Improving Reliability of Deduplication-based Storage Systems with Per-File Parity

Suzhen Wu*, Huagao Luan*, Bo Mao†, Hong Jiang‡, Gen Niu†, Hui Rao*, Fang Yu*, Jindong Zhou†
*Computer Science Department of Xiamen University, China
†Software School of Xiamen University, China
‡Department of Computer Science and Engineering, University of Texas at Arlington, USA

***Abstract*—The reliability issue in deduplication-based storage systems has not received adequate attention. Existing approaches introduce data redundancy after files have been deduplicated, either by replication on critical data chunks, i.e., chunks with high reference count, or RAID schemes on unique data chunks, which means that these schemes are based on individual unique data chunks rather than individual files. This can leave individual files vulnerable to losses, particularly in the presence of transient and unrecoverable data chunk errors such as latent sector errors. To address this file reliability issue, this paper proposes a Per-File Parity (short for PFP) scheme to improve the reliability of deduplication-based storage systems. PFP computes the XOR parity within parity groups of data chunks of each file after the chunking process but before the data chunks are deduplicated. Therefore, PFP can provide parity redundancy protection for all files by intra-file recovery and a higher-level protection for data chunks with high reference counts by inter-file recovery. Our reliability analysis and extensive data-driven, failure-injection based experiments conducted on a prototype implementation of PFP show that PFP significantly outperforms the existing redundancy solutions, DTR and RCR, in system reliability, tolerating multiple data chunk failures and guaranteeing file availability upon multiple data chunk failures. Moreover, a performance evaluation shows that PFP only incurs an average of 5.7% performance degradation to the deduplication-based storage system.**

***Index Terms*—Data Deduplication; Reliability; Per-File Parity; Intra-File Recovery; Inter-File Recovery**

I. Introduction

The explosive growth in data volume has posed one of the most critical challenges for the design and management of large-scale storage systems. Consequently, data reduction technologies have been propelled to the forefront of research and development in addressing this challenge in the big data era [24]. Data deduplication, a space efficient data reduction technology, has spurred a great deal of research interest from both industry and academia [21], [27]. It has been deployed in a wide range of storage systems, including backup and archiving systems [29] and primary storage systems such as VM (Virtual Machine) servers [12].

Most existing studies on data deduplication focus on improving its efficiency by developing or optimizing chunking schemes to find as much redundant data as possible, solving the index-lookup disk-bottleneck problem, and addressing the hash computing overhead issue and the data restore problem [9], [27]. However, the impact of data deduplication on the reliability of the stored data has not been well understood nor studied for large-scale storage systems [4], [21]. This is because reliability is often synonymous with redundancy and, with data deduplication, data redundancy is completely eliminated by its very design. In other words, since only a single copy of duplicate data common to and referenced by different files, also referred to as a critical data chunk, is stored in the persistent storage after deduplication, the loss of one or a few critical data chunks can lead to many referencing files to be lost, thus significantly reducing the reliability of the storage system. Moreover, in a large-scale storage system, a post-deduplication file may have its constituent data chunks stored on multiple different storage devices. If any one of these constituent data chunks or storage devices fails, the file is lost. Therefore, *data deduplication magnifies the negative impact of data loss in large-scale storage systems*.

The existing approaches to the reliability problem in deduplication-based storage systems can be classified into two categories [27], namely, *deduplication-then-RAID* (DTR), where the stored unique data chunks are organized and thus protected by a RAID scheme, and *reference-count based replication* (RCR), where data chunks with sufficiently high reference counts (i.e., number of different files sharing/referencing the same data chunk) are replicated on different storage devices. While approaches in both categories introduce data redundancy for reliability of deduplication-based storage systems, they do so after files have been deduplicated and generate data redundancy based on the stored unique data chunks to *prevent loss of individual data chunks rather than individual files*. In other words, files can still be vulnerable to data loss because of the specific ways in which a data chunk is protected and a storage device fails. There are generally two types of storage device failures, namely, *disk failures* necessitating a disk replacement where all stored data on the failed disk are considered lost, and errors in individual data blocks that cannot be recovered with a re-read or the sector-based error-correction code (ECC), errors often referred to as *latent sector errors* [6]. While the DTR approaches provide RAID protection against one (RAID5) or two (RAID6) disk failures, they can suffer from data loss in face of multiple concurrent latent sector errors or unrecoverable read errors within a stripe [6]. On the other hand, the RCR approaches, while providing good protection for data chunks with high reference count, can suffer from file unavailability if any of a file's constituent data chunks with low reference count (*i.e.*, not replicated) are lost due to device failures, latent sector errors or unrecoverable read errors.

Existing studies on disk failure characteristics in data centers find that a significant fraction of drives (3.45%) develops latent sector errors at some point in their life [1]. Recent studies on many millions of different flash models over 6 years of production use in Google's data centers reveal that between 26-60% of flash drives encounter uncorrectable errors, between 2-6 out of 1,000 drive days experience uncorrectable errors [18]. These data block failures, rather than drive failures, suggest that providing protection against only disk failures (*e.g.*, DTR) is not sufficient to prevent file losses. This is because upon a disk failure, an unrecoverable block read error on any of the active disks during RAID5 reconstruction would lead to data loss. The same problem occurs when two disks fail under a RAID6 scheme. Similarly, only protecting data chunks with high reference counts is insufficient to prevent individual files from becoming unavailable, as discussed above. In contrast, protecting a file in its entirety before it is deduplicated is arguably much more effective in avoiding both file and data chunk failures. The reason is that by applying redundancy protection within a file, a certain number of data chunk failures can be recovered within a file depending on the redundancy scheme and a critical data chunk with a high reference count can be covered by any of the multiple parity groups belonging to as many files as the chunk's reference count.

Based on the above observations, this paper proposes Per-File Parity (PFP) to improve the reliability of deduplication-based storage systems. PFP computes the parity for every N chunks, where N is a configurable parameter, or for a whole file. When a disk failure or block failure is detected, the generated parity chunks can be used to recover from the read errors and failed data chunks by intra-file recovery. On the other hand, when several errors occur in a parity group of parity and these failed data chunks each have reference counts of greater than 1, PFP can recover these failed data chunks by inter-file recovery by leveraging the parity groups of the unaffected referencing files. As demonstrated in Section V, PFP is able to significantly improve the reliability of deduplication-based storage systems per unit of storage.

