the reliability of DCT and *Dedup* in the MS dataset when the deduplication domain size is one, by computing the ratio of the expected amounts of corrupted chunks or files of DCT to that of *Dedup* in each file system snapshot for each reliability metric; a ratio smaller than one means DCT has better reliability than *Dedup*, or vice versa. We make the following observations.

First, we consider the weighted chunk-level reliability in both FSL and MS datasets as shown in Figures 11(a) and 12(a), respectively, where the reliability results of DCT is denoted by *DCTDedup*). In the FSL dataset, DCT reduces the expected bytes of corrupted chunks by 7.1% and 40.6% on average compared to *Dedup* in forward and backward repairs, respectively. In the MS dataset, DCT reduces the expected bytes of corrupted chunks by 8.0%, 7.1%, 11.6%, and 14.3% on average compared to *Dedup* for the deduplication domain size equal to one, two, four, and eight, respectively. Furthermore, compared to *NoDedup*, DCT is less vulnerable in general. From Figure 13, we see that DCT always outperforms *Dedup* in the weighted chunk-level reliability (where the ratio is always less than one).

Observation (7) – *By allocating a small dedicated physical area for storing highly referenced chunks, we can reduce the expected amounts of corrupted chunks by UDFs in both FSL and MS datasets.*

We study the effectiveness of DCT in file-level metrics. We revisit the non-weighted file-level reliability in the FSL and MS datasets as shown in Figures 11(b) and 12(b), respectively. In the FSL dataset, DCT on average reduces the expected number of corrupted files by 5.6% and 21.1% in forward and backward repairs compared to *Dedup*, respectively. As a result, DCT helps *Dedup* achieve 20.1% and 11.3% higher reliability than *NoDedup* in forward and backward repairs, respectively. In the MS dataset, DCT reduces the expected number of corrupted files on average

8.2%, 7.6%, 8.4%, and 19.3% for deduplication domain sizes equal to one, two, four, and eight, respectively.

We also revisit the weighted file-level reliability in the FSL and MS datasets as shown in Figures 11(c) and 12(c), respectively. In the FSL dataset, DCT on average incurs 1.5% less expected bytes in corrupted files than *NoDedup* in forward repair. On the other hand, in backward repair, DCT on average reduces 20.3% of bytes in corrupted files in *Dedup*, but still on average achieves 79.9% worse reliability than *NoDedup* because DCT cannot completely solve the chunk fragmentation problem. On the other hand, in the MS dataset, DCT on average has 2.0%, 15.3%, 1.6% and 19.7% less expected bytes in corrupted files than *NoDedup* for the deduplication domain size equal to one, two, four, and eight, respectively. In particular, we do not see significant improvement of DCT over *NoDedup* in the MS dataset when the deduplication domain size is one (i.e., when individual file system snapshots are considered).

Note that in the FSL dataset, DCT may have worse file-level reliability than *Dedup* (e.g., Mac in Figure 11(c)). We make similar observations for the MS dataset in Figure 13, in which DCT generally has better reliability than *Dedup* in the file-level metrics for the individual file system snapshots, yet there are exceptional cases where DCT may have worse reliability.

In general, increasing the dedicated area size allows more highly referenced chunks to be repaired first and hence improves reliability of *Dedup* (at the expense of larger storage overhead). On the other hand, having a very large dedicated area may eventually store some non-highly referenced chunks that are repaired first, in which case we do not see improved reliability in DCT. Choosing the right dedicated area size depends on the storage workload and is an open issue.

Observation (8) – *DCT in general reduces the expected amounts of corrupted files remarkably in both FSL and MS datasets, but it remains necessary to address chunk fragmentation to further improve reliability in the weighted file-level metric.*

7 Conclusions

This paper presents an in-depth study of the storage system reliability in primary storage deduplication. Our study is based on public real-world file system snapshots from two different groups, i.e., FSL and Microsoft. First, we study the redundancy characteristics of file system snapshots, regarding their reference count distributions and redundancy sources. We observe that there exist a few highly referenced chunks, and that intra-file redundancy, duplicate files, and similar files are the major sources of duplicate chunks. Then we propose a simulation framework and appropriate reliability metrics to compare storage system reliability with and without deduplication in the face of Uncorrectable Sector Errors (USEs) and Unrecoverable Disk Failures (UDFs). Regarding to USEs that cause individual chunk corruptions, we observe that deduplication does not alter the expected amounts of corrupted chunks, and remarkably reduces the expected amounts of corrupted files due to intra-file redundancy elimination. Regarding to UDFs that corrupt large areas of continuous physical chunks, deduplication leads to more corrupted chunks and files due to the unguarded chunk placement and chunk fragmentation. We propose a deliberate copy technique to allocate a small dedicated physical area in RAID for highly referenced chunks and preferentially repair the area during RAID reconstruction. We show that the deliberate copy technique significantly reduces the expected amounts of corrupted chunks and files.

