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

Suzhen Wu, *Member, IEEE*, Bo Mao, *Member, IEEE*, Hong Jiang, *Fellow, IEEE*, Huagao Luan, and Jindong Zhou

Abstract—Data deduplication weakens the reliability of storage systems since by design it removes duplicate data chunks common to different files and forces these files to share a single physical date chunk, or critical chunk, after deduplication. Thus, the loss of a single such critical data chunk can potentially render all referencing (sharing) files unavailable. However, 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 percent performance degradation to the deduplication-based storage system.

Index Terms—Data deduplication, reliability, per-file parity, intra-file recovery, inter-file recovery

1 Introduction

It is the most critical challenges for the design and management of large-scale storage systems in face of the explosive growth in data volume. Consequently, data reduction technologies have been propelled to the forefront of research and development in addressing this challenge in the big data era [1]. Data deduplication, a space efficient data reduction technology, has spurred a great deal of research interest from both industry and academia [2], [3], [4]. It has been deployed in a wide range of storage systems, including backup and archiving systems [5], [6], [7] and primary storage systems [8], [9] such as Virtual Machine (VM) servers [10].

The 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 [2], [4]. 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 [3], [11]. 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.

Existing approaches to address the reliability problem in deduplication-based storage systems can be classified into two categories [4], 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

- *S. Wu and H. Luan are with Computer Science Department, Xiamen University, Xiamen, Fujian 361005, China. E-mail: suzhen@xmu.edu.cn, 1508331522@qq.com.*
- *B. Mao and J. Zhou are with the Software School, Xiamen University, Xiamen, Fujian 361005, China. E-mail: maobo@xmu.edu.cn, zhoujindddd@qq.com.*
- *H. Jiang is with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX 76019. E-mail: hong.jiang@uta.edu.*

Manuscript received 1 Nov. 2018; revised 2 Feb. 2019; accepted 6 Feb. 2019. Date of publication 12 Feb. 2019; date of current version 7 Aug. 2019. (Corresponding author: Suzhen Wu.) Recommended for acceptance by J. Wang PhD. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2019.2898942

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* [12]. 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 [12]. 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.

Previous studies on disk failure characteristics in data centers find that a significant fraction of drives (3.45 percent) develops latent sector errors at some point in their life [13]. 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 percent of flash drives encounter uncorrectable errors, between 2-6 out of 1,000 drive days experience uncorrectable errors [14]. 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 5, PFP is able to significantly improve the reliability of deduplication-based storage systems per unit of storage.

We have conducted extensive experiments on a light-weight prototype implementation of the PFP scheme to assess the system reliability and performance. 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 percent over the DTR and RCR schemes, respectively. On the other hand, the performance assessment shows that PFP's significant reliability gain comes at an acceptable performance cost of an average of 5.7 percent performance degradation to the deduplication-based storage system.

The rest of the paper is organized as follows. Section 2 presents the background and motivation for the PFP research. The design and implementation of the PFP scheme are detailed in Section 3. Section 4 analyzes the reliability of PFP and Section 5 discusses performance results of the PFP scheme. We review the related work in Section 6 and discuss insights we obtained through this work in Section 7. The conclusion and future work of this paper are presented in Section 8.

2 Background and Motivation

In this section, we first present the basic knowledge about data deduplication, followed by an illustration of the reliability problem induced by data deduplication. Then we present the failure characteristics on disks to motivate our Per-File Parity study.

2.1 Data Deduplication

Data deduplication is a data reduction technology that aims at saving the storage space by removing data redundancy in storage systems. With the explosive growth in data volumes, data deduplication has been increasingly widely used in backup, archiving and network transmission applications. Recently, it has found applications in primary storage systems, such as flash memory systems and virtual machine environments.

Fig. 1 shows the data deduplication workflow consisting of four major steps [4], [15]. In Step (1), called *data chunking*, files or data streams are divided into data chunks, either in fixed size (a.k.a. *fixed chunking* (FC)) or in variable size based on data content (a.k.a. *content defined chunking* (CDC)). In general CDC helps detect more data redundancy but at the cost of much higher compute overhead than FC. In Step (2), a secure hash function (e.g., SHA1, MD5 and SHA256) is applied to each data chunk to generate its unique fingerprint, hence the name *fingerprinting* for this step. Fingerprints of the incoming data chunks are then compared with those of

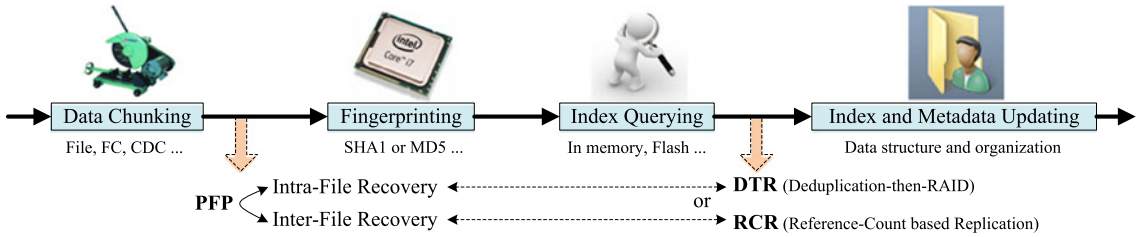


Fig. 1. The data deduplication workflow consisting of four major steps and inserted locations of different protection schemes.