We have implemented a lightweight prototype of the PFP scheme and conducted extensive experiments to assess the system reliability and performance of PFP. The reliability results show that PFP can tolerate much more data chunk failures and guarantee file availability upon multiple data chunk failures. For example, the mean-time-to-data-loss (MTTDL) based reliability analysis shows that PFP outperforms the RCR and DTR schemes in terms of MTTDL by an average of 685.9 times and 2029.2 times respectively. Moreover, the failure-injection based evaluations show that the PFP scheme can tolerate hundreds of concurrent chunk errors without file loss for data sets with high deduplication ratios. The evaluations also show that PFP is highly cost effective in terms of file-loss-tolerance/redundancy measure by an average of 52.2% and 197.5% over the DTR and RCR schemes respectively. On the other hand, the performance assessment shows that PFP's

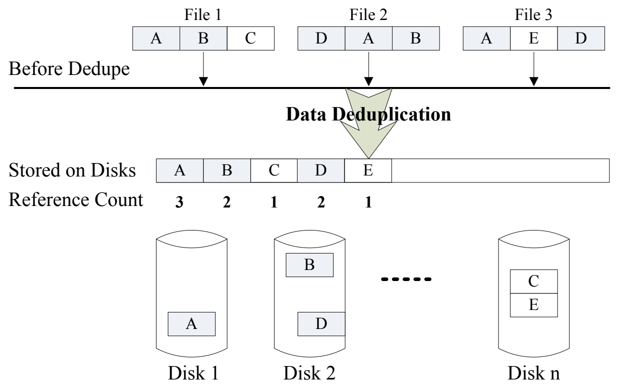


Fig. 1. Illustration of the reliability problem in deduplication-based storage.

significant reliability gain comes at an acceptable performance cost of an average of 5.7% performance degradation to the deduplication-based storage system.

The rest of the paper is organized as follows. Section II presents the background and motivation for the PFP research. The design and implementation of the PFP scheme are detailed in Section III. Section IV analyzes the reliability of PFP and Section V discusses performance results of the PFP scheme. We review the related work in Section VI and conclude this paper in Section VII.

II. Background and Motivation

In this section, we first present an illustration of the reliability problem in deduplication-based storage systems. Then we present the failure characteristics on disks to motivate our Per-File Parity study.

A. Reliability in deduplication-based storage systems

Writing the incoming files to the underlying deduplication-based storage begins with breaking the files into data chunks. Data deduplication only stores the unique data chunks, also referred to as *deduplicated data chunks*, in the underlying storage and eliminates the duplicate data chunks, also referred to as *shared data chunks*. The removed duplicate data chunks are replaced by references (pointers) to the unique data chunks, as depicted in Figure 1. Thus, some data chunks are referenced by multiple files, which implies that data deduplication will amplify the corruptive impact on files of the loss of data chunks. The existing studies generally protect the unique data chunks by replication or erasure codes after deduplication, without explicitly considering the protection of individual files.

However, from the viewpoint of an individual file, data deduplication raises the following two reliability concerns, relative to a storage system without data deduplication:

- **File reliability:** A file is split into multiple data chunks that are often spread across multiple storage devices, which can potentially decrease the file's reliability because the failure of any one of these devices or a latent sector error on any of its constituent data chunks will render the file unavailable. For example, any disk failures among Disk 1, Disk 2 and Disk n can cause File 1 to become unavailable, as shown in Figure 1.

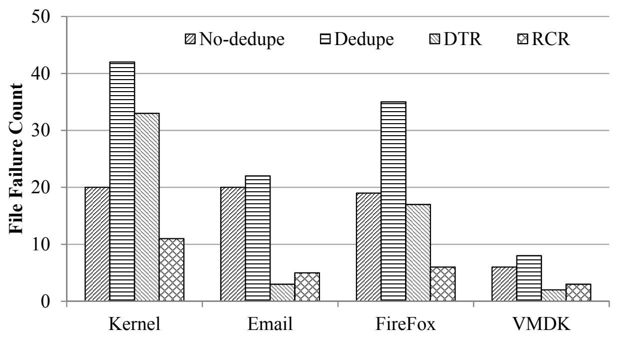


Fig. 2. Preliminary analysis comparing file losses caused by data block failures in storage systems without deduplication (No-dedupe), with deduplication but without protection (Dedupe) and with protections (DTR and RCR).

- **Chunk criticality:** The loss of a critical data chunk with a high reference count due to latent sector errors or unrecoverable errors can render all files referencing this data chunk unavailable. For example, the loss of data chunk A will cause all the three files, Files 1-3, to become unavailable, as shown in Figure 1.

Figure 2 shows our preliminary experimental analysis comparing file losses caused by data block failures in non-deduplication-based storage system, labeled *No-dedupe*, deduplication-based storage system, labeled *Dedupe*, and Dedupe systems with deduplication-then-RAID and reference-count based replication protections, labeled *DTR* and *RCR* respectively. We use the four data sets that are described in Section V, Kernel, Email, Firefox and VMDK, for this analysis. The figure shows the number of file failures for each system under each workload after injecting 20 random data block failures, indicating that the storage system with deduplication has many more file losses than the one without deduplication. The reason is simple: with the same failure count of data blocks, more files in a deduplication-based storage system than in a non-deduplication-based one will be affected because of the chunk criticality in the former. Moreover, existing data protection schemes for deduplication-based systems, DTR and RCR, failed to protect all files, resulting in a significant number of files being lost. It is clear that only protecting the unique data chunks or data chunks with high reference count fails to effectively and sufficiently protect all files from being lost.

B. Motivation

While data deduplication improves the system reliability by reducing the backup time and storage footprints, it amplifies the corruptive impact on files of data loss at the chunk level, meaning that the probability of multiple files being corrupted or lost due to individual chunk losses will increase. The existing approaches to this reliability problem of deduplication-based storage systems introduce data redundancy after files have been deduplicated, which means that these schemes are designed to protect individual unique data chunks rather than individual files.

On the other hand, extensive prior studies of disk drive failures show that data losses at the chunk level (i.e., data block failures) are much more than disk replacements (i.e., disk failures) for both HDDs and flash-based SSDs in large-scale data centers [1], [11], [18], [26]. Thus, only providing protection against disk failures (*e.g.*, deduplication-then-RAID) is not sufficient to prevent file losses in deduplication-based storage systems. Moreover, only protecting data chunks with high reference counts is insufficient to guard against unavailability of individual files. In contrast, protecting the files in their pre-deduplication forms is arguably much more effective from the point of view of individual files, which motivates our design and implementation PFP scheme, which will be elaborated in the next section.

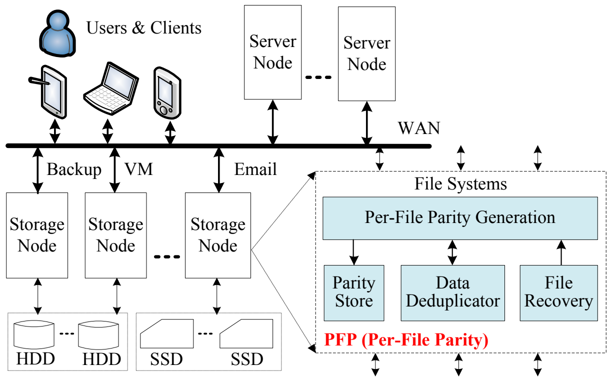


Fig. 3. System architecture of PFP.

