A Simulation Analysis of Reliability in Primary Storage Deduplication

Min Fu[†], Patrick P. C. Lee[‡], Dan Feng[†], Zuoning Chen[*], Yu Xiao[†]
[†]Wuhan National Lab for Optoelectronics
School of Computer, Huazhong University of Science and Technology, Wuhan, China
[‡]Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, HK
[*]National Engineering Research Center for Parallel Computer, Beijing, China

***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 real-world file system snapshots. Specifically, we propose a trace-driven, deduplication-aware simulation framework that analyzes 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.**

I. 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) [6], [18], [38]; recently, it has also been increasingly studied and deployed in primary storage (e.g., file systems) [5], [7], [26], [34], [37]. 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., [3], [6], [20]) add redundancy via replication or erasure coding to post-deduplication data for fault tolerance. Other studies (e.g., [15], [30], [31]) 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 determines whether important data copies are repaired first, and hence affects 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. However, some studies [8], [9], [12] 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 expect that MTTDL is even more inappropriate, due to the varying severity of data loss. Thus, we advocate *simulation* for accurate reliability analysis, at the expense of intensive computations [8]. To this end, this paper makes the following contributions.

First, we propose a trace-driven, deduplication-aware simulation framework to analyze and compare storage system reliability with and without deduplication, so as to identify any possible solution to improve storage system reliability should deduplication be deployed. Specifically, we start with a RAID disk array setting, and extend the notion of NOrmalized Magnitude of Data Loss (NOMDL) [12] 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.

Second, we apply our simulation framework to conduct reliability analysis on primary storage deduplication, which is less explored than secondary storage deduplication but has

Min Fu is now with Sangfor Technologies Co., Ltd.

received increasing attention. Specifically, we examine 18 real-life file system snapshots derived from two user groups. Our simulation results show the following key findings:

- 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 [17] 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 rest of the paper proceeds as follow. Section II presents the background and related work. Section III presents the design of our simulation framework. Section IV describes the datasets for our simulation study. Section V presents our simulation results. Finally, Section VI concludes the paper.

II. Background and Related Work

A. 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 [28]. 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 achieve higher deduplication efficiency than fixed-size chunking. Variable-size chunking can be implemented by Rabin Fingerprinting [29], which computes a rolling hash over a sliding window of file data and identifies boundaries whose rolling hashes match some target pattern.

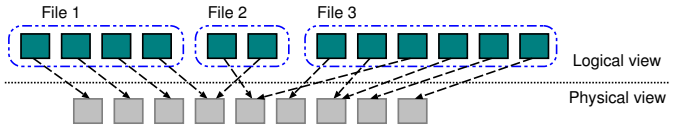

Fig. 1. Logical and physical views of a deduplication system.

To effective remove duplicate chunks, the average chunk size is typically on the order of kilobytes (e.g., 8KB [38]).

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 reconstruction.

In deduplication storage, we need to differentiate the *logical* and *physical* views, which describe the storage organizations with and without 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.

B. 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.* [14] and Jayaram *et al.* [13] show that deduplication effectively reduces the storage of virtual machine disk images, even with fixed-size chunking. Meyer *et al.* [25] analyze 857 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.* [21] propose different techniques on improving deduplication effectiveness in primary storage. Wallace *et al.* [36] 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.* [24] analyze four HPC centers and observe that deduplication can achieve 20-30% of storage savings. Sun *et al.* [35] 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 [27], [33] and latent sector errors [2], [32]. On the other hand, only few studies analyze the reliability of deduplication systems. Most studies propose to improve reliability of deduplication systems through controlled redundancy, either by replication [3] or erasure coding [6], [15], [20], but they do not analyze the reliability affected by deduplication. Li *et al.* [15] propose combinatorial analysis to evaluate the probability of data loss of deduplication systems. Rozier *et al.* [30], [31] propose automata-based frameworks to quantitatively evaluate the reliability of deduplication systems under disk failures and

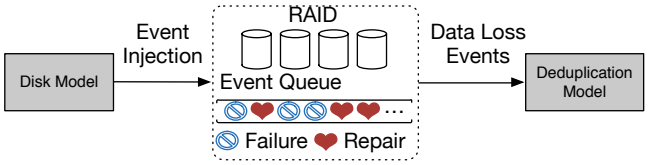

Fig. 2. Architecture of our simulation framework.