already stored chunks to determine if any of the new comers are duplicates, by querying the indexed fingerprints in Step (3), called *index query*. A matched fingerprint in this step indicates that the corresponding incoming data chunk is a duplicate. This step detects the duplicate data chunks by simply checking their fingerprints, which avoids a costly byte-by-byte comparison. Finally, in Step (4), referred to as *index and metadata updating*, the unique data chunks, whose fingerprints failed to match any indexed ones, are written on the storage devices, the index is updated with the new fingerprints, and the duplicate data chunks are replaced by the pointers (to their corresponding stored unique chunks) and are updated in the metadata store.

The advantage of data deduplication is obvious. It reduces the storage space and network bandwidth needed by applications, thus reducing the total cost. However, this advantage comes at some costs. For example, it requires extra computing and memory resources. Most importantly, data deduplication also introduces a reliability problem. In deduplication-based storage systems, a data chunk may be shared by several different files. Once a data chunk that is shared by multiple files is lost due to device failures or unrecoverable errors, it can adversely impact the integrity of all these files, which is elaborated in Section 2.2.

2.2 Reliability in Deduplication-Based Storage Systems

Deduplication splits the incoming files written to the underlying deduplication-based storage into data chunks and only stores the unique data chunks, also referred to as *deduplicated data chunks*. It eliminates the duplicate data chunks, also referred to as *shared data chunks*, and replaces them by references (pointers) to the unique data chunks, as depicted in Fig. 2. 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.

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 Fig. 2.
- *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 Fig. 2.

Fig. 3 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 5, 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

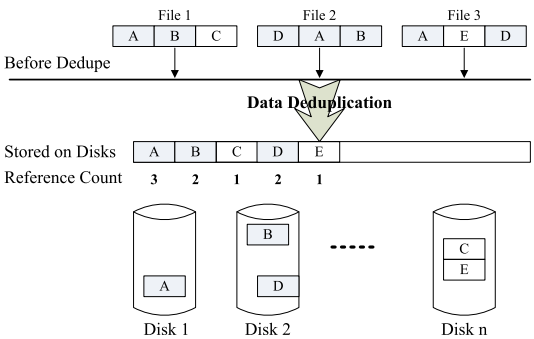


Fig. 2. Illustration of the reliability problem in deduplication-based storage systems.

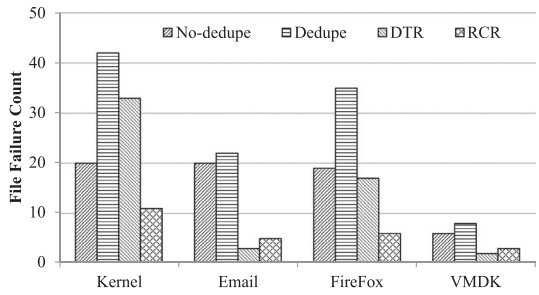


Fig. 3. Preliminary analysis comparing file losses caused by data block failures in storage systems without deduplication (No-dedupe), with deduplication but without protection (Dedupe) and with protections (DTR and RCR).

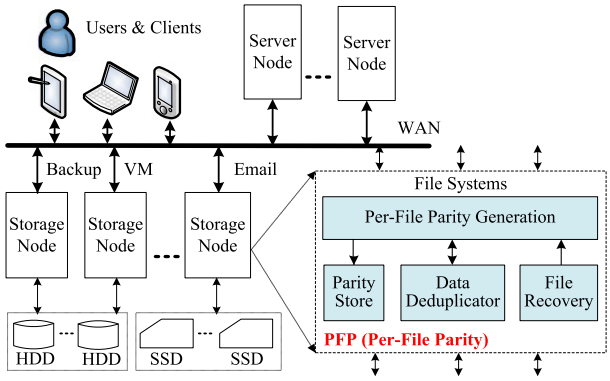


Fig. 4. System architecture of PFP.

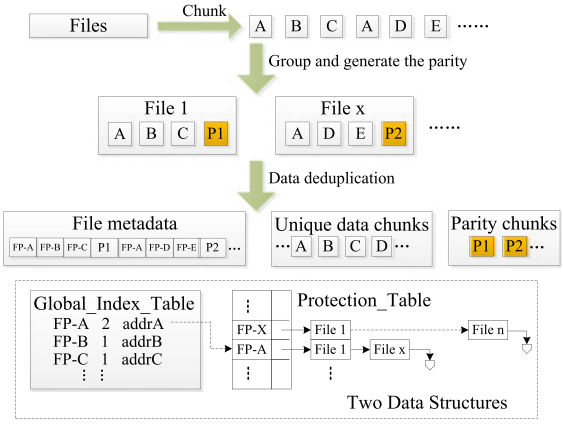


Fig. 5. Workflow of the write process in PFP.

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.

2.3 Motivation

Data deduplication has been widely deployed in both primary and secondary storage systems. While 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, as shown in Fig. 1, 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 [13], [14], [16], [17]. 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 (as shown in Fig. 1) is arguably much more effective from the point of view of individual files, which motivates our design and implementation PFP scheme.