III. The Design of PFP

In this section, we first present an architectural overview of our proposed PFP scheme, which is followed by a detailed description of the PFP data structures and workflow. Then we describe the failure recovery method in PFP.

A. PFP architectural overview

Figure 3 shows a system architecture overview of our proposed PFP scheme in the context of a datacenter system. As shown in the figure, PFP is an augmented module to the *File Systems* in the storage node. PFP interacts with the deduplication module, but is implemented independently of the lower block-level storage systems. Thus, PFP is flexible and portable for deployment in a variety of environments that can benefit from data deduplication, such as redundancy-rich virtual machine environments where the virtual machine images are mostly identical but differ in a few data blocks [12].

PFP consists of four key functional components. *Per-File Parity Generation* is responsible for splitting the incoming write data into data chunks and calculating the XOR parity chunks within each file based on a preset group size threshold. Based on the chunk size, *Data Deduplicator* applies the data deduplication functionality on the data chunks. Note that the chunking functionality of the data deduplication of Figure 1 is implemented in *Per-File Parity Generation*, while the last three functionalities of Figure 1, fingerprinting, index querying, and index/metadata updating, are implemented in *Data Deduplicator*. *Parity Store* is responsible for storing the parity chunks on the back-end storage devices. *File Recovery* is responsible

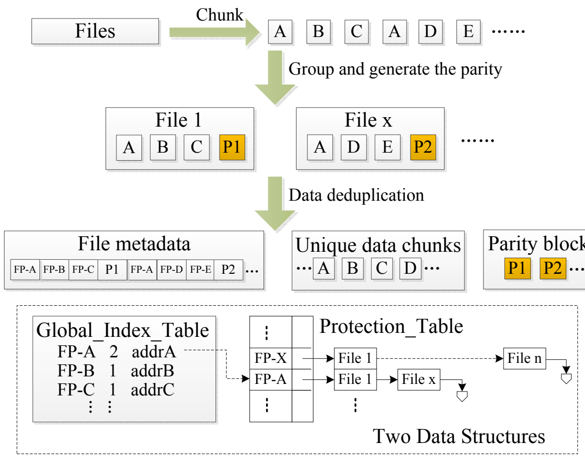


Fig. 4. Workflow of the write process in PFP.

for recovering the lost data upon unrecoverable chunk errors or disk failures via the intra-file recovery process or inter-file recovery process. It must be noticed that, in our PFP design, the parity redundancy protection scheme is used to protect the data chunks of files before these files are deduplicated. In the following subsections, we will describe the workflows of the normal operations and the failure recovery operations in detail.

B. *Workflow of normal operations*

PFP does not modify the internal workflow of data deduplication. That is, it does not change how fingerprints are calculated and how redundant data chunks are detected and removed. It applies the parity generation procedure for each file to protect the data before deduplication and stores the parity metadata as part of the file's metadata.

Figure 4 illustrates the workflow of processing a write request in PFP. PFP relies on two key data structures to deduplicate and recover the files, namely, *Global_Index_Table* and *Protection_Table*, as shown in Figure 4. Global_Index_Table keeps all the information about fingerprints, addresses and reference counts of the corresponding unique data chunks that are already stored on disks. Protection_Table maintains the mapping information between the fingerprint (FP in the figure) of each unique data chunk and a list of files that reference this data chunk. When the reference count of a unique data chunk is increased or decreased in Global_Index_Table, Protection_Table is updated accordingly. A small temporary buffer is maintained in memory to store the latest updates to Protection_Table. It is flushed and merged with the complete Protection_Table on the storage device periodically. During the intra-file recovery and garbage collection periods, Protection_Table is checked for data chunk recovery and secure deletion [5].

Upon receiving a file, the file is first chunked and the fingerprints of its data chunks are computed, by the underlying data deduplication module. PFP then divides these data chunks into different groups according to a preset group-size threshold N (*e. g.*, N = 3 as shown in Figure 4). If the last group is smaller than N, all-0 chunks are added to form a group. Note that the current design of PFP assumes that fixed size chunking is used for simplicity. Small files (e.g., files smaller than 8kB in current PFP) are usually filtered out or packed into a large file by the data deduplication module [27]. For each such N-data-chunk group, a single parity chunk is computed by using the XOR operation on the N data chunks to form an $N+1$ parity group of N data chunks and 1 parity chunk (*e. g.*, N+1 = 4 as shown in Figure 4). Such a scheme can tolerate a single chunk failure anywhere in the group. The parity in a RAID5 scheme is also based on such a single parity-check (SPC) scheme [19]. However, other XOR codes or erasure codes are also applicable to PFP to provide much higher reliability. Since the XOR parity is calculated on the write path and within a file, there is no need to fetch the data chunks that are stored on the back-end storage devices. Next, the data chunks are deduplicated to eliminate the redundant data chunks. For example, chunk A in File x is eliminated by placing a pointer in the file metadata. After deduplication, the unique data chunks and the parity chunks are written to the storage devices. And the file metadata, Global_Index_Table and Protection_Table are updated accordingly.

The workflow of processing a read request is the same as that in the standard deduplication-based storage systems, including fetching the location information from the file metadata and Global_Index_Table, then reading the corresponding data from the storage devices if there are no errors. Upon any errors during read, PFP initiates its data recovery procedure to reconstruct the lost data, which will be described in detail in Section III-C.

C. *Failure recovery*

From the viewpoint of an individual file, failure recovery incurred by data chunk failures can be classified into two categories in PFP, namely, intra-file recovery and inter-file recovery. Upon encountering a data chunk failure, PFP first checks whether the corrupted data chunk is recoverable by intra-file recovery. If the corrupted data chunk can be recovered by the intra-file recovery procedure, PFP immediately reads the other data chunks within the file and the corresponding parity chunks, calculates the lost data chunks and returns the file to the upper system. If the corrupted data chunk cannot be recovered by the intra-file recovery procedure and the reference count of the lost data chunk is greater than "1", the inter-file recovery procedure will be initiated. However, if the reference count is "1", meaning that the lost data chunk in one file is not referenced by any other file, a failure will be reported to the upper system. Obviously, the recovery of the corrupted data chunks may be iterated across multiple files. It must be noted that parity blocks can also be corrupted, which can only be detected during file recovery because they are only accessed for file recovery. During file recovery, whenever a parity block failure is detected, the recovery process is considered failed and turned to the next step, *e.g.*, transitioning from intra-file recovery to inter-file recovery or moving inter-file recovery from the current inter-file parity group to a different one to reconstruct the failed data chunks. After the completion of the file recovery, the corrupted parity block is also updated.