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.

III. Simulation Framework

In this section, we design a simulation framework that analyzes and compares storage system reliability with and without deduplication. Our simulation framework builds on the High-Fidelity Reliability Simulator (HFRS) [11] and specifically addresses deduplication.

A. Architectural Overview

Figure 2 shows the architecture of our simulation framework. The framework targets primary storage deduplication for individual file system snapshots under a disk-based RAID setting. Specifically, it takes a file system snapshot (Section IV), 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 sequential 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 file system snapshot.

B. 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 [22]. Also, we assume constant failure rates, although failure rates actually change over age [8], [27], [33]. 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 [19], 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 [36], 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 $f = 0.4\%$ [36] and $D \leq 2$ [34] (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., [34]) and secondary (e.g., [28]) 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. In addition, we do not consider file-level fragmentation, which is not prevalent [25].

C. Reliability Metrics

Given the limitations of traditional MTTDL (Section I), we consider new reliability metrics for accurate characterization. We start with the reliability metric called NOrmalized Magnitude of Data Loss (NOMDL) [12], 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 [12], 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 (Section II-A) 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

TABLE I
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

(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 last two metrics are called *weighted* metrics to indicate the varying impact of a lost chunk or file, depending on its size.

D. 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* [27], [33] and *latent sector errors (LSE)* [2], [32]. 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 [32], 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 [9], while the parameters of other disk models in [9] are also applicable and only change the absolute output numbers. Table I 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 [11].

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.

E. Deduplication Model

The deduplication model computes the reliability metrics in the logical view (Section III-C) based on the failure and repair patterns in the disk model that are actually defined in the physical view (Section III-D). 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:

$$R_L = \sum_i \frac{|c_i| \times r_i}{C_L}, \tag{1}$$

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

TABLE II
STATISTICS OF EACH FILE SYSTEM SNAPSHOT IN OUR DATASETS.

Snapshot	OS	Date	Raw Size (GB)	# Files	# Chunks	Dedup (%)
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%
S1	OS X Yosemite	Collected in July and August 2015	143.20	1,392,222	17,828,066	29.0%
S2	Linux Mint 15		263.47	282,934	27,780,340	43.1%
S3	Linux Mint 17		127.97	117,488	14,335,275	31.8%
S4	Windows 7		123.60	255,285	12,830,663	53.5%
S5	Ubuntu 14.04		185.47	416,971	22,505,162	12.7%
S6	Ubuntu 14.04		151.37	166,112	17,947,291	18.6%
S7	Linux Mint 13		83.33	348,356	9,931,882	22.0%
S8	Windows 7		274.43	693,620	31,328,634	34.4%
S9	Ubuntu 12.04		219.15	409,453	26,544,008	15.2%

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.

IV. DATASETS

Our reliability analysis focuses on primary storage deduplication, in which we consider the real-world static file system snapshots of two user groups. Table II 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 first dataset, namely *FSL*, consists of nine file system snapshots collected by the File system and Storage Lab (FSL) at Stony Brook University [1]. 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 [35].

The second dataset, namely *SELF*, contains nine file system snapshots from our own research group, and serves for the cross-validation purpose. We scan the file system snapshots, denoted by S1–S9, during July and August in year 2015. The snapshots S1–S8 are taken from the home directories of different users' desktops, while S9 is taken from the repository of a file server. We chunk each snapshot using Rabin Fingerprinting with the same setting as in the FSL dataset.

Our current datasets are limited by scale. Nevertheless, our position is to provide preliminary insights based on our available datasets and guide the design of our simulation framework (Section III) for our reliability comparisons. Our methodology is general and applicable to any larger-scale dataset of file system snapshots.

V. RESULTS

We now conduct reliability analysis via our simulation framework to the datasets. Our analysis focuses on the most prevalent RAID-6 configuration, with 16 1TB disks and a 10-year system mission time [9]. We run 1,025 billion 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