3 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.

3.1 PFP Architectural Overview

The main design objective of PFP is to improve the reliability of deduplication-based storage systems. Fig. 4 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 [18].

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 Fig. 1 is implemented in *Per-File Parity Generation*, while the last three functionalities of Fig. 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 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 sections, we will describe the workflows of the normal operations and the failure recovery operations in detail.

3.2 Workflow of Normal Operations

To improve the portability, 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.

Fig. 5 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 Fig. 5. 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 [19].

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 Fig. 5). 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 8 kB in current PFP) are usually filtered out or packed into a large file by the data deduplication module [4]. 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 Fig. 5). 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. 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.

In the normal state if there are no errors, 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. Upon any errors during read, PFP initiates its data recovery procedure to reconstruct the lost data, which will be described in detail in Section 3.3.

3.3 Failure Recovery

Failure recovery plays an important role in the design of storage systems and is one of the most important factors in improving the storage system reliability [20]. Different from data chunk failures, disk failures are easy to detect as the disks are embedded with mature failure detection/prediction mechanisms such as SMART information [21] that directly report failures to the upper-layer systems. When a disk failure occurs, the device-level RAID recovery process is initiated immediately and sometimes even proactively [20]. However, data chunk failures, such as latent sector errors, are not so easily detected unless they are being read (e.g., via ECC or read error) [22]. This means that such data chunk errors might never be detected if the corresponding data chunks are not read. Detecting the corrupted data chunks at an early stage is extremely important because these chunks would then be recovered by the device-level RAID capability. However, if the data chunk failures are detected during the RAID recovery period, data loss becomes unavoidable. Existing studies have shown that data chunk failures happen at a significant rate in the field and are expected to become more frequent with the increasing disk capacities. Thus, preventing data loss caused by data chunk failures becomes increasingly important in guaranteeing the storage reliability [22].

Algorithm 1 shows a pseudo-code of the failure recovery procedure in PFP. From the viewpoint of an individual file, failure recovery triggered 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 reconstructed 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 chunks 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 chunk 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 chunk is also updated.

Algorithm 1. Failure Recovery Algorithm in PFP

```
1: Initialization:
2: A represents a data chunk.
3: file represents the file containing data chunk A.
4: ref_file represents a different file containing data A.
5: function RECOVERY(A)
6:   if re_read(A) == ERROR then
7:     if file_read(file) == OK then
8:       intra_recovery(A, file)
9:       Return A
10:    else
11:      if reference_count(A)> 1 then
12:        inter_recovery(A, ref_file)
13:        Return A
14:      else
15:        Return FAILURE
16:      end if
17:    end if
18:  else
19:    Return A
20:  end if
21: end function
```

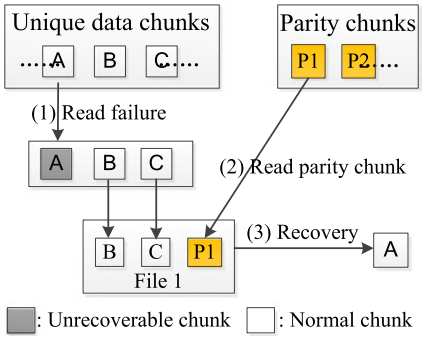


Fig. 6. Workflow of the intra-file recovery in PFP upon a data chunk failure assuming the XOR parity scheme.

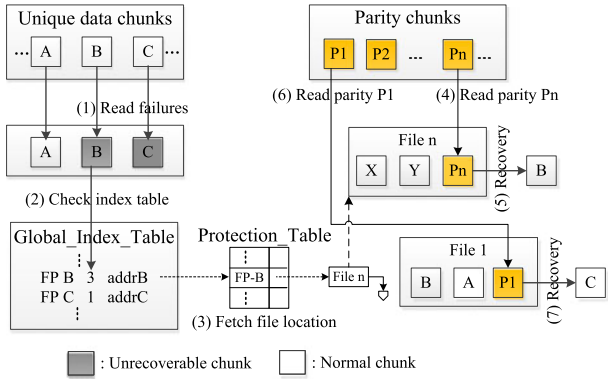


Fig. 7. Workflow of the inter-file recovery in PFP upon multiple data chunk failures.

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 of a given file, depending on the parity scheme used (e.g., RAID5 and RAID6), can be recovered. Fig. 6 shows an example of the intra-file recovery workflow in PFP upon a data failure. 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. However, if PFP detects multiple data chunk failures in 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 chunk C is a critical data chunk referenced by other files, it can potentially be reconstructed within those files through inter-file recovery.

Fig. 7 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 the other referencing file (s) by inter-file recovery, such as data chunk B in Fig. 7. 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 from 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.

Clearly, data chunks with high reference counts are protected by multiple parity groups in as many files from the procedure of inter-file recovery. Thus, read failures on these critical data chunks do not render multiple files unavailable. The design objective for the critical data chunks is similar to that of reference-count based replication schemes. Since data chunk failures are detected by reading involved files on-demand or by the background disk scrubbing, PFP can be incorporated into the background disk scrubbing schemes to further improve the system reliability [12], [22].