Intra-file recovery is a process in which the data recovery can be executed within a file upon data chunk failures. Since PFP generates parity chunks within each file, a certain number of data chunk failures, depending on the parity scheme used (*e.g.*, RAID5 and RAID6), within each file can be recovered. If a read failure is encountered on data chunk A during reading a file, the other data chunks of the parity group containing chunk A in File 1 (*i.e.*, data chunks B and C) and the corresponding parity chunk (*i.e.*, P1) are fetched to reconstruct data chunk A [20]. However, if PFP detects data chunk failures in multiple data chunks within any given parity group in File 1 that are beyond the recovery capability of the parity scheme used, *e.g.*, data chunk B and data chunk C are corrupted while under the RAID5 protection, the intra-file recovery process will be unable to recover the lost data chunks.

Clearly, if data chunk B or data block C is a critical data chunk referenced by other files, it can potentially be reconstructed within those files through inter-file recovery. Figure 5 shows an example of the inter-file recovery workflow in PFP upon multiple data chunk failures. First, when PFP finds that the data chunk failures cannot be reconstructed by intra-file recovery, the reference counts of the failed data chunks are checked in Global_Index_Table. If the reference count is “1”, the corresponding data chunk is only referenced by the current file being read and cannot be reconstructed from any other file by inter-file recovery. Otherwise, the data chunk is referenced by at least one other file and can potentially be reconstructed from other files by inter-file recovery, such as data chunk B in Figure 5. The information of the other file (*e.g.*, File n) is inquired from Protection_Table. Then data chunk B can be first reconstructed by intra-file recovery within the parity group containing data chunk B in File n. By successfully recovering data chunk B, data chunk C can also be reconstructed from the parity group containing both B and C in File 1 that is being read. Therefore, the file is recovered by both inter-file recovery and intra-file recovery. It must be noted that not all data chunk failures are recoverable. For example, if data chunks B and C in File 1 fail and their reference counts are both “1” under RAID5 protection, they cannot be recovered, thus leading to data loss.

From the procedure of inter-file recovery, we can see that data chunks with high reference counts are protected by multiple parity groups within many files. Thus, read failures on these critical data chunks do not render multiple files unavailable. The protection of the critical data chunks is similar to that of reference-count based replication schemes. Since data chunk failures can be detected by the background disk scrubbing, PFP can be incorporated into it to further improve the system reliability [6], [16].

IV. Reliability Analysis

In this section, we adopt the widely used Mean Time To Data Loss (MTTDL) metric to assess the reliability of DTR, RCR and PFP [28]. We assume that the latent sector errors

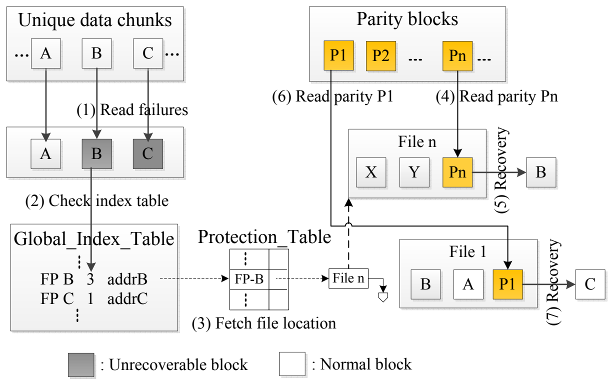


Fig. 5. The inter-file recovery workflow in PFP upon multiple chunk failures.

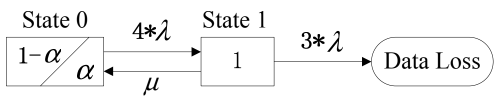
follow an exponential distribution of rate λ and repairs follow an exponential distribution of rate μ. For RCR and PFP, since the deduplication ratio is also an important factor for the reliability analysis, we assume that the deduplication ratio is α. For RCR, only the deduplicated data chunks are replicated. For simplicity and without the loss of generality, we consider the analytic model from the viewpoint of an individual file consisting of 3 data blocks and 1 parity block because of the per-file reliability emphasis and nature of this paper. This is similar to and consistent with existing models of MTTDL that, for example, consider a disk array consisting of 4 disks to keep the models tractable yet meaningful [28].

Figure 6 shows the state transition diagrams of RCR, DTR and PFP. We consider a data chunk lost if it is rendered unreadable by latent sector errors or any other forms of failure. For PFP, State $<0>$ represents the normal state of a file when its 4 chunks are all readable. State $<1>$ represents the case when one or more deduplicated data chunks (i.e., data chunks with a reference count of greater than 1) of a file are lost and State $<2>$ represents the scenario where a unique data chunk (i.e., chunk whose reference count is 1) is lost within the file. The State of Data Loss means that the file is unavailable due the unrecoverable loss of one or more of its chunks. The loss of any single deduplicated data chunk would bring the file to State $<1>$. The loss of any unique data chunk in State $<0>$ or State $<1>$ would bring the file to State $<2>$. The loss of any deduplicated data chunk in State $<1>$ or State $<2>$ will not cause file to become unavailable because these lost deduplicated data chunks can be recovered from the inter-file recovery. However, the loss of any unique data chunk in State $<2>$ would move the file to the $<$Data Loss$>$ state resulting in the file’s becoming unavailable.

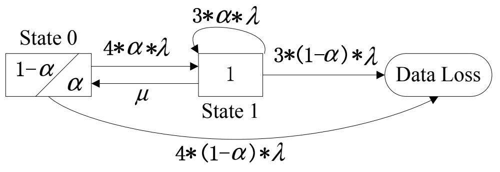
The Kolmogorov system of differential equations describing the behavior of PFP is given in Equation (1):

$$\begin{cases} \dfrac{dp_0(t)}{dt} = -4\lambda p_0(t) + \mu p_1(t) + \mu p_2(t) \\ \dfrac{dp_1(t)}{dt} = -[3(1-\alpha)\lambda + \mu]p_1(t) + 4\alpha\lambda p_0(t) \\ \dfrac{dp_2(t)}{dt} = -[3(1-\alpha)\lambda + \mu]p_2(t) + 4(1-\alpha)\lambda p_0(t) + 3(1-\alpha)\lambda p_1(t) \end{cases} \tag{1}$$

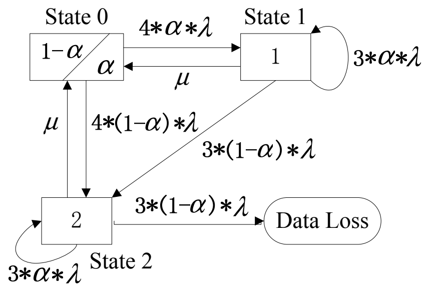
where $p_i(t)$ is the probability that the file is in state $<i>$ with the initial conditions of $p_0(0) = 1$ and $p_i(0) = 0$ for $i \neq 0$.



(a) DTR scheme



(b) RCR scheme



(c) PFP scheme

Fig. 6. State transition diagrams for the RCR, DTR and PFP schemes from the viewpoint of an individual file consisting of 3 data blocks and 1 parity block, where λ, μ and α are the error rate, repair rate and deduplication ratio respectively. Data Loss means that the file becomes unavailable.