In future work, we plan to study the deduplication reliability on (NAND-based) solid-state storage devices (SSDs). SSDs have inherently different I/O characteristics from harddisks, such as out-of-place updates and limited write endurance [3], Also, the flash error rates of SSDs increase with program/erase cycles [39]. How deduplication affects storage reliability on SSDs needs further investigation.

Acknowledgments

This work was supported by NSFC 61502190; Fundamental Research Funds for Central Universities, HUST, Grant 2015MS073; and GRF CUHK413813 from HKRGC.

References

[1] Microsoft traces and snapshots public archive. http://iotta.snia.org/tracetypes/6, 2009.

[2] Fsl traces and snapshots public archive. http://tracer.filesystems.org, 2015.

[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX Annual Technical Conference*, volume 57, 2008.

[4] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An analysis of latent sector errors in disk drives. In *Proc. ACM SIGMETRICS, 2007*.

[5] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In *Proc. IEEE MASCOTS, 2009*.

[6] D. Bhagwat, K. Pollack, D. D. Long, T. Schwarz, E. L. Miller, and J.-F. o. Paris. Providing high reliability in a minimum redundancy archival storage system. In *Proc. IEEE MASCOTS, 2006*.

[7] F. C. Botelho, P. Shilane, N. Garg, and W. Hsu. Memory efficient sanitization of a deduplicated storage system. In *Proc. USENIX FAST, 2013*.

[8] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29, Jun 1997.

[9] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in san cluster file systems. In *Proc. USENIX ATC, 2009*.

[10] 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 FAST, 2009*.

[11] 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. USENIX ATC, 2012*.

[12] J. G. Elerath and M. Pecht. A highly accurate method for assessing reliability of redundant arrays of inexpensive disks (raid). *Computers, IEEE Transactions on*, 58(3):289–299, 2009.

[13] J. G. Elerath and J. Schindler. Beyond mttdl: A closed-form raid 6 reliability equation. *ACM Trans. on Storage*, 10(2):7, 2014.

[14] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, and Q. Liu. Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information. In *Proc. USENIX ATC, 2014*.

[15] M. Fu, P. Lee, D. Feng, Z. Chen, and X. Yu. A simulation analysis of reliability in primary storage deduplication. In *Proc. IISWC, 2016*.

[16] K. M. Greenan. *Reliability and power-efficiency in erasure-coded storage systems*. PhD thesis, University of California, Santa Cruz, 2009.

[17] K. M. Greenan, J. S. Plank, and J. J. Wylie. Mean time to meaningless: Mttdl, markov models, and storage system reliability. In *Proc. USENIX HotStorage, 2010*.

[18] K. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei. An empirical analysis of similarity in virtual machine images. In *Proc. Middleware, 2011*.

[19] K. Jin and E. L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proc. ACM SYSTOR, 2009*.

[20] X. Li, M. Lillibridge, and M. Uysal. Reliability analysis of deduplicated and erasure-coded storage. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):4–9, 2011.

[21] Y.-K. Li, M. Xu, C.-H. Ng, and P. P. Lee. Efficient hybrid inline and out-of-line deduplication for backup storage. *ACM Trans. on Storage*, 11(1):2, 2015.

[22] M. Lillibridge, K. Eshghi, and D. Bhagwat. Improving restore speed for backup systems that use inline chunk-based deduplication. In *Proc. USENIX FAST, 2013*.

[23] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proc. USENIX FAST, 2009*.

[24] X. Lin, F. Douglis, J. Li, X. Li, R. Ricci, S. Smaldone, and G. Wallace. Metadata considered harmful ... to deduplication. In *Proc. USENIX HotStorage*, 2015.

[25] 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. ACM ICS, 2009*.