4 Reliability Analysis

In this section, we adopt the widely used Mean Time To Data Loss metric to assess the reliability of DTR, RCR and PFP [23]. We assume that the latent sector errors 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 chunks and 1 parity chunk 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 [23].

Fig. 8 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.

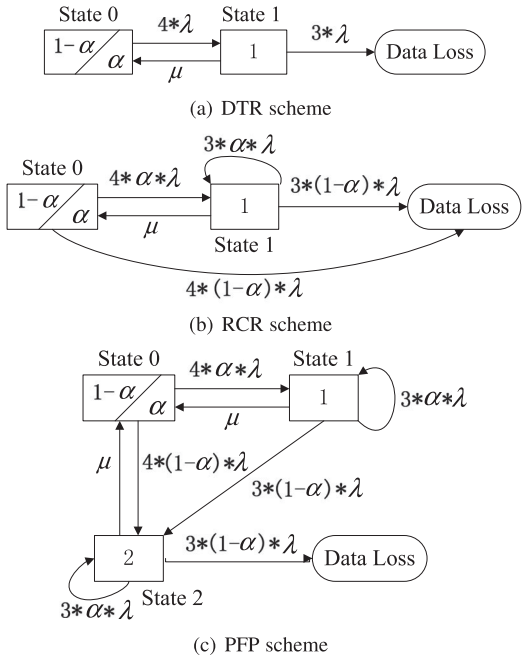


Fig. 8. State transition diagrams for the RCR, DTR and PFP schemes from the viewpoint of an individual file consisting of 3 data chunks and 1 parity chunk, where λ, μ and α are the error rate, repair rate and deduplication ratio, respectively. Data Loss means that the file becomes unavailable.

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

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

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 $i \neq 0$.

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 [23]

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

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 5.2.

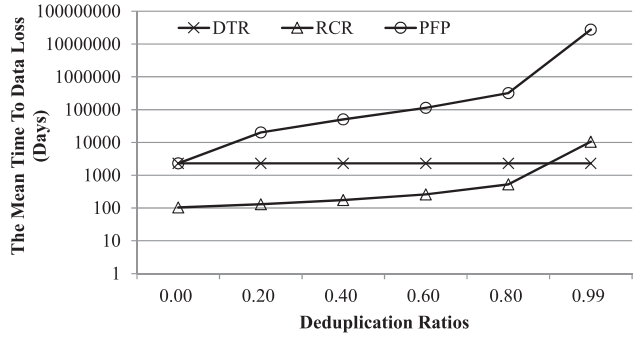


Fig. 9. MTTDL of files achieved by different schemes. Note that higher MTTDL indicates higher reliability.

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

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

The MTTDL of RCR scheme is

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

Fig. 9 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 [13], [22]. 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 [22]. From Fig. 9, 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.

5 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.

5.1 Experimental Setup and Methodology

We implement a prototype of PFP by integrating it into the open-source data deduplication project called SDFS [24], [25].

TABLE 1
The Characteristics of the Four Data Sets

Data sets	*Dedup. ratio*	*Size (GB)*	*Number of Files*
Kernel	81.0%	330.3	29,010,461
Email	21.1%	424.1	5,175,912
FireFox	73.2%	547.9	316,341
VMDK	39.5%	1621.2	12,768

The Deduplication Storage Engine (DSE) module within the SDFS volume is enhanced to store, read and recover all unique data chunks. We use data-driven, failure-injection based experiments to evaluate the reliability, performance and storage efficiency. Chunk failures are injected on storage devices based on the failure characteristics in real data centers [13], [26], [27]. 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 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 (250 GB) 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 750 GB). 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) [28], email inbox and outbox files (Email), different Firefox installation images (Firefox) [29] and Virtual Machine images (VMDK) [30]. The four data sets represent different data redundancy characteristics with a data chunk size of 4 kB, as summarized in Table 1.

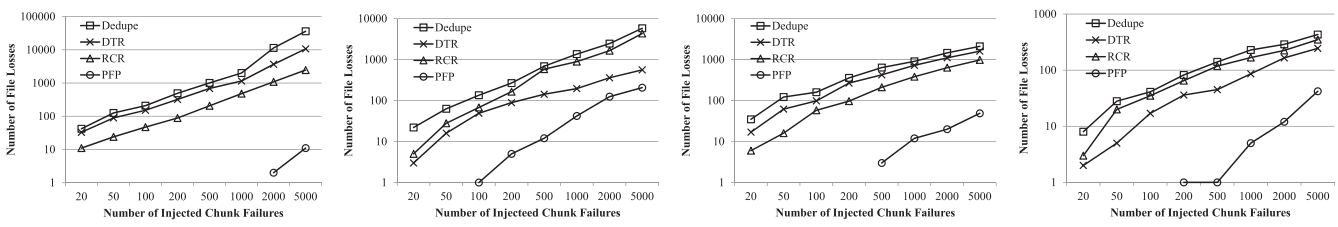
We compare the reliability, performance and storage efficiency of PFP with those of the deduplication-then-RAID scheme (*DTR*) [31], reference-count based replication scheme (*RCR*) [32] and a deduplication-based storage system without any redundancy protection (*Dedupe*). In the DTR scheme, a RAID5 set with a strip size of 256 kB 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 $k = f(w) = min(max(1, a + b\ log(w)), \mathrm{k}_{max})$ 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 [32].