The Laplace transform of Equation (1) is:

$$\begin{cases} sp_0^*(s) - 1 = -4\lambda p_0^*(s) + \mu p_1^*(s) + \mu p_2^*(s) \\ sp_1^*(s) = -[3(1-\alpha)\lambda + \mu] p_1^*(s) + 4\alpha\lambda p_0^*(s) \\ sp_2^*(s) = -[3(1-\alpha)\lambda + \mu] p_2^*(s) + 4(1-\alpha)\lambda p_0^*(s) + 3(1-\alpha)\lambda p_1^*(s) \end{cases} \tag{2}$$

Observing that the MTTDL of the storage system is given by [28]:

$$MTTDL = \sum_i p_i^*(0) \tag{3}$$

Using Equation (3) we solve the Laplace transform for the file being considered for $s = 0$ and use Equation (2) to compute the MTTDL of PFP:

$$MTTDL_{PFP} = \frac{[3(1-\alpha)\lambda + \mu]^2 + 4\alpha\lambda[3(1-\alpha) + \mu] + 4\lambda(1-\alpha)(3\lambda + \mu)}{12\lambda^2(1-\alpha)^2(3\lambda + \mu)} \tag{4}$$

For RCR, the loss of any unique data chunk with a reference count of 1 would result in data loss. For DTR, the occurrence of any two concurrent latent sector errors within a file would result in data loss. It must be noted that the reliability for DTR is overestimated here since we do not consider the failure amplification from the viewpoint of an individual file. That is, in practice, the loss of a single data chunk in DTR may cause multiple files to become unavailable, amplifying/multiplying the failure impact, as described in Section V-B.

Based on the state transition diagrams, the MTTDL of DTR is:

$$MTTDL_{DTR} = \frac{7\lambda + \mu}{12\lambda^2} \tag{5}$$

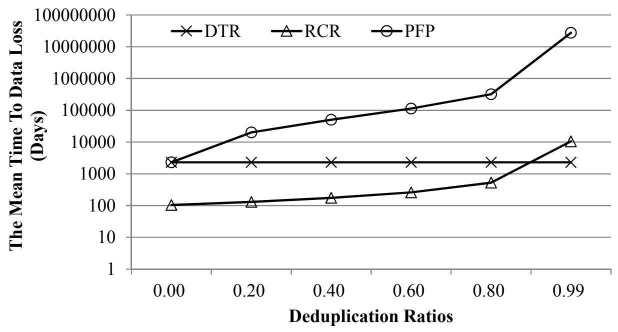


Fig. 7. MTTDL of files achieved by different schemes. Note that higher MTTDL indicates higher reliability.

The MTTDL of RCR is:

$$MTTDL_{RCR} = \frac{(3+\alpha)\lambda + \mu}{4\lambda(1-\alpha)(3\lambda + \mu)} \tag{6}$$

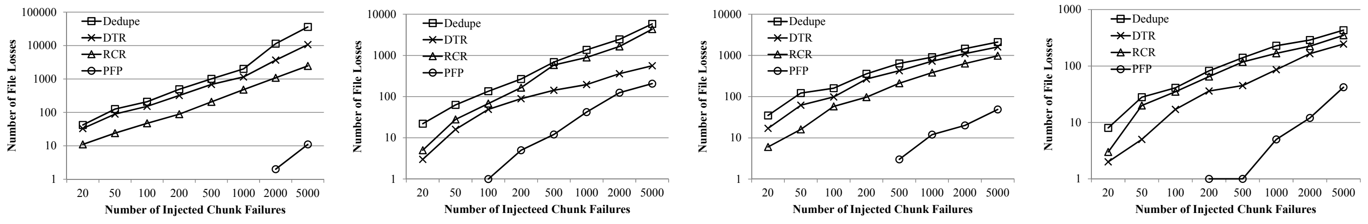
Figure 7 plots comparisons of file's MTTDLs achieved by DTR, RCR and PFP. The latent sector error rate λ is assumed to be one error occurrence every ten thousand hours, which is adopted from the failure analysis in real data sets [1], [16]. The repair of a latent sector error could be fast if it is recoverable. However, the detection of a latent sector error could take days or even up to weeks. Based on the disk scrubbing studies, the detection delay is set to be one week [16]. From Figure 7, we can see that FPF improves the storage reliability in terms of MTTDL by an average of 685.9 times and 2029.2 times over the RCR and DTR schemes respectively. Moreover, FPF increases MTTDL of files with increasing deduplication ratios. When the deduplication ratio is 0, meaning that all data chunks are unique data chunks, the MTTDLs of PFP and DTR are the same. The gap in MTTDL between PFP and DTR widens as the deduplication ratio increases, because PFP can provide better protection for the data chunks with higher reference counts.

V. Performance Evaluations

In this section, we first describe the experimental setup and methodology. Then we evaluate the reliability of PFP-optimized deduplication-based storage systems through extensive data-driven and fault injection experiments. Finally, we present the performance results and analyze the storage efficiency of PFP.

A. Experimental setup and methodology

We implement a prototype of PFP in a deduplication-based storage system and use data-driven, failure-injection based experiments to evaluate its reliability, performance and storage efficiency. Chunk failures are injected on storage devices based on the failure characteristics in real data centers [1], [13], [17]. In order to obtain a fair comparison among different schemes, the exact pattern of chunk failure injections conducted on any first comparative scheme's experiments is recorded so that the exact same failure-injection pattern is applied to all other comparative schemes' experiments. Moreover, the chunk failures are injected within a fixed storage space for all the



(a) Kernel (b) Email (c) FireFox (d) VMDK

Fig. 8. Comparisons of numbers of file losses for different systems as a function of the number of chunk failures driven by the four data sets.

TABLE I
THE CHARACTERISTICS OF THE FOUR DATA SETS

Data sets	*Dedup. ratio*	*Size (GB)*	*Number of Files*
Kernel	81.0%	330.3	29010461
Email	21.1%	424.1	5175912
FireFox	73.2%	547.9	316341
VMDK	39.5%	1621.2	12768

schemes. In other words, the experimental platform and the failure injection method are the same for all the schemes.

All the experiments were conducted on a Dell PowerEdge T320 node with an Intel Xeon E5-2407 CPU and 32GB memory. In this system, a SAMSUNG HE253GJ SATA HDD (250GB) is used to host the operating system (Ubuntu Linux kernel version 4.2.0), the Linux software RAID module (*i.e.*, MD) and other software. The LSI Logic MegaRAID SAS 2208 controller is used to connect eight SATA HDDs (Seagate ST9750420AS 7200RPM 750GB). In the experiments, four different data sets are used for the data-driven evaluation. The four data sets are downloaded from four different systems, including source codes of different Linux kernel versions (Kernel) [23], email inbox and outbox files (Email), different Firefox installation images (Firefox) [22] and Virtual Machine images (VMDK) [25]. The four data sets represent different data redundancy characteristics with a data chunk size of 4kB, as summarized in Table I.