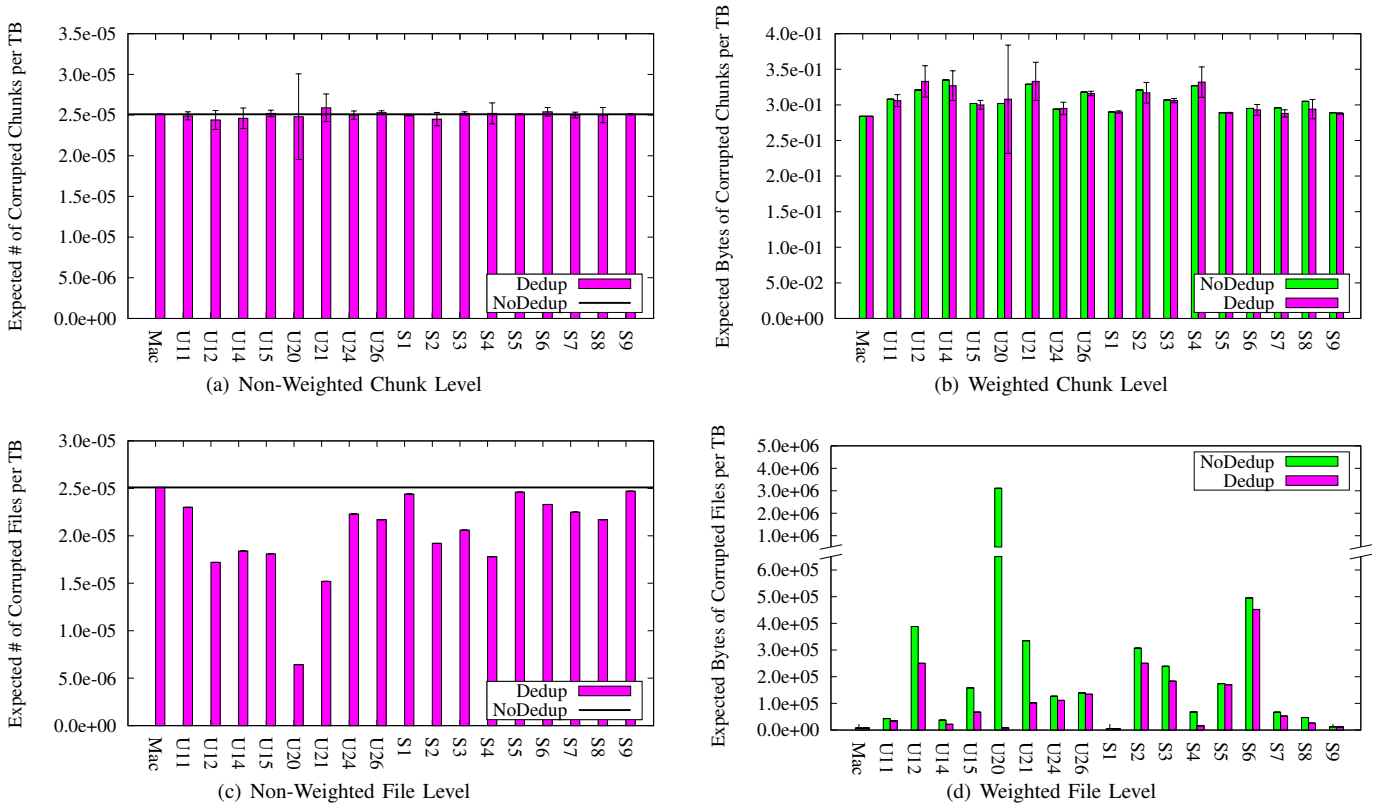

Fig. 3. Reliability metrics due to uncorrectable sector errors.

make key observations from our analysis. We also consider a deduplication strategy that improves reliability at the expense of (slight) storage overhead.

A. Uncorrectable Sector Errors

As expected, USEs occur more frequently than UDFs. We now study the reliability due to USEs with deduplication (denoted by *Dedup*) and without deduplication (denoted by *NoDedup*). Figure 3 shows different reliability metrics.

Figure 3(a) first shows the non-weighted chunk-level reliability. We observe no notable difference between *Dedup* and *NoDedup*, conforming to the conjecture in [15]. 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, so each loss of the chunk implies a high magnitude of loss and leads to a high deviation.

Figure 3(b) shows the weighted chunk-level reliability. 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 3(c) shows the non-weighted file-level reliability. We observe that *Dedup* reduces the expected number of corrupted files per TB by up to 74.4% 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 and S1) or low deduplication efficiency (e.g., S5 and S9).

Figure 3(d) shows the weighted file-level reliability. *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. 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.