[26] M. Lu, D. Chambliss, J. Glider, and C. Constantinescu. Insights for data reduction in primary storage: a practical analysis. In *Proc. ACM SYSTOR*, 2012.

[27] A. Ma, F. Douglis, G. Lu, D. Sawyer, S. Chandra, and W. Hsu. Raidshield: characterizing, monitoring, and proactively protecting against disk failures. In *Proc. USENIX FAST, 2015*.

[28] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux Symposium*, volume 2, pages 21–33. Citeseer, 2007.

[29] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel. A study on data deduplication in hpc storage systems. In *Proc. IEEE SC, 2012*.

[30] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *Proc. USENIX FAST, 2011*.

[31] C.-H. Ng, M. Ma, T.-Y. Wong, P. P. C. Lee, and J. C. S. Lui. Live deduplication storage of virtual machine images in an open-source cloud. In *Proc. Middleware, 2011*.

[32] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proc. USENIX FAST, 2007*.

[33] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proc. USENIX FAST, 2002*.

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

[35] E. W. Rozier, W. H. Sanders, P. Zhou, N. Mandagere, S. M. Uttamchandani, and M. L. Yakushev. Modeling the fault tolerance consequences of deduplication. In *Proc. IEEE SRDS, 2011*.

[36] E. W. D. Rozier and W. H. Sanders. A framework for efficient evaluation of the fault tolerance of deduplicated storage systems. In *Proc. IEEE/IFIP DSN, 2012*.

[37] B. Schroeder, S. Damouras, and P. Gill. Understanding latent sector errors and how to protect against them. *ACM Trans. on Storage*, 6(3):9, 2010.

[38] 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. USENIX FAST, 2007*.

[39] B. Schroeder, A. Merchant, and R. Lagisetty. Reliability of NAND-based ssds: What field studies tell us. *Proceedings of the IEEE*, 105(9):1751–1769, Sep 2017.

[40] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. iDedup: Latency-aware, inline data deduplication for primary storage. In *Proc. USENIX FAST*, 2012.

[41] Z. Sun, G. Kuenning, S. Mandal, P. Shilane, V. Tarasov, N. Xiao, and E. Zadok. A long-term user-centric analysis of deduplication patterns. In *Proc. IEEE MSST*, 2016.

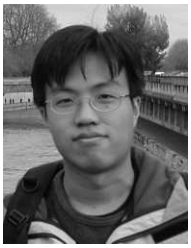
[42] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu. Characteristics of backup workloads in production systems. In *Proc. USENIX FAST, 2012*.

[43] A. Wildani, E. L. Miller, and O. Rodeh. Hands: A heuristically arranged non-backup in-line deduplication system. In *Proc. IEEE ICDE, 2013*.

[44] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proc. USENIX FAST, 2008*.

Min Fu is a Research Engineer in Sangfor Technologies Co., Ltd. He received the Ph.D. degree in Computer Science from Huazhong University of Science and Technology in 2016. His current research interests include storage systems, data deduplication, and caching. He publishes several papers in major journals and conferences including IEEE-TPDS, IEEE Proceedings, USENIX ATC, FAST, INFOCOM, MSST, IPDPS, IISWC, etc.

Shujie Han is currently a Ph.D. student in the Department of Computer Science and Engineering at the Chinese University of Hong Kong. Her research interests include data deduplication, reliability, etc.



Patrick P. C. Lee received the B.Eng. degree (first-class honors) in Information Engineering from the Chinese University of Hong Kong in 2001, the M.Phil. degree in Computer Science and Engineering from the Chinese University of Hong Kong in 2003, and the Ph.D. degree in Computer Science from Columbia University in 2008. He is now an Associate Professor of the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests are in various applied/systems topics including storage systems, distributed systems and networks, operating systems, dependability, and security.

Dan Feng received the B.Eng, M.Eng, and Ph.D. degrees in Computer Science and Technology in 1991, 1994, and 1997, respectively, from 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 on the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012. She is a member of IEEE and a member of ACM.

Zuoning Chen received the M.Eng degree in Computer Science and Technology from Zhejiang University, China. She is a fellow of Chinese Academy of Engineering, and engaged in research of system software and computer architecture. Her current research interests include secure platform for domestic CPU, and big data.

Yu Xiao is currently a Ph.D. student majoring in computer architecture at Huazhong University of Science and Technology (HUST), Wuhan, China. Her research interests include data deduplication, reliability, etc.