In this paper, we compare the reliability and performance of PFP with those of the deduplication-then-RAID scheme (*DTR*) [8], reference-count based replication scheme (*RCR*) [2] and a deduplication-based storage system without any redundancy protection (*Dedupe*). In the DTR scheme, a RAID5 set with a strip size of 256kB is used to protect the unique data chunks and its parity group (stripe) size is the same as that of the PFP scheme. For the RCR scheme, a function of type $k = f(w) = min(max(1,\ a + b\ log(w)),\ k_{max})$ is used to calculate the number of replicas k depending on weight w of a data chunk. If a chunk only belongs to a single file, it is not protected in the RCR scheme. In the function used in the RCR scheme, a and b are constants that will yield different storage space utilization and robustness levels. a and b are determined experimentally and depend on the characteristics of the data set. In our experiments, we set a to "0" and b to "1". The maximum number of replicas of a data chunk is capped at k_{max} which is set to "4". These parameters are set following the example of and consistent with the original RCR study published in [2].

B. Failure injection and analysis

In the chunk failure injection model, chunk failures include data chunks, parity and replica chunks introduced by the DTR, RCR and PFP schemes. Although file metadata is also stored on storage devices, its size is much smaller than that of data chunks. Since our main focus is on the reliability impact on individual files by data chunk failures and the reliability impact of metadata loss is the same for the different schemes, we consider the reliability of metadata to be out of scope of this paper. In the reliability evaluation, we measure the numbers of files rendered unavailable by the injection of different numbers of chunk failures on the storage devices. In the DTR and PFP experiments, the parity group size is set to 4.

Figure 8 plots the number of file losses for different systems as a function of the number of chunk failures driven by the four data sets respectively. We draw several interesting observations from this evaluation. First, for a given number of chunk failures, the Dedupe scheme incurs the largest number of file losses. It is noteworthy that, in the experiments driven by the Kernel, Email and FireFox data sets, the number of file losses is even larger than the number of chunk failures. The reason is that the unavailable chunks may be referenced by multiple files in these three data sets, leading to a typical scenario in which a single chunk failure causes multiple files to become unavailable. However, for the VMDK data set, an individual file is usually very large, *i.e.*, hundreds of MB. Multiple chunk failures may cause only a single file to be unavailable. Therefore, the number of file losses is smaller than the number of chunk failures in this case.

Second, only protecting the data chunks with high reference counts (RCR) or protecting the unique data chunks (DTR) is not sufficient in protecting individual files from losses. For the RCR scheme, failures of data chunks with a reference count of 1 will cause file loss. Unlike the data sets with high deduplication ratio where there are many chunks with relatively high reference count, relatively few data chunks are of reference count of more than 1 for data sets with low deduplication ratio, such as Email and VMDK data sets. In the experiments on these two data sets, the number of file losses for the RCR scheme is larger than that for the DTR scheme. In contrast, the number of file losses for the DTR scheme is larger than that for the RCR scheme in the experiments driven by the data sets with high deduplication ratio, such as Kernel and FireFox data sets. The reason is that two or more chunk failures occurring within an individual stripe of RAID5 in the

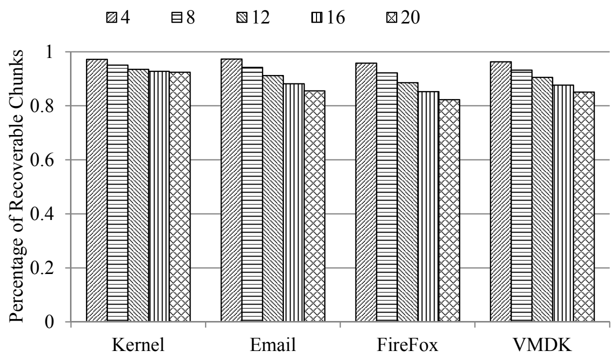


Fig. 9. The percentage of recoverable data chunks by the PFP scheme as a function of the group size, driven by the four data sets.

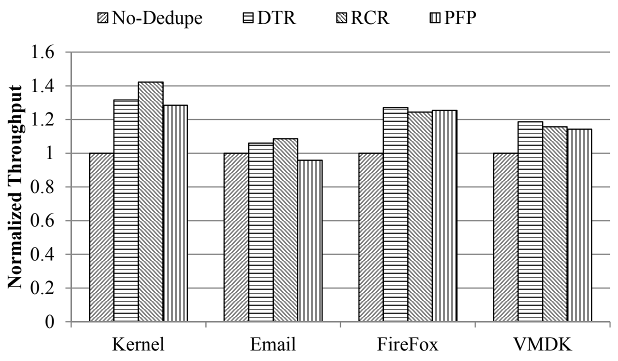


Fig. 10. The normalized (to No-Dedupe) system throughput of the different schemes driven by the four data sets.

DTR scheme will render the stripe unavailable, causing a large number of file losses. Besides, the loss of data chunks with high reference counts will cause further file losses. Therefore, for data sets with high deduplication ratios, protecting data chunks with high reference counts plays a critically important role in improving reliability of deduplication-based storage systems [4]. As a result, for the high-deduplication-ratio date sets Kernel and FireFox, the RCR scheme is shown to have fewer file losses than the DTR scheme in Figure 8(a) and Figure 8(c) respectively.

Third, the PFP scheme has the least number of file losses among the four schemes. The reasons are twofold. On the one hand, PFP calculates the parity redundancy with a small parity group size (*e.g.* 4 in the experiments) within a file. Thus, multiple parity chunks are generated for a large file. When there are multiple chunk failures within a single file, these lost chunks can be recovered by intra-file recovery if the lost chunks are relatively evenly distributed in the file. If the lost chunks are not evenly distributed, e.g., with an extreme case where there are multiple chunk failures occurring in a single parity group within a file, the lost data chunks with a high reference count (larger than 1) can still be recovered by inter-file recovery. Therefore, for data sets with high deduplication ratio, the PFP scheme can tolerate a much larger number of chunk failures without any file losses than the other schemes. The only condition under which PFP fails to prevent file losses is when at least two concurrent data chunk failures occur within a single parity group and both of the two data chunks are only referenced by a single file. The probability of this happening is somewhat low, which is demonstrated by the evaluation results driven by the four data sets. On the other hand, data chunks with a high reference count have been more securely protected by the fact that each of these critical chunks is covered by the multiple XOR parity groups of their corresponding referencing files in the PFP scheme. This makes files containing critical data chunks much less vulnerable to file losses in the PFP scheme than in the DTR scheme.