5.2 Failure Injection and Analysis

We use the failure injection method to emulate the chunk failures. 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.

Fig. 10 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, 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.

Fig. 11 shows the percentage breakdown of files recovered by either intra-file recovery alone or joint intra- & inter-file recovery with 100 injected chunk failures in the PFP scheme. The results show that a majority of files can be recovered by the intra-file recovery. However, the inter-file recovery is



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

Fig. 10. Comparisons of numbers of file losses for different systems as a function of the number of chunk failures driven by the four data sets.

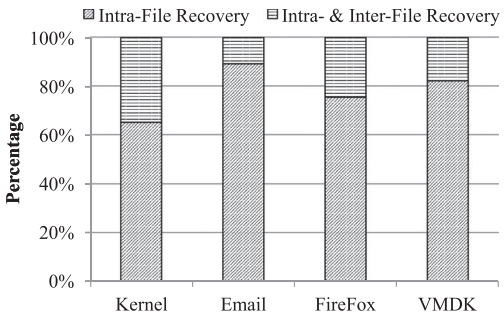


Fig. 11. The percentage breakdown of files recovered by either intra-file recovery alone or joint intra- & inter-file recovery with 100 injected chunk failures in the PFP scheme.

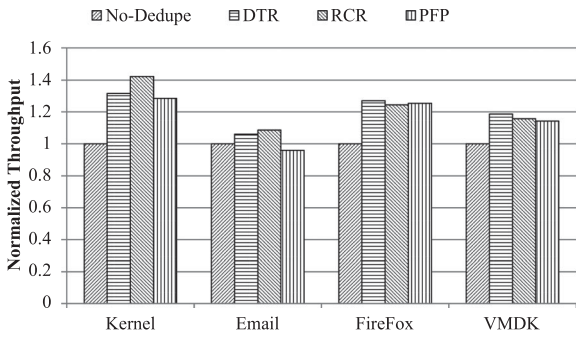


Fig. 13. The normalized (to No-Dedupe) system throughput of the different schemes driven by the four data sets.

also important and contributes to recovering files in the Kernel and FireFox data sets due to their higher data redundancy. 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.

Second, 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.

Third, 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 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 [11]. 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 Figs. 10a and 10c, respectively.

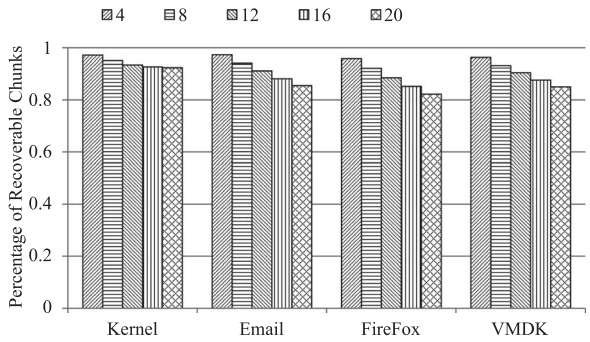


Fig. 12. The percentage of recoverable data chunks by the PFP scheme as a function of the group size, driven by the four data sets.

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. Fig. 12 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.

5.3 Performance Results

Fig. 13 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 percent for the Kernel, Email, FireFox and VMDK data sets, respectively. The average throughput improvement is 16.0 percent. The DTR and RCR schemes also improve the throughput over the No-Dedupe storage system by

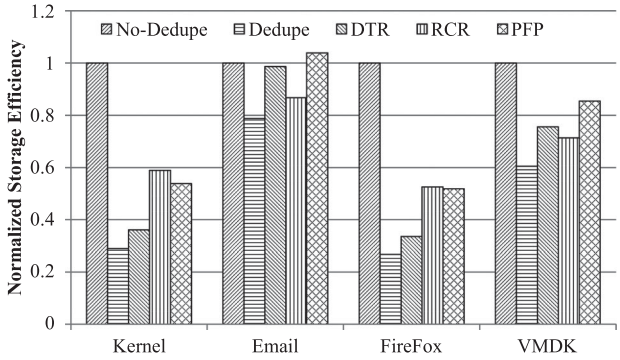


Fig. 14. The normalized (to No-Dedupe) storage efficiency of different schemes driven by the four data sets.

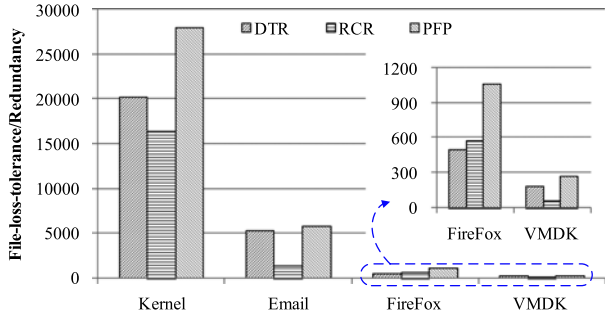


Fig. 15. Number of file losses prevented per added data redundancy driven by the four data sets. The Y-axis shows the difference between the number of file losses under Dedupe and that under the given protection scheme DTR/RCR/PFP.