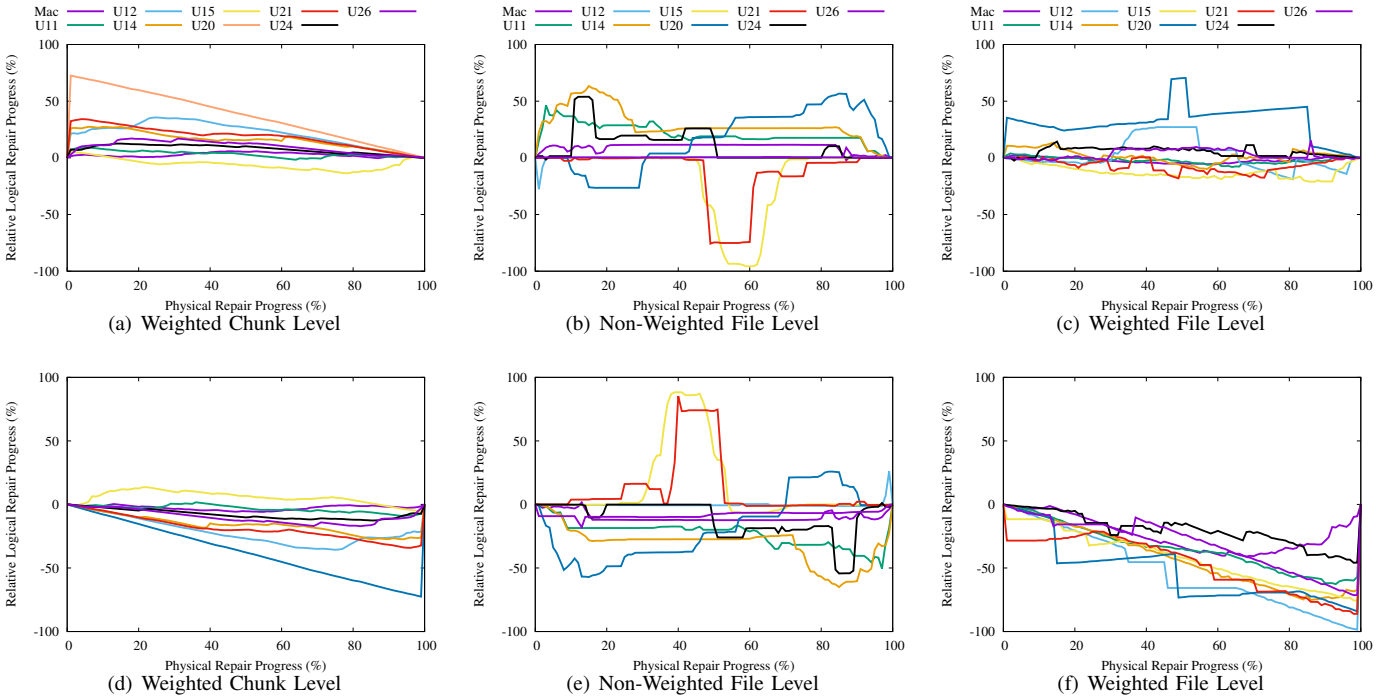

Fig. 4. The relative logical repair progress versus the physical repair progress under deduplication for two repair strategies: forward repair (figures (a)-(c)) and backward repair (figures (d)-(f)).

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 [20] 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.

B. 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).

1) Logical Repair Progress: Figure 4 shows the forward and backward repair strategies (Section III-E). Here, we only show the snapshots 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 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 4(a) and 4(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 4(b) and 4(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 [35]), 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