In the PFP scheme, the parity group size is an important design parameter for system reliability, cost and performance. To better understand the sensitivity of this parameter, we conduct experiments on different parity group sizes: 4, 8, 12, 16 and 20 in the PFP scheme. In the experiments, we randomly inject 100,000 data chunk failures to measure how many data chunks can be recovered. Figure 9 shows the percentage of recoverable data chunks by the PFP scheme as a function of the parity group size driven by the four data sets. The results show that the percentage of recoverable data chunks decreases as the group size increases. The reason is that the probability of two concurrent data chunk failures within a larger parity group is higher than that in a smaller group. If both failed data chunks are each referenced by a single file, the corresponding file will be lost.

C. Performance results

Figure 10 shows the normalized system throughput of the different schemes driven by the four data sets. Compared with the non-deduplication-based storage system (No-Dedupe), PFP improves the system throughput by 28.4%, -4.2%, 25.4% and 14.2% for the Kernel, Email, FireFox and VMDK data sets, respectively. The average throughput improvement is 16.0%. The DTR and RCR schemes also improve the throughput the No-Dedupe storage system by 20.8% and 22.6% on average respectively. The reason is that deduplication-based storage systems reduce the total written data significantly, compared with the non-deduplication-based storage system. Reducing write requests can reduce the queue length, thus directly improving system performance [10].

However, compared with the DTR and RCR schemes, PFP degrades the throughput by 5.7% on average. This performance overhead of PFP stems from two factors. First, the parity generation process is executed on the critical I/O path, thus affecting system performance. However, from the evaluation results, the parity generation consumes much less I/O bandwidth than the normal I/Os, which makes its impact relatively small. Second, the 0-padding within small files affects system performance. The number of small files in the Kernel and Email data sets is much larger than that in the FireFox and VMDK data sets, thus the performance degradation for the former two data sets is much higher than that for the latter two data sets. Note that PFP's inferior performance to No-Dedupe stems from the fact that the deduplication ratio of the Email data set is only about 21.1%, which causes PFP to

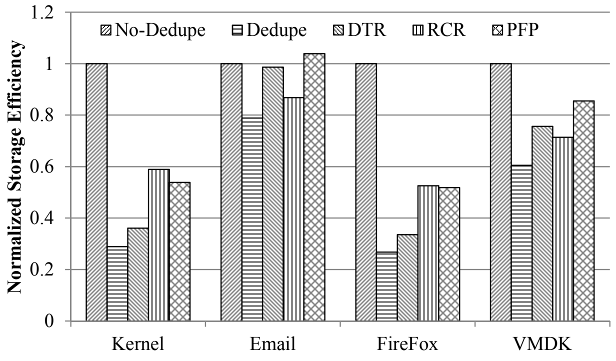


Fig. 11. The normalized (to No-Dedupe) storage efficiency of different schemes driven by the four data sets.

incur much more storage overhead relative to No-Dedupe and underperform No-Dedupe in system throughput.

D. Storage efficiency

The primary objective of data deduplication is to reduce storage space requirement thus improving storage efficiency. However, applying redundancy-based protection schemes in deduplication-based storage systems re-introduce data redundancy, which may negatively affect the storage efficiency. Figure 11 shows the normalized storage efficiency of the different schemes driven by the four data sets. Storage efficiency in this context is defined to be the total data volume written to storage devices divided by the total data volume arrived at the file system, thus the lower the better. We assume that the storage efficiency is 1 for the No-Dedupe system. In this evaluation, due to the small size of metadata and index, relative to data chunks, we do not consider the storage overhead introduced by metadata and fingerprint index. As before, the parity group size in the DTR and PFP experiments is set to 4.

A few key conclusions are drawn from the evaluation. First, deduplication-based storage systems outperform the non-deduplication-based storage systems in terms of storage efficiency, which demonstrates the advantage of data deduplication. Second, DTR, RCR and PFP introduce much more storage overhead than the Dedupe system. For data sets with high deduplication ratios, such as Kernel and FireFox, the RCR scheme introduces much more redundancy than the DTR scheme. Third, the PFP scheme introduces less storage overhead than the RCR scheme for the Kernel and FireFox data sets. However, PFP introduces the largest storage overhead for the Email and VMDK data sets. The reason is that PFP applies the data redundancy protection on the original data set that is larger than the deduplicated data set. Nevertheless, PFP significantly improves the storage efficiency of the No-dedupe system, by 46.1%, 48.2% and 14.5% for the Kernel, FireFox and VMDK data sets, respectively. Although PFP degrades the storage efficiency by 3.9% for the Email data sets, it significantly outperforms the other schemes in terms of system reliability. In conclusion, we argue that PFP's significant advantage in system and file reliability over other deduplication-based redundancy schemes DTR and RCR strongly justifies its relatively small additional space overhead.

VI. Related Work

Data deduplication as a space-efficient technique has received a great deal of attention from both industry and academia [27]. However, recent studies have suggested that other important deduplication-specific problems such as reliability have not been adequately addressed in the literature and should be seriously considered [21].

The existing schemes addressing the reliability problem in deduplication-based storage systems can be classified into two categories, reference-count based replication (RCR) and deduplication-then-RAID (DTR) [27]. To the best of our knowledge, Bhagwat et al. [2] are the first to address the reliability concern in deduplicated storage systems. They observe that deduplication alters the reliability of the stored data due to the sharing of common data chunks. Thus, they use replication-based storage to store the data chunks with high reference counts and argue that the number of copies of a data chunk should be logarithmic to the reference count of the data chunk. HYDRAstor [3] is a deduplicated secondary storage system that allows data chunks to be placed in different resilience classes, each of which has a different level of reliability. However, which resilience class to store each data chunk is the responsibility of the user and has no relationship to the sharing degree of the data chunks. Recently, Fu et al. [4] propose a deliberate copy technique that allocates a small dedicated physical area to store the data chunks with high reference counts and first repairs the dedicated physical area during RAID reconstruction upon failure. The reference-count based replication methods only consider the importance of the data chunks with high reference counts to alleviate the impact of corrupted files caused by the loss of these data chunks. However, from the point of view of a file, any loss of a file's constituent data chunks that are spread across multiple disks will render the file unavailable. Thus, only protecting the data chunks with high reference counts is not sufficient.

On the other hand, the DTR schemes directly apply a certain erasure code on the unique data chunks after data deduplication. DDFS [29] improves the reliability of deduplication storage by applying the software RAID-6 protection method to the unique data chunks. Liu et al. [8] suggest that variable-size chunking, *i.e.*, CDC, should be preferred over fixed-size chunking because the former has proven to yield more space savings. The variable-size data chunks are first packed into bigger fixed-size objects for erasure-code protection and then stored on multiple storage nodes. HP-KVS [7] allows each object to specify its own reliability level and uses the software erasure coding to improve the data reliability. However, the importance of all the unique data chunks is considered equally in the DTR schemes. Since the same erasure code is used on all unique data chunks after deduplication, the criticality of the data chunks with high reference counts is not considered.