20.8 and 22.6 percent 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 [33].

However, compared with the DTR and RCR schemes, PFP degrades the throughput by 5.7 percent 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 percent, which causes PFP to incur much more storage overhead relative to No-Dedupe and underperform No-Dedupe in system throughput.

5.4 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. Fig. 14 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 percent for the Kernel, FireFox and VMDK data sets, respectively. Although PFP degrades the storage efficiency by 3.9 percent 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.

5.5 File-Failure-Tolerance Efficiency

How much data redundancy added to the system can directly affect the efficiency of failure tolerance. In the other words, for the same amount extra redundancy added, the more file failures tolerated by a given scheme, the more efficient the scheme is. Thus, a fair and reasonable comparison metric for the DTR, RCR and PFP schemes should be such efficiency, namely, the number of file failures (by the conventional deduplication scheme) tolerated divided by the amount of data redundancy added on top of the conventional deduplication system. That is, out of the file failures induced by the conventional deduplication scheme, how many are prevented (tolerated) from rendering actual data loss by the added data redundancy of a given scheme? Obviously, the higher this metric is, the better the scheme will be.

Fig. 15 shows the number of file losses from the conventional data deduplication scheme (Dedupe) that are prevented from rendering data loss per added data redundancy, driven by the four data sets on the condition of 5,000 injected chunk failures. For each dataset, the file-failure tolerance values for all the schemes are measured based on the same amount of added redundancy, that of the DTR scheme. It is clear that PFP achieves the highest file-failure tolerance per unit of added redundancy among all the schemes. It improves the file-failure-tolerance efficiency measure by an average of 52.2 and 197.5 percent over the DTR and RCR

TABLE 2
Summary of the Related Studies to PFP

Schemes	*File reliability*	*Chunk criticality*	*Performance*	*Method*
RCR [32]	No	Good	Good	Reference-count based replication
DCT [11]				Deliberate copy technique
DDFS [5]		Weak	Moderate	RAID6 on unique data chunks
R-ADMAD [31]				Erasure-code protection on unique and grouped data chunks
PFP	Good (Intra-file recovery)	Good (Inter-file recovery)	Moderate	Per-file XOR-based parity before deduplication

schemes, respectively. Although PFP adds a little more data redundancy than the DTR and RCR schemes, the former is shown to be able to tolerate significantly more file failures than the latter. In other words, PFP trades a slightly increased data redundancy (over DTR and RCR) for vastly improved file-loss protection.

6 Related Work

Data deduplication as a space-efficient technique has received a great deal of attention from both industry and academia. While more details about data deduplication research and development can be found in the two survey articles [2], [4], 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 [3].

The existing schemes addressing the reliability problem in deduplication-based storage systems can be classified into two categories, reference-count based replication and deduplication-then-RAID [4]. To the best of our knowledge, Bhagwat et al. [32] 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 [34] 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. [11] 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 [5] improves the reliability of deduplication storage by applying the software RAID-6 protection method to the unique data chunks. Liu et al. [31] 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 [35] 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.

Table 2 presents a summary of the existing studies most relevant to the PFP scheme. The existing solutions focus on only one of the reliability-related parameters for deduplication-based storage systems: either the criticality of losing a chunk (RCR) or the probability of losing any chunk (DTR). Different from these schemes, PFP scheme improves the reliability by encoding the data chunks of a file before they are deduplicated. Thus, PFP provides the best reliability by considering the two reliability-related parameters simultaneously via intra-file recovery and inter-file recovery. Taking the storage efficiency into account, PFP is still shown to prevent the highest number of files from being lost per added data redundancy, for the four data sets. Moreover, the three schemes perform comparably in terms of system throughput.

Some studies also analyze the reliability of deduplication-based storage systems theoretically. Li et al. [35] propose combinatorial analysis of deduplication and erasure coding to evaluate the system reliability. Rozier et al. [36], [37] 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 analysis, 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.

7 Lessons Learned

The idea of Per-File Parity is motivated by the need to simultaneously protect individual files and provide a higher-level protection for critical data chunks with high reference counts in deduplication-based storage systems. Interestingly, our design and implementation of PFP to meet this increasingly important need reveal several useful insights and demonstrate that data deduplication is a double-edged sword for system reliability.

First, the impact of deduplication on reliability of storage systems is complicated. Deduplication is widely deployed in backup and archiving environments to significantly shorten

the backup window and save storage space. It reduces the storage footprint and the cost and number of storage devices, thus offering a potential to reduce the possibility of data loss [11]. Moreover, data deduplication significantly improves the reliability of flash-based storage systems by virtual of reducing the write traffic to the devices and thus reducing the number of internal write and garbage collection (GC) operations [38], [39] that are harmful to the reliability and performance of the flash device. On the other hand, keeping only a single instance for each data chunk that is shared by many files poses a serious threat to the reliability of deduplication-based storage systems, which has been demonstrated by our experimental results in Section 5.2 and previous studies [11], [31], [32]. Thus, a systemic reliability analytical study for deduplication-enabled HDD-based and SSD-based storage systems is needed.