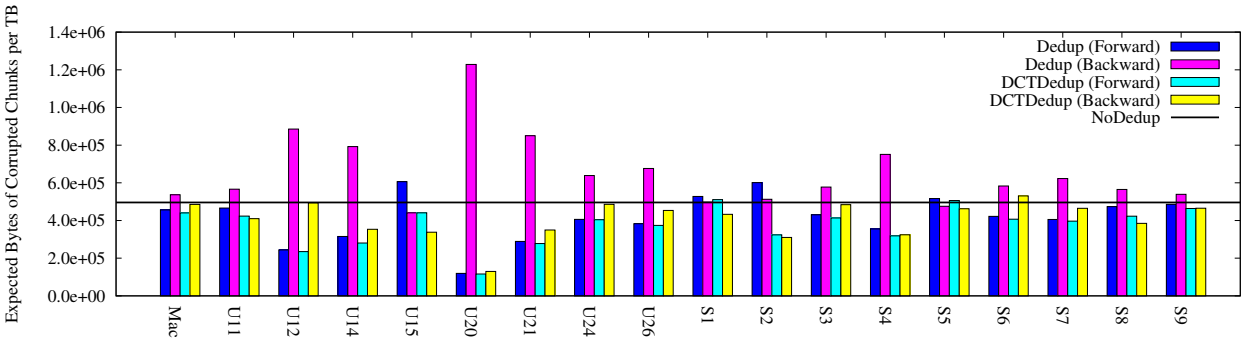

Fig. 5. Chunk-level comparisons of UDFs with and without deduplication.

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 4(c) and 4(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* [17], 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 (3) – *The logical repair progress is affected by the placement of highly referenced chunks and the severity of chunk fragmentation.*

2) Chunk-Level Comparisons: We now compare the impact of UDFs with and without deduplication. Figure 5 shows the simulation results of UDFs in the chunk level (note that *DCTDedup* shows the results of the deliberate copy technique, which will be discussed in Section V-C). We again only show the weighted results since the non-weighted ones are very similar. In *NoDedup*, we observe no difference between the forward and backward repair strategies, and UDFs will corrupt 495880 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 exceptions are U15, S1, S2, and S5, in which *Dedup* increases the expected bytes of corrupted chunks by 4.2%–22.3%. Figure 4(a) explains the reasons. For example, in U15, *Dedup* degrades the logical repair progresses because some highly referenced chunks unfortunately appear closer to the log end (as confirmed by Figures 4(b) and 4(e)) In backward repair, deduplication degrades reliability in most snapshots. On average, we observe a 38.1% increase in the expected bytes of corrupted chunks. As expected, backward repair is worse than forward repair because highly referenced chunks are likely to appear in the log beginning.

We point out that 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 [4]. Since RAID is unaware of deduplication semantic, there is no guarantee that the highly referenced chunks would be repaired preferentially in the presence of a UDF. As a consequence, deduplication potentially exacerbates UDFs.

Observation (4) – *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.*

3) File-Level Comparisons: We now compare the impact of UDFs in the file level. Figure 6(a) first shows the non-weighted file-level reliability. 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 1.4 files in forward repair and 1.7 in backward repair. Similar to chunk-level results, *Dedup* on average reduces the expected number of corrupted files by 24.2% in forward repair but increases the number by 32% 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 6(b) shows the weighted file-level reliability. *Dedup* generally achieves reliability comparable to *NoDedup* in forward repair, but significantly degrades reliability in backward repair (102.8% more bytes in corrupted files). Figure 4(f) 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.

Deduplication systems are naturally more fragmented than non-deduplication systems, how to reduce the chunk fragmen-