In summary, the existing solutions focus on only one of the reliability-related parameters: either the criticality of losing a chunk (RCR) or the probability of losing any chunk (DTR). Different from these schemes, our proposed PFP scheme improves the reliability of deduplication-based storage systems from the point of view of an individual file by coding the data chunks of a file before they are deduplicated. Thus, the loss of up to a certain number of the constituent data chunks of a file does not cause file loss, depending on the erasure code used and the deduplication ratio. Moreover, since the parity is generated before deduplication and each file has its own parity redundancy, the data chunks with high reference counts have multiple parity protections. Thus, both the criticality of the data chunks with high reference counts and the probability of losing any given data chunk to cause a file failure are considered in the PFP design.

Some studies also analyze the reliability of deduplication-based storage systems theoretically. Li et al. [7] propose combinatorial analysis of deduplication and erasure coding to evaluate the system reliability. Rozier et al. [14], [15] design and implement a modeling framework to evaluate the reliability of a deduplication-based storage system with different hardware and software configurations. Inspired by and based on these theoretical analyses, we build an MTTDL-based model to evaluate the reliability of different protection schemes. Further, different from these theoretical analyses, we build a system for deduplication-based storage systems, populated with real data sets, to analyze the system reliability under different situations of data corruption. We also conduct experiments to examine the system throughput and storage efficiency.

VII. Conclusion

Data deduplication has been widely used to improve the storage efficiency in modern primary and secondary storage systems. While increasingly important, the reliability issue of deduplication-based storage systems has not received sufficient attention. In this paper, we propose a per-file parity (PFP) scheme to improve the reliability of deduplication-based storage systems. PFP computes the parity for each parity group of N chunks ($N-1$ data chunks and 1 parity chunk, where N is configurable) within each file before the file is deduplicated. Therefore, PFP can provide redundancy protection for all files by intra-file recovery as well as a higher level of protection for data chunks with high reference counts, critical data chunks, by inter-file recovery. The reliability analysis and evaluations show that PFP can provide better reliability than the state-of-the-art DTR and RCR schemes.

VIII. Acknowledgments

We are grateful to all the members of the ASTL group for their continuous support and discussion. This work is supported by the China National Natural Science Foundation No. 61772439, No. U1705261, and No. 61472336, the US NSF under Grant No. CCF-1704504 and CCF-1629625.

References

[1] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *SIGMETRICS'07*, Jun. 2007.

[2] D. Bhagwat, K. Pollack, D. Long, T. Schwarz, E. Miller, and J.-F. Pâris. Providing High Reliability in a Minimum Redundancy Archival Storage System. In *MASCOTS'06*, Sept. 2006.

[3] U. Cristian, A. Benjamin, A. Akshat, G. Salil, R. Stephen, C. Grzegorz, D. Cezary, and B. Aniruddha. HydraFS: a High-throughput File System for the HYDRAstor Content-addressable Storage System. In *FAST'10*, Feb. 2010.

[4] M. Fu, Patrick P. C. Lee, D. Feng, Z. Chen, and Y. Xiao. A Simulation Analysis of Reliability in Primary Storage Deduplication. In *IISWC'16*, Sept. 2016.

[5] F. Guo and P. Efstathopoulos. Building a High-performance Deduplication System. In *USENIX ATC'11*, Portland, OR, Jun. 2011.

[6] I. Iliadis, R. Haas, X. Hu, and E. Eleftheriou. Disk Scrubbing versus Intra-disk Redundancy for High-Reliability RAID Storage Systems. In *SIGMETRICS'08*, Annapolis, MD, Jun. 2008.

[7] 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.

[8] 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 *ICS'09*, Yorktown Heights, NY, Jun. 2009.

[9] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian. Read Performance Optimization for Deduplication-based Storage Systems in the Cloud. *ACM Transactions on Storage*, 10(2):1–22, 2014.

[10] B. Mao, H. Jiang, S. Wu, and L. Tian. POD: Performance Oriented I/O Deduplication for Primary Storage Systems in the Cloud. In *IPDPS'14*, May. 2014.

[11] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. A Large-Scale Study of Flash Memory Failures in the Field. In *SIGMETRICS'15*, pages 177–190, Portland, Oregon, USA, Jun. 2015.

[12] C. Ng, M. Ma, T. Wong, Patrick P. C. Lee, and John C. S. Lui. Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud. In *Middleware'11*, Dec. 2011.

[13] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure Trends in a Large Disk Drive Population. In *FAST'07*, Feb. 2007.

[14] E. Rozier and W. Sanders. A Framework for Efficient Evaluation of the Fault Tolerance of Deduplicated Storage Systems. In *DSN'12*, Boston, Massachusetts, Jun. 2012.

[15] E. Rozier, W. Sanders, P. Zhou, and N. Mandagere. Modeling the Fault Tolerance Consequences of Deduplication. In *SRDS'11*, Madrid, Spain, Oct. 2011.

[16] B. Schroeder, S. Damouras, and P. Gill. Understanding latent sector errors and how to protect against them. In *FAST'10*, San Jose, CA, Feb. 2010.

[17] B. Schroeder and G. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *FAST'07*, Feb. 2007.

[18] B. Schroeder, R. Lagisetty, and A. Merchant. Flash Reliability in Production: The Expected and the Unexpected. In *FAST'16*, pages 67–80, San Jose, CA, Feb. 2016.

[19] Z. Shen, J. Shu, and Y. Fu. Seek-Efficient I/O Optimization in Single Failure Recovery for XOR-coded Storage Systems. In *SRDS'15*, Sep. 2015.

[20] Z. Shen, J. Shu, and Patrick P. C. Lee. Reconsidering Single Failure Recovery in Clustered File Systems. In *DSN'16*, Jun. 2016.

[21] P. Shilane, R. Chitloor, and U. Jonnala. 99 Deduplication Problems. In *HotStorage'16*, Jun. 2016.

[22] The FireFox install images. https://www.mozilla.org/en-US/firefox/new/.

[23] The Linux Kernel Archives. https://www.kernel.org/.

[24] J. Tucci. Cloud + Big Data = Massive Change, Massive Opportunity, Keynote at UW CSE Annual Industrial Affiliates Meeting. Oct 2010.

[25] VMware Virtual Appliance Marketplace. http://www.vmware.com/appliances/.

[26] G. Wang and W. Xu. What Can We Learn from Four Years of Data Center Hardware Failures? In *DSN'17*, Jun. 2017.

[27] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou. A Comprehensive Study of the Past, Present, and Future of Data Deduplication. *Proceedings of the IEEE*, 104(9):1681–1710, 2016.

[28] Q. Xin, E. Miller, T. Schwarz, D. Long, S. Brandt, and W. Litwin. Reliability Mechanisms for Very Large Storage Systems. In *MSST'03*, Apr. 2003.

[29] B. Zhu, K. Li, and H. Patterson. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In *FAST'08*, Feb. 2008.