Second, higher storage reliability can be achieved by applying erasure codes [40]. With the increasing processing power of CPUs in host machines and within device controllers, erasure codes have been well utilized in storage systems with acceptable processing overhead. Besides the parity and replication redundancy that are traditionally used in RAID-structured storage systems, erasure codes can tolerate two or more data chunk failures within a code group (i.e., codeword). Thus, incorporating erasure codes into the PFP scheme is a direction of our future work, which can provide higher levels of protection and avoid file losses caused by multiple concurrent data chunk failures within a code group.

Third, how to manage fingerprint information and other metadata is non-trivial in deduplication-based storage systems [41]. In the PFP scheme, in order to support inter-file recovery, the files sharing high-reference-count data chunks should be managed through a separate mapping table. When lost data chunks cannot be recovered by the intra-file recovery procedure, the mapping table is used to recover the data chunks with high reference counts by inter-file recovery. Although the inter-file recovery module can be embedded into the background disk scrubbing process to reduce the metadata overhead, it will degrade the system performance.

8 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 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.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. U1705261, No. 61872305, No. 61772439, and No. 61472336, the US NSF under Grant No. CCF-1704504 and CCF-1629625.

References

[1] J. Tucci, "Cloud + big data = massive change, massive opportunity, keynote at UW CSE annual industrial affiliates meeting," (Oct. 2010). [Online]. Available: http://photos.cs.washington.edu/?29-DLS-2011/1-Tucci/

[2] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1–30, 2014.

[3] P. Shilane, R. Chitloor, and U. Jonnala, "99 deduplication problems," in *Proc. 8th USENIX Conf. Hot Topics Storage File Syst.*, Jun. 2016, pp. 86–90.

[4] 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," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016.

[5] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. 6th USENIX Conf. File Storage Technol.*, Feb. 2008, Art. no. 18.

[6] D. Frey, A. Kermarrec, and K. Kloudas, "Probabilistic deduplication for cluster-based storage systems," in *Proc. 3rd ACM Symp. Cloud Comput.*, Oct. 2012, Art. no. 17.

[7] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2012, p. 4.

[8] J. Paulo and J. Pereira, "Efficient deduplication in a distributed primary storage infrastructure," *ACM Trans. Storage*, vol. 12, no. 4, pp. 1–35, 2016.

[9] H. Wu, S. Sakr, C. Wang, L. Zhu, Y. Fu, and K. Lu, "HPDedup: A Hybrid Prioritized Data Deduplication Mechanism for Primary Storage in the Cloud," in *Proc. 33rd Int. Conf. Massive Storage Syst. Technol.*, Jun. 2017, pp. 1–14.

[10] C. Ng, M. Ma, T. Wong, P. P. C. Lee, and J. C. S. Lui, "Live deduplication storage of virtual machine images in an open-source cloud," in *Proc. ACM/IFIP/USENIX Int. Conf. Distrib. Syst. Platforms Open Distrib. Process.*, Dec. 2011, pp. 81–100.

[11] M. Fu, P. P. C. Lee, D. Feng, Z. Chen, and Y. Xiao, "A simulation analysis of reliability in primary storage deduplication," in *Proc. IEEE Int. Symp. Workload Characterization*, Sep. 2016, pp. 1–10.

[12] I. Iliadis, R. Haas, X. Hu, and E. Eleftheriou, "Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2008, pp. 241–252.

[13] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2007, pp. 289–300.

[14] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *Proc. 14th USENIX Conf. File Storage Technol.*, Feb. 2016, pp. 67–80.

[15] F. D. A. Duggal, P. Shilane, T. Wong, S. Yan, and F. Botelho, "The logic of physical garbage collection in deduplicating storage," in *Proc. 15th USENIX Conf. File Storage Technol.*, Feb. 2017, pp. 29–43.

[16] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2015, pp. 177–190.

[17] G. Wang and W. Xu, "What can we learn from four years of data center hardware failures?" in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, Jun. 2017, pp. 25–36.

[18] K. Jin and E. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proc. SYSTOR: The Israeli Experimental Syst. Conf.*, May 2009, Art. no. 7.

[19] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, Jun. 2011, p. 25.

[20] S. Wu, H. Jiang, and B. Mao, "IDO: Intelligent data outsourcing with improved RAID reconstruction performance in large-scale data centers," in *Proc. 26th Int. Conf. Large Installation Syst. Administration: Strategies Tools Techn.*, Dec. 2012, pp. 17–32.

[21] J. Li, X. Ji, Y. Jia, B. Zhu, G. Wang, Z. Li, and X. Liu, "Hard drive failure prediction using classification and regression trees," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, Jun. 2014, pp. 383–394.