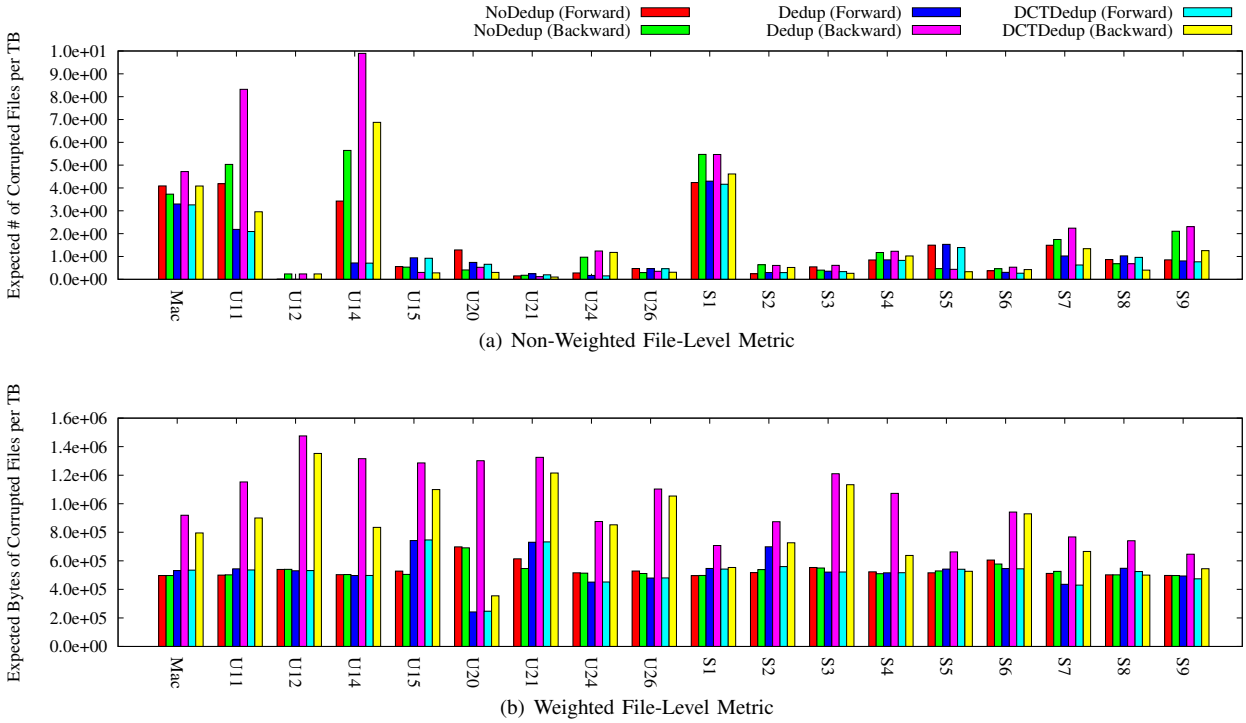

Fig. 6. The consequences of UDFs in terms of file-level non-weighted/weighted metric.

tation to improve read performance has been a hot topic [10], [16], [17]. 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 [10], [17] and offline defragmentation tools (e.g., e4defrag in ext4 file system [23]). We would like to take these as our future work.

Observation (5) – *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.*

C. 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.

We implement DCT in our simulator framework to show its effectiveness. Specifically, we allocate the first 1% of physical sectors for the highly referenced chunks. 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 Figure 5 for the simulation results of DCT (denoted by (*DCTDedup*) in the chunk level. In forward repair, DCT reduces the expected bytes of corrupted chunks by 9% on average, while in backward repair, we observe a 35.8% decrease on average. Compared with *NoDedup*, *DCTDedup* is less vulnerable to UDFs in general. Thus, we believe that DCT can maintain the chunk-level reliability in deduplication.

Observation (6) – *By allocating a small dedicated physical area for storing highly referenced chunks, we can reduce the expected amounts of corrupted chunks by UDFs.*

We now study the effectiveness of DCT in file-level metrics. Figure 6(a) shows the non-weighted file-level metric. DCT on average reduces the expected number of corrupted files by 6.1% in forward repair and by 33.4% in backward repair in *Dedup*. As a result, DCT helps *Dedup* achieve 28.8% and 12.2% higher reliability than *NoDedup* in forward and backward repair strategies, respectively.

Figure 6(b) shows the weighted file-level metric. In forward repair, DCT on average has 3.9% less bytes in corrupted files than *NoDedup*. On the other hand, in backward repair, DCT on average reduces 21.4% of bytes in corrupted files in *Dedup*, but still achieves (59.1%) worse reliability than *NoDedup* because DCT cannot completely solve the chunk fragmentation problem.

Observation (7) – *DCT reduces the expected amounts of corrupted files remarkably, but a defragmentation algorithm*

is necessary to further improve reliability in the weighted file-level metric.

VI. Conclusions

This paper revisits the problem of storage system reliability in deduplication. 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 (USE) and Unrecoverable Disk Failures (UDF), and examine real-life file system snapshots that represent the workloads for primary storage deduplication. 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 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.

Acknowledgments

This work was supported in part by NSFC No. 61502190; Fundamental Research Funds for the Central Universities, HUST, under Grant No. 2015MS073; and GRF CUHK413813 from HKRGC.

References

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

[2] 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*.

[3] 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*.

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

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

[6] 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*.

[7] 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*.

[8] 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.

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

[10] 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*.

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

[12] 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*.

[13] 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*.

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

[15] 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.

[16] 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.

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

[18] 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*.

[19] 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.

[20] 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*.

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

[22] 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*.

[23] 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.

[24] 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*.

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

[26] 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*.

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

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

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

[30] 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*.

[31] 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*.

[32] 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.

[33] 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*.

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

[35] 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.

[36] 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*.

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

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