[22] B. Schroeder, S. Damouras, and P. Gill, "Understanding latent sector errors and how to protect against them," in *Proc. 10th USENIX Conf. File Storage Technol.*, Feb. 2010, pp. 71–84.
[23] Q. Xin, E. Miller, T. Schwarz, D. Long, S. Brandt, and W. Litwin, "Reliability mechanisms for very large storage systems," in *Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Syst. Technol.*, Apr. 2003, pp. 146–156.
[24] Deduplication based filesystem, (2017). [Online]. Available: https://github.com/opendedup/sdfs
[25] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian, "Read performance optimization for deduplication-based storage systems in the cloud," *ACM Trans. Storage*, vol. 10, no. 2, pp. 1–22, 2014.
[26] B. Schroeder and G. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *Proc. 5th USENIX Conf. File Storage Technol.*, Feb. 2007, Art. no. 1.
[27] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conf. File Storage Technol.*, Feb. 2007, pp. 2–2.
[28] The linux kernel archives, (2017). [Online]. Available: https://www.kernel.org/
[29] The FireFox install images, (2017). [Online]. Available: https://www.mozilla.org/en-US/firefox/new/
[30] VMware virtual appliance marketplace, (2017). [Online]. Available: http://www.vmware.com/appliances/
[31] C. Liu, Y. Gu, L. Sun, B. Yan, and D. Wang, "R-ADMAD: High reliability provision for large-scale de-duplication archival storage systems," in *Proc. 23rd Int. Conf. Supercomput.*, Jun. 2009, pp. 370–379.
[32] 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 *Proc. 14th IEEE Int. Symp. Model. Anal. Simul.*, Sep. 2006, pp. 413–421.
[33] B. Mao, H. Jiang, S. Wu, and L. Tian, "POD: Performance oriented I/O deduplication for primary storage systems in the cloud," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May. 2014, pp. 767–776.
[34] 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 *Proc. 8th USENIX Conf. File Storage Technol.*, Feb. 2010, pp. 17–17.
[35] X. Li, M. Lillibridge, and M. Uysal, "Reliability analysis of deduplicated and erasure-coded storage," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 3, pp. 4–9, 2011.
[36] E. Rozier and W. Sanders, "A framework for efficient evaluation of the fault tolerance of deduplicated storage systems," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, Jun. 2012, pp. 1–12.
[37] E. Rozier, W. Sanders, P. Zhou, and N. Mandagere, "Modeling the fault tolerance consequences of deduplication," in *Proc. IEEE 30th Int. Symp. Reliable Distrib. Syst.*, Oct. 2011, pp. 75–84.
[38] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. 9th USENIX Conf. File Stroage Technol.*, Feb. 2011, p. 6.
[39] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND flash-based SSDs," in *Proc. 9th USENIX Conf. File Stroage Technol.*, Feb. 2011, p. 7.
[40] J. S. Plank, "Erasure codes for storage systems: A brief primer," *login: The Usenix Mag.*, vol. 38, no. 6, pp. 44–50, 2013.
[41] F. Botelho, P. Shilane, N. Garg, and W. Hsu, "Memory efficient sanitization of a deduplicated storage system," in *Proc. 11th USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 81–94.

Suzhen Wu received the BSc and PhD degrees in computer science and technology and computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2005 and 2010, respectively. She is an associate professor of Computer Science Department, Xiamen University since August 2014. Her research interests include computer architecture and storage system. She has more than 40 publications in journal and international conferences including the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, the *ACM Transactions on Storage*, USENIX FAST, LISA, ICS, ICCD, MSST, IPDPS, SRDS, and DATE. She is a member of the IEEE and ACM.

Bo Mao received the BSc degree in computer science and technology from Northeast University, Shenyang, China, in 2005; and the PhD degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2010. His research interests include storage system, cloud computing and big data. He is an associate professor with the Software School of Xiamen University. He has more than 40 publications in international journals and conferences including the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, the *ACM Transactions on Storage*, USENIX FAST, LISA, ICS, ICCD, MSST, IPDPS, SRDS, and DATE. He is a member of the IEEE and ACM.

Hong Jiang received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China; the MASc degree in computer engineering from the University of Toronto, Toronto, Canada; and the PhD degree in computer science from Texas A&M University, College Station, Texas. He is currently chair and Wendell H. Nedderman Endowed professor of Computer Science and Engineering Department at the University of Texas at Arlington. Prior to joining UTA, he served as a Program director at National Science Foundation (2013.1-2015.8) and he was at University of Nebraska-Lincoln since 1991, where he was Willa Cather professor of Computer Science and Engineering. He has graduated 16 PhD students who upon their graduations either landed academic tenure-track positions in PhD-granting US institutions or were employed by major US IT corporations. He has also supervised 20 post-doctoral fellows and visiting scholars. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, performance evaluation. He recently served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 300 publications in major journals and international conferences in these areas, including the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *Proceedings of IEEE*, the *ACM Transactions on Architecture and Code Optimization*, the *ACM Transactions on Storage*, the *Journal of Parallel and Distributed Computing*, the *International Science Community Association*, MICRO, USENIX ATC, FAST, EuroSys, SOCC, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF, DOD, and industry. He is a fellow of the IEEE, and member of the ACM.

Huagao Luan received the master's degree from Computer Science Department, Xiamen University, Fujian, China, in 2018. He is currently a software engineer at Meituan-Dianping Company. His research interests include data deduplication and flash storage systems.

Jindong Zhou received the BSc degree in software engineering from Xiamen University, Fujian, China, in 2018. He is currently working toward the master's degree in software engineering at Xiamen University. His research interests include flash storage systems and data deduplication. He has publications in the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* and SRDS.